

Taking Advantage of Distributed Multicast Video to Deliver and Manipulate Television

by

Constantine Kleomenis Christakos

Submitted to the Program in Media Arts and Sciences, School of
Architecture and Planning

in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

Massachusetts Institute of Technology

February 2001

©MIT, 2001. All rights reserved.

Author _____

Program in Media Arts and Sciences, School of Architecture and
Planning

January 19, 2001

Certified by _____

Dr. Andy Lippman
Senior Research Scientist
MIT Media Laboratory
Thesis Supervisor

Accepted by _____

Stephen A. Benton
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Taking Advantage of Distributed Multicast Video to Deliver and Manipulate Television

by

Constantine Kleomenis Christakos

Submitted to the Program in Media Arts and Sciences, School of Architecture and
Planning

on January 19, 2001, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

Protocols exist to scalably multicast data to any number of clients. Scalability problems exist because some systems unacceptably increase the load on the server as more clients are added. Scalability can be achieved by hierarchically partitioning the network and by handling error correction between clients. One application for network multicasting is the distribution of video in the same way that television is delivered, replacing the standard antenna or cable service with a digital network using IP. However, clients using multicast implementations, such as the Scalable Media Delivery System, are not equipped to handle display of video in real time. Flaws in the current multicast implementation are identified, and methods of optimization are explored and tested for effectiveness. These optimizations will allow video to be displayed in real time over a network and to appear to the end-user to act like a cable TV system. Accessing the video on clients is done with an innovative interface that allows the user to “carry” the video with him as he travels to different physical locations in the network.

Thesis Supervisor: Dr. Andy Lippman

Title: Senior Research Scientist, MIT Media Laboratory

**Taking Advantage of Distributed Multicast Video to Deliver
and Manipulate Television**

by

Constantine Kleomenis Christakos

The following people served as readers for this thesis:

Thesis Reader _____

V. Michael Bove, Jr.
Principal Research Scientist
MIT Media Laboratory

Thesis Reader _____

Walter Bender
Senior Research Scientist
MIT Media Laboratory

Acknowledgments

This thesis was a long time in the works and beset by many stops and starts along the way. Without the help of several people inside and outside the Media Lab, this thesis would not have become a reality.

I would like to thank Andrew Lippman, my advisor, who patiently waited through many months of failed demos and stalled attempts at putting this multicast system together.

Jacky Mallett was immensely helpful in providing me with pointers and ideas to resolve various problems putting together this system along with giving me useful snippets of code that have made my life much easier.

Dimitrios Vyzovitis and Ying Lee were invaluable in implementing the vision of a video-navigation system that used laser pointers.

Finally, the UROPs I worked with over the past year have truly demonstrated a desire to go above and beyond the call to duty when it came to making our dreams a reality. Casey Muller, Cynthia Johanson, and Jessica Baker have been great contributors to the project.

Contents

1	Background	11
1.1	Introduction	11
1.2	Using Multicast to Deliver Video	12
1.3	Previous Work in Multicasting	14
1.4	Multicasting and Distributed Video	18
1.5	New Interfaces for Video	19
1.6	Use as a file sharing system	20
2	Current Multicast Systems	22
2.1	Background	22
2.2	Smart Network Caches	24
2.3	Video Encoding and Decoding	24
2.4	Data Repair	26
2.5	Dynamic Sessions	26
2.6	Optimizing the Current System	27
2.7	Optimization Methods	28
2.8	Results of Optimization	30
2.9	Summary	30
3	Design of New Multicast System	31
3.1	Introduction	31
3.2	Allowing Listeners to Tune in Late	31
3.3	A Late Join Scenario in SMDS	35

3.4	Creating Dynamic Sessions	36
3.5	Playing the Video	38
3.6	Comparison to Other Multicast Systems	39
3.7	Comparison to Other File Sharing Systems	42
4	New Interfaces for Video	45
4.1	Introduction	45
4.2	New Interfaces Created for SMDS	46
4.3	Summary	50
5	Extending SMDS	52
6	Conclusions and Further Work	54
6.1	Conclusions	54
6.2	Further Work	55

List of Figures

2-1	How the multicast modules and MPEG-decoding modules interact. They both access the same file, but the processes are independent . . .	25
3-1	An overview of how listeners interact with multicast channels	32
3-2	An overview of Chaining	34
3-3	An overview of how listeners interact in the multicast system	35
3-4	The Handshaking Protocol for Requesting a Dynamic Session	36
3-5	Relative throughput of multicast systems with different error correction schemes	39
3-6	NACK suppression, part 1: Only one listener sends a NACK even if both listeners have experienced a loss	40
3-7	NACK suppression, part 2: repairs arrive from the server. As more clients are added, the potential for more NACK packets rises	40
3-8	In SMDS, clients can make the packet repairs	42
4-1	The video tape (left) stores an entire copy of the video that the user carries from client to client. The encoded pen (right) only stores a representation of the video that is transmitted to different parts of the network as the user carries the pen from client to client.	47
4-2	A depiction of a video following scenario and protocol	48
4-3	Communication between modules in the laser target control system	50

List of Tables

3.1	Structure of the Dynamic Session Request Packet. Each row represents 4 bytes	37
3.2	Structure of the Dynamic Session Bid Packet. Each row represents 4 bytes	37
3.3	Structure of the Dynamic Session Bid Accept Packet. Each row represents 4 bytes	37
5.1	Structure of an IOC packet with specifies a session ID and multicast channel	53

Chapter 1

Background

1.1 Introduction

One potential future for the television industry is that it takes on the characteristics of publishing. That is to say, visual material is available on demand, is drawn from inventory, is provided by companies that have both blockbusters and a large backlist, and where the output becomes owned by the consumers. Available devices and networks lead in this direction: TiVo is the bookshelf, rental outlets are the bookstores, networks are the delivery chains, and DVD is a publishing medium.

This has structural implications for the industry, social impact on viewers, and technology implications required to support it. This thesis propels the technology towards a networked vision of that publishing model. Specifically, I implement a community distribution system for video where the library is shared among an unlimited, connected group of viewers. I build this on a backbone of internet multicast. The particular multicasting system proposed solves performance and reliability problems associated with growth and large scale by distributing video program bits among the entire community that has "tuned in" to a set of programs. The work, therefore, consists of the following three elements:

- A distribution system where bits are shared
- A client that displays those bits where and when you want

- An interface that facilitates navigation through the community of bits in ways sympathetic to the nature of the content (a video bookstore, if you will)

A side effect of the work is a file distribution system that allows each user to receive a file at his whim and ultimately be guaranteed a complete version of the file.

The work builds on progress in three areas: internet multicast, an in-house protocol for networked video called HPRTP (Hierarchically Partitioned Reliable Transport Protocol), and the combination of personal gizmos with entertainment systems.[Lippman2001] This system is known as the Scalable Media Delivery System (SMDS).

Chapter 2 describes current multicast work at the Media Lab and what was necessary to create a more robust system for distributing video in a manner consistent with the experience of viewing television. At the same time, it covers the issues involved in rebuilding the multicast system to run using PC-based computers running the Linux operating system as we moved away from the use of DEC Alpha-based workstations and optimizing the system performance.

In Chapter 3, this thesis discusses how new protocols were created to allow users to tune in late and compares this scheme to other alternate multicast protocols that have been developed.

Chapter 4 examines how distributed digital video opens the door to experiment with new interfaces to video browsing and archiving.

Chapter 5 examines how SMDS can be expanded and how further functions can be added to allow the listeners to collaborate in different ways.

Finally this work discusses new directions in which to take this research to create a robust and fully developed digital television system in which users can take full advantage of the medium.

1.2 Using Multicast to Deliver Video

Internet multicasting allows multiple recipients to simultaneously receive data without forcing the server to make a separate individual connection to each listener. Rather than broadcasting the data to all possible listeners on an internet network, internet

multicasting sends data only to the subnets of subscribers who have decided to tune into a given channel. Thus, if a subnet contains no active listeners, then routers will not forward multicast packets to that particular subnet. A model like broadcasting is implemented over the internet without flooding the entire network. Multicasting is a facility now built into IP, and attempts have been made to design new multicast systems to distribute data to an audience as wide as a television viewing audience. However, the most commonly available multicast schemes are insufficient.

Delivering video to a large audience depends on a few necessary attributes. The first of these is *timeliness*. When a user requests video, the data containing the beginning of the program must arrive first, followed by the middle and the end. This attribute of *timeliness* is not necessary for all types of files. After all, a data file or executable file could conceivably arrive in any order and simply be reconstructed after all the packets arrive. However, for applications such as video and audio, as soon as the data arrives, the user expects to start viewing it in order.

The next necessity in delivering video is an amount of *relative reliability*. While not every packet may arrive, and a few may arrive out of order, the viewer must be generally assured that he receives most of the transmission. In the analog world of video-viewing, a certain amount of static or signal-degradation is considered acceptable, but viewers expect to have a reasonable picture come through on their televisions.

Finally, a video-delivery system must be *scalable* such that the network and other servers and listeners are not adversely affected by the addition of new listeners who tune in. In fact, SMDS depends on having large numbers of listeners to take full advantage of media sharing between listeners, but this will end up forcing us to adopt a more stringent reliability requirement.

It is not enough for a server to simply send out packets onto the network in the hope that they will be received and reconstructed by anyone listening. An efficient video delivery system capable of serving a large number of listeners must fulfill certain requirements of timely delivery, reliability, and overall scalability of the system. The examples of other multicast systems demonstrate the tradeoffs made between these

attributes.

1.3 Previous Work in Multicasting

The multicast application that is most familiar to internet users is the Multicast Backbone, commonly known as the “MBone.” The MBone provides multicast media on a regular basis to users who have installed the MBone software. The MBone clients are notified of new multicast channels as they are announced.

Usually, MBone data is an audio or video stream that is delivered via UDP packets. There is none of the bandwidth control that is available with TCP packets, nor is there any error correction. In general, this is not a severe problem, since the occasional loss of a UDP packet in an audio or video stream is usually acceptable compared to the overhead of TCP error correction and packet retransmission. In fact, UDP is more analagous to a television or radio broadcast model since a broadcaster, like a server sending UDP packets, continues to send without concerning itself with whether the data is arriving correctly and without any feedback from any of the listeners.

The MBone also acts as an application-level layer that makes point-to-point connections across domains to deliver multicast packets to users if their internet service provider does not grant access to normal multicast services. This results in many scalability issues in making separate connections to route the packets to the registered multicast receivers.

The MBone is only a small portion of network traffic on the internet today. Most video is distributed over a standard “point-to-point” connection where a listener makes a specific request which is fulfilled by a specific server. Of course, in a point-to-point scenario, as more users make simultaneous requests, the load on the server increases and consumes the bandwidth on the server’s outgoing internet connection. These problems increase linearly as more listeners make requests from the server, so this obviously violates any requirements we may have for scalability and cannot be considered as an option to deliver video to as wide an audience as that of television. In fact, an attempt to deliver video to a wide audience in this manner was a spectacular

failure when Victoria's Secret publicized an "Internet Fashion Show" during the 1999 Super Bowl. When the date of the web-cast fashion show arrived, the video server immediately received millions of simultaneous requests and promptly crashed, forcing viewers to catch the video days later, defeating the purpose of staging a live video broadcast event. [Nickell99]

In some cases, reliability *is* desired when broadcasting to a large audience. One example is a shared whiteboard application[Floyd97] where senders *do* want to ensure that all listeners are receiving the same data and receiving it accurately. Without error correction, the MBone cannot make these assurances. The MBone is neither reliable nor scalable. As will be discussed, the most basic attempts at reliable multicast cause scalability problems.

Many different video delivery schemes have trouble when deployed outside of small local networks. Traditional multicast schemes have scalability problems because, inevitably, some packets are lost on their way to the listener. TCP-based error correction, with sender-initiated corrections and demands for individual packet acknowledgements, does not serve the needs of a broadcast model. To enforce some level of reliability in UDP, receiver-initiated reliable multicast protocols are created. To recover lost packets, listeners may send Negative Acknowledgement (NACK) packets to the server. However, packet-loss is bursty, and it is possible that many listeners will simultaneously lose packets and simultaneously make requests to the server for these lost packets. Network performance will degrade as more and more clients swamp the network and increase the burden on the server. These sorts of multicast solutions will suffer from *NACK implosion*,[Macker96] which will cause network congestion and force the server to suffer an additional load as more listeners send NACK packets.

By partitioning off requests for lost data, this congestion can be decreased. A chief node in a network tree can decide to be responsible for requesting all error corrections in his partition. Even if five leaves in a network partition are missing a packet, only the "leader" of the partition will make a NACK request to the server. This reduces the amount of bandwidth required to request packet corrections, though it requires topology knowledge of the network.[Papadopoulos98]

In addition, Forward Error Correction (FEC) schemes can provide “parity packets” to reconstruct lost packets without forcing clients to send NACKs back to the server. To distribute a file with FEC-based error correction, a file is broken up into groups of k packets each. These groups are encoded into groups of n packets, where $n > k$. The encoding scheme allows each original group to be decoded with any k packets from the encoded group of n . In a simple example of FEC-based error correction, a file can be broken into groups of 8 packets each. If each group carries with it a 9th parity packet, then receivers will still be able to reconstruct the entire group if they only receive any 8 out of the 9 packets in the group. For example, if the parity packet is lost, then the 8 original packets remain. If any of the 8 original packets are lost, then the parity packet can be used to reconstruct the contents of the lost one in the original group.

Different multicast systems will seek to maximize different attributes of the system. The MBone, for example, seeks to emphasize the timely arrival of data, without regard to reliability. On the other hand, The Fcast system [Gemmell00] reliably multicasts files to users and reconstructs the file exactly in its entirety using FEC-based error correction. It uses multicast to distribute file updates to end-users. Reliability is key here, and network bandwidth is saved by re-multicasting the file again and again to handle the recovery of “late join” data or any other packets that could not be recovered via FEC. It does not provide for any NACK-based error correction, and late joiners must wait until the next re-multicast by the main server to pick up lost data from the beginning. Thus, a user will not have access to the start of the data immediately if he misses the beginning of the multicast.

In addition, as mentioned above, packet losses typically appear in bursts on the network. This could cause an entire group to be lost which cannot be reconstructed. Fcast interleaves the group by sending out all packets with index i in each group before sending packets with index $i+1$ for each group. In the event of bursty packet loss, a listener will likely only lose one packet from several different groups, and each group will be able to be reconstructed. Without interleaving, a listener will be more likely to lose several packets from the same group, preventing that group from being

reconstructed with the parity packets. The bandwidth overhead of parity data and the requirement for late joiners to wait for retransmission can be reduced to as little as 20% above the original size of the file. However, data will still not arrive in order, even if the user is fortunate enough to tune into the multicast at the very beginning. The user will have to wait for the system to reconstruct the file in order after all the data has arrived. Fcast may provide scalable file distribution, but it fails to satisfy the timeliness requirement when delivering video.

Another example of a multicast architecture that will maximize the reliability of data is a stock exchange application. The Swiss Exchange uses an all-electronic stock exchange and is implemented using multicast technologies. [Birman99] However, the system is not designed to handle a large number of users. This is not a problem, as the system is built to handle, at most, 1000 active traders. It is considered acceptable to enforce the reliability model by making the system temporarily unresponsive to users who cannot be served because of the load.

Performance degrades as network congestion increases. In the Swiss Exchange example, network congestion is going to be caused by enforcing reliability conditions. For example, if a listener needs to make up for any lost data, the listener will request the data to be replaced by the server. This request will be sent through the network responsible for delivering the multicast data to all listeners. The request will then be fulfilled by sending the replacement data back to the requester while at the same time continuing its multicast to all other listeners. Clearly, as more listeners are added and more listeners make simultaneous requests for lost data, network congestion will increase, and performance will degrade in the same way performance over analog lines declines as noise enters the system. However, the current project here seeks to deliver video media to a large audience of users. The strict reliability of a stock exchange is not necessary, but being able to serve thousands of recipients in a manner consistent with the viewers' experience with television is a necessity.

Some multicast systems try to enforce strict reliability to a small audience. Others attempt to faithfully adhere to scalability while sacrificing the ability of users to receive the video in a timely fashion. We seek to construct a system that can deliver

the video to a large audience reliably right as the users request it.

1.4 Multicasting and Distributed Video

With a scalable multicast scheme capable of delivering video, television entertainment can be distributed digitally. Digital multicast video has several advantages over an analog broadcast. The broadcast paradigm sends a selection of channels to all users who pick which channel they choose to tune in to. With a multicast system, several users can choose from a much wider range of programming choices and view the program simultaneously. A multicast system can support a much wider array of choices because the network bandwidth is not consumed by being forced to carry every single channel over the same pipeline.

Having access to television content that is stored and distributed digitally provides more flexibility to the user. Products like TiVo and Replay TV already cache programs in a digital format that the user selects. Thus, the programs can be instantly accessed and viewed in a time-shifted fashion, and users are allowed to rewind and pause the program while it is in progress. To take advantage of this flexibility, one must buy one of these digital recording units that receive analog broadcasts. Unfortunately, these recording units are “stand-alone” and cannot communicate with other units. Multicast video starts with media in digital format and makes the video available from any location on the network, regardless of where it is physically stored. Also, the digital media itself can be multi-layered and can be viewed independent of the display, since the way the bits are decoded is independent of the network. The user gains not only the ability to watch time-shifted video that can be manipulated but also additional flexibility regarding how the media is viewed. Multicast video can be displayed on a television screen, in a video window on a computer monitor, or broadcast as sound-only through a portable audio player.

In addition, our project seeks to take advantage of file-sharing technologies which have become popular. Enough bandwidth and computational power is becoming available to most people that the distinction between the client and the server is

blurring. Filesharing systems such as Gnutella and Napster have attracted hundreds of thousands of users for the purpose of sharing digital media such as music. Our work attempts to create a community of video-sharing devices in which the shared video can be cataloged and downloaded by any other member of the community. The videos are distributed through the multicast protocol allowing others to tune in after another individual has made the original request. The community can either be a model in which many users in the community observe large numbers of channels being created or a model in which one user controls all of the listeners in a community. In the latter case, the user allows the video in progress to “follow” him as he traverses the community of listeners. However, since the file will be multicast and re-multicast, it must not degrade with successive transmissions. Thus we go from a requirement of *relative reliability* to a requirement for *absolute reliability*, which SMDS seeks to achieve.

1.5 New Interfaces for Video

Instead of a finite number of channels being broadcast to everyone, as in the television model, the distributed multicast model assumes almost an infinite number of channels each being multicast to a limited number of listeners. Effectively having an infinite number of channels available means that new ways of browsing through or switching channels must be explored. Under normal means of viewing television, one browses by learning to use a machine such as a remote control, VCR, or TV set controls. Since the proposal changes the transmission medium of television delivery, it also experiments with changes in the browsing mechanism.

An analogy to this is the Web. The Hypertext Mark-up Language, HTTP protocol, and interfaces were a package that enabled new ways of thinking about distributed media. Similarly, the proposal creates a new community based around distributed video data and packages it with an innovative interface that enables new ways of thinking about the manipulation of video.

Our project for multicasting distributed video also proposes experimenting with

new interfaces for accessing this video. The video itself is distributed over the network, and thus traditional conceptions of moving video from place to place with a physical copy of it are no longer required, and tags acting as representations of the video can play the same role.

The use of physical metaphors has been used in other projects to represent and control media such as answering machine messages.[Smith95] More recently, “mediaBlocks”[Ullmer98] have been used to represent video clips without actually storing any video data. Another example is the “musicBottles” which represent different musical instruments that are played when a bottle is opened, though the bottles themselves do not contain any musical information. [Ishii99]

Using representations of video not only gives more technical flexibility by not requiring a copy of the video data itself to be in the token, but also creates a flexibility in the content, as well. A video tag could represent a specific program, such as “Star Trek Voyager: Episode #1,” which would be the same every time it was played, or it could represent an abstract program, such as “Today’s News”, which plays a different program each day but represents the same concept every time it is played, even though the precise content will change between uses.

These tokens are the innovative interface to go along with the enhanced flexibility that is provided by distributed video. We are addressing issues of television viewing for a generation that lives in a world where computers have already unified with traditional media such as television. Breadth of content is assumed in this scenario. Some combination of access and navigation is necessary. We propose to make the act of navigating this breadth of content usable, in the same way that a web browser makes accessing the web content usable.

1.6 Use as a file sharing system

Finally, SMDS reliably reconstructs video files for the end users and allows listeners to receive data from other listeners. This is a basis for a distributed file sharing system in which clients can collaborate in distribution.

SMDS shares many attributes with other peer-to-peer file sharing systems that are currently available. SMDS can be compared to server-based filesharing systems such as Napster, and more closely compared to more distributed filesharing systems such as Gnutella and Freenet. Later on, SMDS will be examined in more detail regarding its use in filesharing, along with other systems that use multicasting for file distribution.

Chapter 2

Current Multicast Systems

2.1 Background

Projects at the Media Lab have attempted to solve the scalability problem of network multicasting. The Hierarchically Partitioned Reliable Transport Protocol (HP RTP) was created to solve the scalability issue by handling error correction between clients and allowing the network to be partitioned in a hierarchy so that errors affecting many clients in the same geographical area could be handled by a single “leader” [Kermode98].

Scalability was achieved in HP RTP by accomplishing administrative scoping of error correction between clients. Within a scoped region, error correction packets are handled locally, and, as a result, the entire multicast network does not become flooded with NACKs and repair packets.

Furthermore, HP RTP supports the creation of “dynamic partitions” to provide “late-join data” when clients tune in late to a multicast session. Any client can fulfill a request for a dynamic partition, and this reduces scalability problems as more clients tune in late. The burden of providing this “late-join data” is not placed entirely on the server, and listeners can cooperate in fulfilling different parts of the dynamic partition that has been requested by a late joiner.

The first generation of the Scalable Media Delivery System (SMDS) used HP RTP to multicast audio and video to the clients.[Chen99] This achieved scalable broadcast-

ing of these media to clients and allowed clients to “tune in” to a multicast channel by receiving data from a neighboring client. However, while scalability issues affecting the server were solved through the use of this protocol, the clients were unable handle the display and playback of video in real time. The problem shifted from trying to solve issues of network traffic and server performance to dealing with the issue of constructing the clients to handle the influx of this traffic.

The Java-based SMDS allows every machine on the network to act as both a client and server, and it also provides protocols for announcing the existence of new multicast channels, making requests for those channels, and correcting errors. Error correction can be handled by either listeners or servers and can be partitioned by local subnet if this was desired.

Listeners and servers exchange session information over the Inter-Object Control (IOC) channel. This provides the listeners with the addresses of the multicast channels as well as the size of the session.

Sessions announced by an SMDS server give a channel name, size of the file being sent, and multicast channel information. The sessions contain “Logical Data Streams” (LDS), where each separate stream is carried over its own multicast channel to the receivers. These additional streams can be used for applications such as multi-layered video, where one LDS contains the “base layer” and the second LDS is an optional “enhancement layer.”

SMDS also specified support for dynamic partitions but this was never completely implemented. Instead, late joiners were able to watch the entire multicast stream from the beginning by automatically caching data that was being sent over multicast streams that it was aware of. However, SMDS has a structure that allows the creation of a dynamic session to be negotiated. Specifically, the IOC channel serves as a means of negotiating these dynamic partitions.

2.2 Smart Network Caches

By allowing clients to cache data received from a multicast stream, they can share their information with others without further impacting the original source of the multicast stream. The SMDS system provides for smart network caches to uniquely identify programs across the network, wherever they are stored.

SMDS, as deployed today, keeps the entire contents of the multicast file it receives in its cache. The multicast server breaks the file into packets and each client reconstructs that file in its own cache. The file is identified both by name and by a unique session ID number that the multicast listeners receive. SMDS uses a 32-bit identifier to uniquely identify files sent over the multicast system. In this case it uses Java's `HashCode()` function to map the filename to this 32-bit identifier. Further development of this project will require a naming and hash function that does not depend on a specific programming language.

When a client receives a request for a single packet or set of packets out of its cache, the client retrieves the required contents out of its cache and sends it out over the appropriate multicast channel.

2.3 Video Encoding and Decoding

There are many different video formats available that can be used for sending data over the network. If the video compression is efficient enough, then it can be sent through the network quickly, and playback appears sufficiently close to television-quality, then the system can be considered acceptable.

Since SMDS is effectively a means of multicasting files to all listeners, it runs independent of the sort of video encoding scheme being used. The system normally multicasts files in MPEG-1 format. These have been encoded off television broadcasts or taken directly off of Video CDs (VCDs) that carry MPEG-1 video. As these files stream over the network and get saved in the caches of the listeners that are tuned in, separate processes are responsible for playing the files. MPEG-1 video with

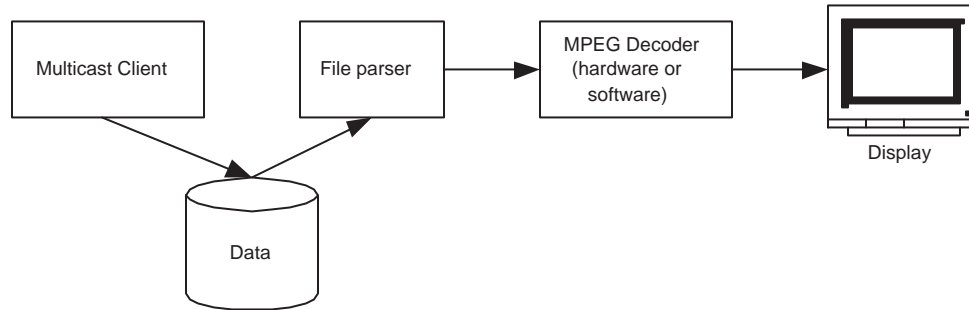


Figure 2-1: How the multicast modules and MPEG-decoding modules interact. They both access the same file, but the processes are independent

352x240 pixel resolution at 30 frames/sec has a bandwidth of approximately 1.5Mbps. [Chen99] This allows video of reasonable quality to be sent across the network fairly quickly.

As it was originally implemented, SMDS decoded these files with a player based on the MpegTV SDK. This played video in a small window on the Digital Unix desktop when SMDS was being run on the DEC Alpha machines. While this did not look at all like television, it demonstrated the ability to multicast video files around the network. The MpegTV SDK was also a poor performer on the DEC Alpha machines and made the video playback jerky and of poor quality. Profiling experiments indicated that a major contributor to the poor performance was the simultaneous reading and writing to disk while the video was being downloaded and played, as well as the overhead of software-based MPEG decoding. Since the MpegTV SDK was a separate add-on to SMDS, the player could be replaced with other decoders as they became available.

While the original SMDS implementation used MPEG-1 encoded video files and the MpegTV software decoder, neither of these specific solutions is an indispensable part of the system. The multicast protocols we have developed concentrate entirely on reconstructing files that originate at the multicast server on to the clients tuned in to the multicast channel. Files with other video formats can be multicast using SMDS, and the listeners can use players of their choice to display this video (Figure 2-1). Additional work on SMDS can make any necessary changes in the types of files being multicast or the sort of players being used to display the video, depending on

how needs change for the system.

2.4 Data Repair

SMDS is able to get repairs from both the multicast server and other listeners who are receiving the multicast stream. If repair requests are locally scoped, then scalability can be improved.

In traditional multicast systems, attempts to improve scalability in reliable systems are made by constructing NACK-based systems and implementing NACK-suppression within network partitions. This ensures that if a packet is lost within a sub-group, then only one listener will send a NACK, and when that listener's neighbors detect that NACK, they will suppress their own and wait for an incoming correction packet from the multicast server. Correction packets will be received by all of the listeners, although only one listener is required to report the loss of that packet.

In SMDS, each listener both sends out repair requests and listens for repair requests from others on the multicast repair channel. When it receives a repair request for a packet that it has already received, it sends out the requested packet on the repair channel. Listeners that have not yet received this packet use the packet to reconstruct the file being multicast. Repair requests can be locally scoped by limiting the size of the time-to-live (TTL) socket option on the repair request packets.

Allowing local listeners to fulfill repair requests improves scalability while making the repair process seamless to the requesters. SMDS accomplishes this by placing the same repair software on both the server and the listeners, so requests are fulfilled on equal standing by anyone.

2.5 Dynamic Sessions

When the idea of repairing a few sequential packets is taken to a larger scale, instead of making individual repair requests for all of those packets, a listener can request a

“dynamic session.”

In the SMDS specifications, a dynamic session requester negotiates to find a dynamic session provider over the IOC channel. The requester announces a need for a set of packets over the IOC channel and listeners that are able to fulfill this request reply with bids to do so. The requester accepts one of the bids and the provider begins sending over a pre-negotiated multicast channel. As the packets arrive, the requester fills in the missing packets that were originally requested.

Thus, it is possible for a listener to be receiving the main multicast stream, receiving a dynamic session stream over a second channel, and sending a dynamic session stream over a third channel, thus blurring the distinction between explicit clients and servers in the video system.

With dynamic sessions, data can be distributed not only reliably and scalably, but also in a timely fashion. This ensures that the user receives the beginning of the file as quickly as possible from one of his neighbors, regardless of the point at which he tunes in to the main multicast stream.

The role of dynamic sessions is parallel to that of repair requests, just on a larger scale. This is the final feature in SMDS that helps the reliability and scalability of the system, while at the same time allowing users to access the beginning of the file as quickly as possible. The latter feature is necessary because users are receiving video and want to watch the beginning of the program when they tune in, instead of waiting for a re-multicast from the server. Because clients will re-multicast the video they have received, absolute reliability is a requirement. Without absolute reliability, the video will slowly degrade as packet errors creep in with each successive re-multicast. However, since we now have an error-correction scheme which is both reliable and scalable, dynamic sessions are now feasible.

2.6 Optimizing the Current System

Video multicasting using the system developed at the Media Lab by Dean Chen [Chen99] exhibits unacceptably poor performance when displaying video. As men-

tioned in our discussion of playback performance, profiling experiments revealed that a major contributor to the poor performance was the simultaneous reading and writing to disk while the video was being downloaded and played. To replicate the experience of television, the system had to demonstrate much better performance in the video display.

Furthermore, moving the multicast system from an Alpha-based Digital Unix system to a Pentium-based Linux showed more performance problems. The code that implemented the multicast system had several simultaneously running threads that were responsible for ensuring that the data was received. While the system accomplished its goals when running under Digital Unix, the system could not successfully download multicast video when running under Linux.

The optimization problems were two fold: the video received in the old multicast system did not look like television once it was received, and the system did not run at all when it was moved to a Linux-based platform. Fortunately, the necessary optimizations were made, and they allowed the system to more fully replicate the experience of television.

2.7 Optimization Methods

While the motivations for optimizing the system stemmed from the initial poor video display performance, the fact that the system could not even successfully be used when moved to a Linux platform made it necessary to look at the basis of the multicast code itself.

Though Java makes claims of platform-independence – “write once, run anywhere” – the reality is that there are important differences that must be considered when running large Java-based systems on different platforms. Different versions between Java SDKs cause runtime differences in the Java code. Furthermore, threading is highly platform-dependent in Java, and this causes the multicast system to vary wildly when moving from a Digital Unix-based platform to a Linux-based platform. Experiments with Linux indicated that Linux is more susceptible to thread-locking

than Digital Unix, and coders creating systems that use many simultaneous threads must be more explicit about scheduling when developing for Linux.

Furthermore, Java is slow by nature. The intention in building the multicast system was to replicate television by sending video over IP networks, ensure that all the data was correct, and fulfill repair requests from other clients. Java is sometimes inadequate to fill this role. As an example, a Java-based multicast server was running during a test of the system while a receiving client was running with software written in C. When the client missed a packet, it sent out a repair request to the server. In the time it took the client to realize it missed a packet and send out a repair request, the server had not yet even realized that it had already sent out the packet that the client already detected as missing. Ultimately, to test out the integrity of the repair system, the client was forced to pause before sending out a repair request. This ensured that the server had updated its current state to reflect the fact that it had already sent out the packet that had ultimately gone missing.

The multicast client code was re-written entirely in C. All the data was successfully received by the clients over the multicast channel with the new C-based system and received much more quickly because there was less code overhead than there was in the old Java-based system. In retrospect, while Java's structure and cross-platform nature make it appear like an attractive choice with which to build a system for multicasting video, the slow performance of Java makes it a poor choice to satisfy the real-time needs of multicast video. Reimplementing the system in C turned out to be a natural and successful choice.

With a client that could successfully receive multicast data, the system was further improved by switching the MPEG decoding from a software-based decoder to a hardware-based decoder. The software was relieved of the burden of dealing with the MPEG decoding, which would have distracted it from receiving the data over the network. The decoder has a video output which allowed the video received to be shown on a separate video display. The improvement in both performance and presentation venue allowed the system to more accurately reflect the experience of watching television.

The MPEG decoder card used was the Stradis professional MPEG decoder. This is the only card that has drivers which allow it to run under the Linux operating system. Sending raw MPEG data to the Stradis card sends video to the composite video outputs in the card which can then be displayed on an external video monitor, such as a television. Under the Java-based SMDS running on the Alpha platform, the MPEG player is a separate application, `mtvp`, which handles MPEG decoding and the display of the video in an Xterm window. In the system running on Linux, the MPEG player software is only a very simple C program that writes successive bytes from the MPEG file out to the Stradis card, represented by the `/dev/video0` device. The decoding and video display is handled entirely in hardware, and the MPEG player application is only a very simple file reader that writes out to a socket. This requires much less software overhead than the original solution.

2.8 Results of Optimization

The end result of the reimplementing of the system is clear – the multicast client functions efficiently on the Linux-based platform, whereas the old Java-based system did not function at all. Furthermore, by switching to a hardware-based MPEG decoder, the system did not reflect the same playback-performance problems, and it actually reflected the television-viewing experience more accurately.

2.9 Summary

SMDS specifies a system with scalability, reliability, and timeliness. Optimizing the video delivery and playback performance allows users to have their viewing experience equal the playback performance of television. In the following chapter, we will examine how this system was implemented specifically to ensure that video delivery is scalable, reliable, and timely for the users.

Chapter 3

Design of New Multicast System

3.1 Introduction

Rewriting the old multicast system presented the opportunity to add necessary features that were previously unimplemented.

The multicast system was re-written to allow listeners to tune in late by asking for missing data from other clients. While tuned into a multicast session, a client looking for a large amount of missing data is able to request a “dynamic session” from another listener. A protocol for requesting these dynamic sessions from other listeners had to be developed.

Using these dynamic sessions for video-sharing between listeners means that as more listeners are added to the multicast community, the system does not collapse under its own weight, but rather allows many more opportunities for new listeners who join. Since new listeners can take advantage of data in the caches of others, an expanding library of video becomes available to all of the listeners as the client base expands.

3.2 Allowing Listeners to Tune in Late

Users in a multicast video system are placed in an unusual position. As with a television schedule, users are selecting from a schedule of multicast video programs

being sent by a server. At the same time, they are selecting individual *programs* that they wish to watch. When they select those programs, they expect the programs to arrive, even if they have tuned in late. The purest form of this is the standard web-based video-on-demand model in which users request specific videos and a unicast TCP/IP connection is set up between the client and the server. However, as the audience increases, larger and larger amounts of bandwidth are consumed. When the video is being multicast, one alternative proposed is *batching*[Chan98]. In this case, the server waits until a specific number of requesters has queued for a video. When this threshold is reached, the server sends a multicast stream to the queued listeners and waits until another group of listeners begins to queue up for the same program. Each time the threshold of queued listeners is reached, new multicast streams are created. While this is an improvement over creating separate connections for each user, it also consumes more bandwidth than necessary and forces listeners to wait for an indeterminate amount of time. It is possible that this delay could be so long as to make the system unusable.

An alternative is to start sending the multicast stream at a specified time no matter when users tune in. Those coming in late can pick up the main multicast stream. To allow those listeners to start from the beginning of the program, one solution is for the server to provide *patching* services. When the listener tunes into a multicast stream beginning at time n , the server also provides the listener with a separate stream from time 0 to $n-1$. While this has been shown to cause less bandwidth consumption than *batching*[Hua98], the method violates the *scalability* requirement outlined earlier. As more and more listeners tune in, the server's load will increase, as it will be required to serve more and more of these *patch streams*.

Another alternative which is not server dependent like patching is a method called *chaining*. In this scenario, late-joiners don't pick up the main multicast stream at all, but rather have each successive packet in the multicast stream forwarded to them by an earlier joiner.[Sheu97] This continues along in a chain of listeners who have joined at different times. (Figure 3-2) This model assumes that each listener has only a limited amount of space in which to store the video stream and that listeners tuning

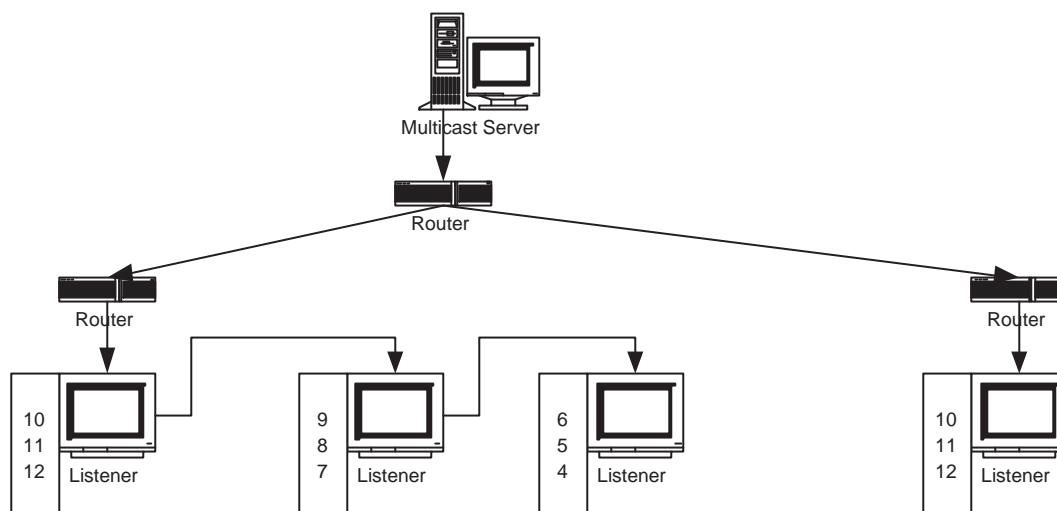


Figure 3-2: An overview of Chaining

in will only receive one stream at a time. In fact, the whole thing appears much like a peer-to-peer network of thin clients, with little storage or intelligence. The project's date – 1997 – leads one to suspect that the model was inspired by Oracle's evangelism of “network computers” at about that time.

In SMDS, the assumption is that all of the listeners have sufficient local storage to keep at least an entire program, if not several programs. Consumer-electronics devices are already capable of digitally storing several programs received from broadcasts. By creating clients with large caches that store incoming multicast programs, the clients could supply much longer patch streams. Since a late-joining listener can also pick up the main server's multicast stream, the later-joiner is not solely dependent on having a multicast forwarded to him, as in the chaining[Sheu97] example. Dynamic sessions could come from several different sources. The main multicast stream from the server would be only one part, and other parts could come from other listeners. If the late-joiner was in the same subnet as another earlier listener, then the dynamic sessions would not only be a more scalable solution over batching, but also save bandwidth across subnets (Figure 3-3).

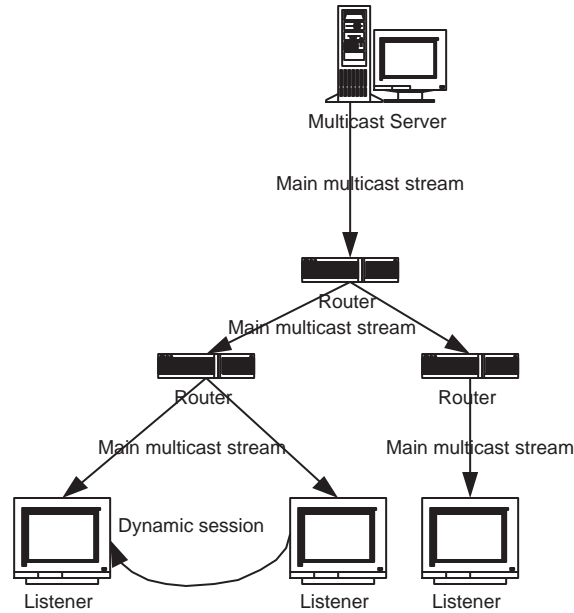


Figure 3-3: An overview of how listeners interact in the multicast system

3.3 A Late Join Scenario in SMDS

When a new video program becomes available, listeners receive announcements of its existence over the SAP channel and the IOC channel. When the first listener sends a request over the IOC channel for the program, the server begins sending packets over the data channel, and it fulfills repair requests over the repair channel. When other listeners tune in later, they will realize that the data packets they are receiving do not start with the first packet. When listeners start writing packets to disk that are out-of-sequence, the system fills in “blank” packets to make up for missing data while retaining the integrity of the file’s structure. For example, when the system realizes that the first packet being written to disk is packet #103, it will fill in packets 0-102 by writing blank packets out to disk as though they had been received and overwrite those blank packets later when these missing packets are finally received.

As other listeners tune in, they, too, may fulfill requests for missing data. However, at the same time, in the case of listeners who join late, they are responsible for recovering scores or hundreds of packets of lost data. This is more substantial than merely making some requests for individual lost packets over the repair channel.

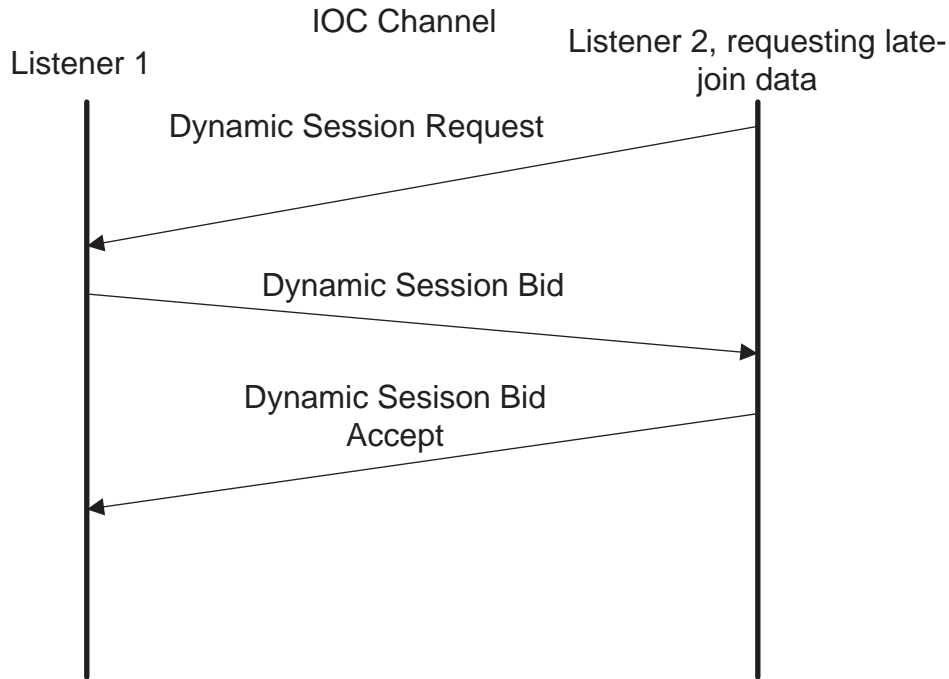


Figure 3-4: The Handshaking Protocol for Requesting a Dynamic Session

Instead, the listener can ask for another channel, or a “dynamic session” to be created.

3.4 Creating Dynamic Sessions

When a listener tunes in late to a multicast channel, it makes a request for a dynamic partition for the set of packets that it missed. The request packet is sent out over the IOC channel. This request specifies which session it needs “late join” data for, the range of packets needed, and the IP address of the requester. All machines on the multicast network listen on the IOC channel for Dynamic Session Requests. When machines receive a request for a dynamic session, the receivers check if they are currently receiving that session. Those that are not already fulfilling another dynamic session request then place a bid. A bid contains data that will be used to add another Logical Data Stream (LDS) to the current session to fulfill the dynamic partition. A new LDS will be constructed out of a new multicasted data channel and repair channel. In addition, the bid packet contains the IP address of the requester and the bidder, so that listeners on the IOC channel know for whom the bid is intended.

7	Reserved
LDS Id	
Start Packet Num	End Packet Num

Table 3.1: Structure of the Dynamic Session Request Packet. Each row represents 4 bytes

8	Reserved
IP of Bidder	
IP of Requester	
Session Id	
New LDS Id	
Original LDS Id	
Data Channel	
Repair Channel	
Data Port	Repair Port
Packet Size	Number of Packets
StartPacket	End Packet

Table 3.2: Structure of the Dynamic Session Bid Packet. Each row represents 4 bytes

The requester accepts the first bid it receives by sending out an IOC packet which contains the IP address of the bidder. Once the bidder receives the acceptance intended for it, the bidder sets up a dynamic session and begins sending the packets that were requested. While the IP addresses of the requester and bidder are specified within the dynamic session packets, any listener is eligible to tune in to this dynamic session and receive the packets.

The dynamic session is received by the listener(s) in the same manner as any other session. When the packets get written to disk, they replace the blank packets in the file that have already been placed there while the system was receiving packets from the main multicast stream. While a dynamic session in this case is being used for “late

9	Reserved
IP of Bidder	
IP of Requester	
Session Id	
New LDS Id	

Table 3.3: Structure of the Dynamic Session Bid Accept Packet. Each row represents 4 bytes

join” data, – the first N packets – in fact the structure allows us to create a dynamic session for any block of packets needed. A listener that received the beginning of the program, stopped, and then returned to the channel later could create a dynamic session to receive the missing block in the middle and pick up where the listener had left off when he stopped.

3.5 Playing the Video

The driver software for the Stradis Professional MPEG-2 decoder card comes with an application that parses MPEG files and streams the data out to the card so that it can be displayed on an external viewscreen. The MPEG parsing application was modified to change it into a library of functions that are used to play the video. The controlling process uses these functions to spawn off a playing process which sends the file data to the MPEG decoder. The controlling process can at any time execute functions that pauses the decoding of the data or can rewind the data and allow it to pick up at an earlier place. When the command to start parsing is given, the parsing process is forked off. The controlling functions and the parsing process share a byte of shared memory which is used as a flag to indicate when to pause. The functions that tell the parsing to pause or continue change this byte. Each iteration of the parser checks this flag and halts the parsing if the byte is set to pause.

The library of functions can be used to create a separate interface to control MPEG parsing and then create a graphical MPEG player. A graphical interface was created for the MPEG player using the Qt toolkit. When packets from a channel start to be received, the receiving process spawns a separate application that presents a graphical front end. Pressing the play and pause buttons causes a communicates with the functions used to play MPEG files. This can be integrated with more physical interfaces to manipulating the video, as will be described later.

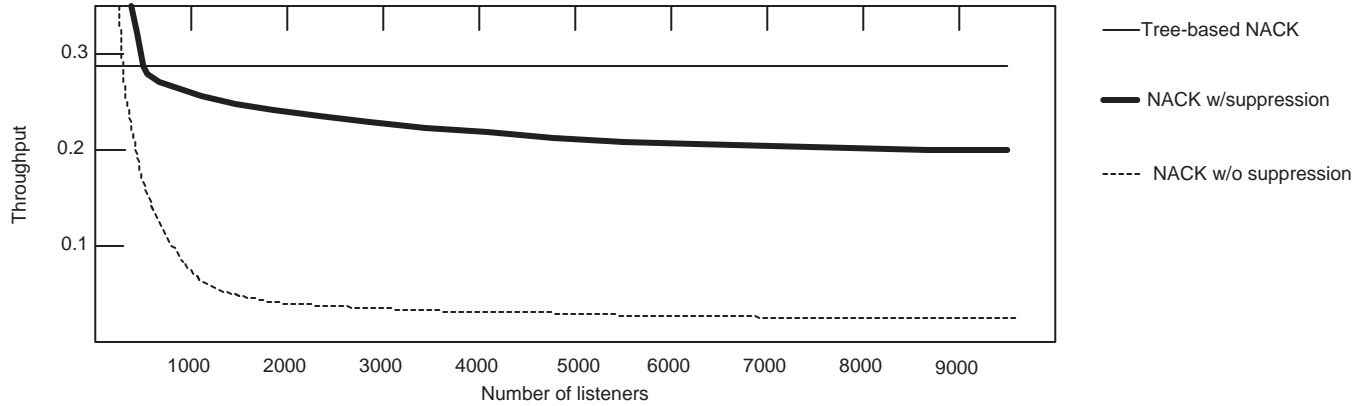


Figure 3-5: Relative throughput of multicast systems with different error correction schemes

3.6 Comparison to Other Multicast Systems

SMDS thus provides for scalable cooperation of clients for both small repairs and dynamic sessions that reconstruct large portions of the files. The multicast system is both scalable and reliable. In addition, it provides data starting from the beginning in a timely manner.

The MBone provided for neither timeliness, reliability, nor scalability. Late joiners could not make up for missed data, there were no provisions for error correction, and the unchecked consumption of bandwidth over the TCP “tunnels” prevented the system from becoming scalable. Additions to the MBone [Vicisano97], attempt to add reliability through FEC techniques and implement some forms of congestion control.

Other scalable reliable multicast schemes[Floyd97] use NACKs for error correction and network partitioning so that NACKs can be suppressed if they are lost within the same network partition. The hope is that suppressing NACKs reduces the number of NACKs sent to those at the head of a particular network partition. However, lost packets are still corrected by the server. The hope is only that there will be fewer NACKs than there would be otherwise. SMDS eliminates the server from the equation as much as it can by allowing other SMDS listeners to send out packet corrections.

These NACK-based schemes share with SMDS the use of network partitioning by

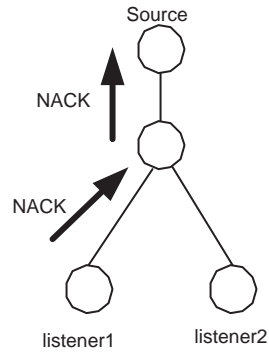


Figure 3-6: NACK suppression, part 1: Only one listener sends a NACK even if both listeners have experienced a loss

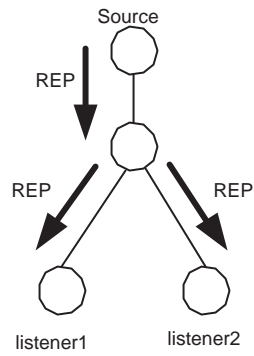


Figure 3-7: NACK suppression, part 2: repairs arrive from the server. As more clients are added, the potential for more NACK packets rises

specifying TTL (time-to-live) attributes for the repair packets. This way, repairs can be scoped within a local region. As shown by [Floyd95], as the number of listeners increases from 1 up to 500, local repairs can be contained within a neighborhood range of between 4 and 6 nodes. This demonstrates extremely good scaling which justifies the use of this means of partitioning local repairs in SMDS.

The graph in Figure 3-5 compares the relative throughput in multicast systems with different error corrections schemes. As shown, the error correction scheme that scales best as listeners are added is a NACK-based one in which the network is explicitly scoped through knowledge of network topology into a tree-based network and that uses periodic polling messages sent back to the server to acknowledge all earlier messages.[Ramakrishnan87] However, Floyd's[Floyd97] method of NACK-suppression with TTL-based scoping provides very good scalability as well without the administrative overhead of scoping out the network ahead of time. Without any NACK suppression or administrative scoping, throughput quickly collapses under the load as more listeners tune in.[Levine96]

Finally, the Fcast[Gemmell00] system handles reliability by depending entirely on FEC as well as repeatedly re-multicasting file data again and again. Any packets not fixed via FEC can be made up in subsequent multicasts of the file. Users tuning in late are forced to wait until the next re-multicast. Fcast makes the assumption that any NACK-based error correction will hurt scalability, while SMDS attempts to create a scalable system while still using NACK-based error correction. Use of NACK is essential in SMDS because there is no guarantee of an immediate re-multicast of the video from the server, as there is in Fcast. While Fcast can guarantee delivery with as little as 20% overhead over the original size of the file, data must be decoded and reordered by the listeners before it can be used, which is unsuitable for video. Finally, Fcast makes no provision for outsourcing error correction to other clients, as SMDS does.

SMDS combines the best scalable and reliable aspects of the other multicast systems. SMDS depends on data sharing between multicast listeners. In addition, it not only provides a reliable reconstruction of a file needed for file sharing, such as in Fcast,

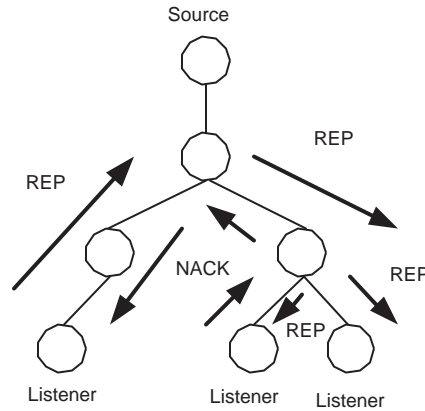


Figure 3-8: In SMDS, clients can make the packet repairs

but also allows users to tune in at the beginning of a program. This is desirable in a video broadcast, whereas it probably does not matter for simply file sharing where the data is going to be re-multicast immediately after the first stream ends. SMDS is distinctive because it provides support for dynamic sessions and client-based error correction and does not depend on re-multicasts from the server, as it does in Fcast.

3.7 Comparison to Other File Sharing Systems

A system like SMDS, which allows users to receive files reliably and send parts of files to other users on demand, looks suspiciously like a file sharing system. In fact, SMDS could be used as a file sharing system. When a “dynamic session” is requested by one of the listeners, the listener is “provoking” one of the other listeners into sending a multicast stream. This is analogous to the unicast file sharing scenario in which a client requests another client (in the case, acting as a server) to send it a file. In both cases, a requester “provokes” a response from another machine that provides the necessary data. The key problem is locating the necessary data. In SMDS, as described before, listeners request dynamic sessions over the pre-negotiated IOC multicast channel, and other listeners receive this request and decide whether or not to answer the request. This is conceptually similar to other de-centralized flesharing systems such as Gnutella and Freenet. On the other hand, more centralized flesharing

systems, such as those that use the Napster protocol, depend on a central database to direct the requester to a machine that has the needed file. Other projects have even created an entirely multicast-based file system. When SMDS is compared with file sharing systems, a better idea can be gained regarding whether it would itself function well as a file sharing system.

The Napster protocol has established itself as perhaps the most famous file sharing system. While it was originally conceived as a means of sharing MP3-encoded music files, people can run their own servers that use the Napster protocol to share any kind of files. Users connect to a server and tell the server database the contents of their file sharing directory— analogous to the SMDS cache. Users searching for files query the database and are returned a list of other users who have files matching their search are available. The searcher then sets up a point-to-point TCP connection between each other to receive the file from another logged-in user. The search process is centralized at the server, though users can access data from one of any number of other users connected to the server. Napster has encountered scalability issues with its servers becoming overloaded by connecting users.

The Gnutella protocol has established itself as an alternative to Napster and is distinguished by the fact that there is no central server. Since the search mechanism is completely decentralized, this lends itself to a comparison with SMDS. With Gnutella, search packets and responses are forwarded from client to client over the Gnutella network as the system tries to match a set of search terms. However, in the Gnutella network, the clients are responsible for forwarding these packets, and it causes problems for low-bandwidth clients who suddenly become responsible for running a large number of search requests through their relatively narrow pipes. On the other hand, SMDS solves this problem by using multicast, passing off the responsibility for forwarding packets to the routers, rather than depending on listeners to do this for them. Therefore, other low bandwidth listeners are not saddled with the additional obligation of being forced to forward on packets to others.

Freenet[Clarke99] implements a form of smart caching across the network. As requests are made, the network caches much-needed information closer to the place

where it is more commonly requested. It seeks to make file sharing more scalable. However, it still foresees a point-to-point scenario for distributing data. On the other hand, SMDS is specifically geared towards a simultaneous reception of the same data to a large audience, and Freenet attempts to cluster data effectively where it is needed most. Therefore, although a few individuals may access a certain file at any one time, the “large audience” simultaneously receiving data in Freenet could be considered to be the nodes that cache the data nearby to the most frequent users.

Instead of simple file distribution, multicast has also been used to implement a full-fledged file system. The JetFile[Gronvall99] project created a distributed file system that uses a form of reliable multicast [Floyd97]. Naming was handled by giving each file a unique multicast channel over which the data was distributed. Since this was not merely a means of file distribution but a distributed file system, much effort was expended on handling synchronization and keeping track of file versions as they were changed across the network. In comparison, SMDS focuses on distributing and sharing unique, read-only files, so it is not affected by this additional complexity that JetFile is forced to consider.

Chapter 4

New Interfaces for Video

4.1 Introduction

Having video distributed over the network allows us to create new interfaces to access to video. Having digital video sent to clients that can communicate with each other provides more flexibility. This flexibility allows us to explore different ways of experiencing video viewing. These new interfaces take the forms of different sorts of tags that make references to the video somewhere on the network, without actually storing any of it. The challenge becomes creating protocols to access the video and a means of keeping track of which tags refer to which video and delivering it to the user's location.

The use of physical metaphors to access digital data has been examined in other contexts. Roy Want[Want99] outlined an ambitious plan to connect physical objects with virtual representations through electronic RF tags. These tags could be associated with electronic documents or hyperlinks. As examples, Want used business cards whose tags would bring up the owner's web page and physical bookmarks whose tags would reference a particular document page.

Another tag-based system using metaphors was the InteractiveDESK[Arai95]. This used a camera to recognize color-coded tags on scrapbooks that were linked with a set of files. Instead of forcing the user to traverse through a directory hierarchy for his files, he could simply take out his scrapbook, place it on the desk, and

then select one of the several files associated with the scrapbook on his desk's video display.

Finally, the mediaBlocks[Ullmer98] project created physical icons (“phicons”) which function as “containers” of online media, such as video without actually storing the media internally. Like Want’s work[Want99], the blocks were embedded with electronic ID tags associated with media. Because they do not contain any specific media data themselves, the size of the actual media is irrelevant and they can refer to live streaming media. Furthermore, the blocks could be used as the interface to the video itself. Ordering the blocks side-by-side could be used to specify an order in which various video clips would be played back. Also, blocks could be manipulated within their slots to specify cut-and-paste operations. These operations would create a final video product on the media server, and this final product becomes associated with a specific mediaBlock.

All of these examples use a physical object and associate it with online data. In the case of the mediaBlocks and the electronic tags, RF IDs are placed on the objects to uniquely identify them, whereas the InteractiveDESK uses a camera to visually identify tag markers on the scrap books. SMDS envisions a scenario in which the “online data” is not in a specific place on a specific server. Rather, the data needed for SMDS is the name of a program that can be accessed *somewhere* on the network, either from the main multicast stream or from the cache of another listener. Taking advantage of the fact that the video is distributed throughout the network presents new ideas for exploiting the advantages of these physical tags.

4.2 New Interfaces Created for SMDS

The re-written multicast system at first duplicated the interface of the original Java-based SMDS using the Qt toolkit. The user is able to see which multicast channels are available and select the one he wishes to tune into and view.

However, the GUI for both tuning into multicast channels and playing the MPEG video is just an overlay that interfaces with the underlying API. We are not limited



Figure 4-1: The video tape (left) stores an entire copy of the video that the user carries from client to client. The encoded pen (right) only stores a representation of the video that is transmitted to different parts of the network as the user carries the pen from client to client.

to graphical windows-like interfaces for accessing the video. Recent experiments were done attempting to create physical interfaces to the multicast video. These interfaces take the forms of physical “tokens” that the user carries around with him to view the video.

We examined other forms of tokens that will control the video and play on screens of the user’s choice and follow the user to other displays as he carries the token with him. Like a videotape, the user can carry the video around with him and play it wherever he wants using the token, except that the token does not actually store any video; it only acts as though it does. In this case proposed here, the tokens are being used to take advantage of the distributed nature of the media being multicast. Since media is available all over the network, it is unnecessary that an actual copy of it be carried, only that a metaphor for the video be used. (Figure 4-1)

In early experiments, pens with a keyword digitally encoded in them emitted an infrared signal that caused a computer to make a database lookup and play a video from a database. The video could be picked up from any computer that was able to read the pens and access the database. This was an early demonstration of the concept I was attempting to develop.

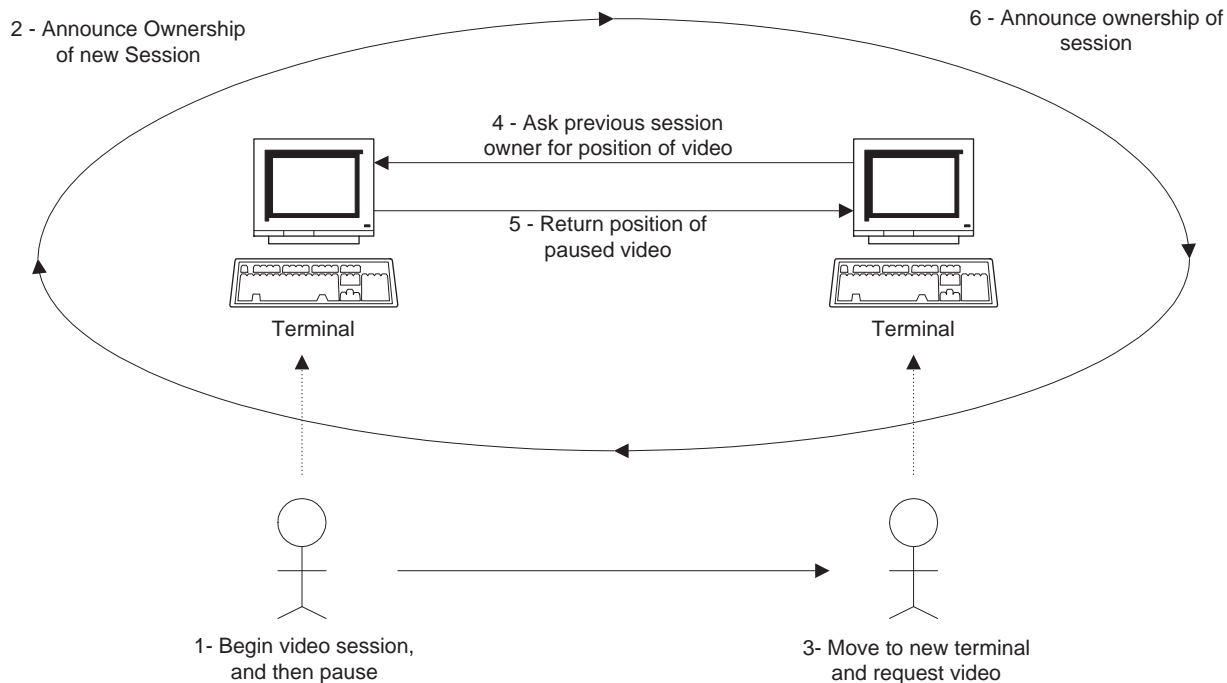


Figure 4-2: A depiction of a video following scenario and protocol

Later on, new experiments used the idea of firing “guns” at the client to provoke the broadcast and playing of a multicast channel. These “guns” were actually iRx boards with infrared emitters that were pointed at iRx boards with infrared receivers connected to the serial port of a computer. The emitter would send out two numbers—an identification number of the board as well as a number signifying a specific program. Thus, two users with their own guns loaded with the same program ID could traverse the network with their own guns and have the programs follow them individually. A basic system was created using shared files over NFS. All of the clients listen on a multicast channel which announces the creation of new sessions when users start viewing a new program. When a user moves to a new client, the new client looks at the board ID number which was sent to it by the user’s “gun” and looks up which previous client that ID has been associated with. Because all clients have been keeping track of which tag IDs are associated with which programs and the location where they are being played, the new client looks up the previous client using the tag ID, asks the previous client for the current position of the video that

was playing, updates its own session list and then announces its “ownership” of the session to the other clients.

This provides a basic framework to integrate video following with the multicast system. The original tests and demonstration used a shared file over NFS, but since the multicast system was meant to distribute data to all listeners, this was a natural initial test.

Another use of the “gun” tokens is to use them as metaphors for an entire program encapsulated from beginning to end, rather than a metaphor of a videotape which picks up from where it left off. In this scenario, the user first shoots his “gun” at a poster or other physical object representing the program he is interested in. Shooting the “gun” at the first client causes the multicast to begin. Shooting the second client causes it to pick up the multicast in mid-stream and pick up the missing information from the beginning from the first client. As the user shoots other clients, the multicasted program starts from the beginning, picking up missing information from its neighbors using the dynamic sessions described above.

The physical metaphor here is one in which the user picks up the program by targetting a movie poster and then passes the program around the network with each firing of the “gun.” In actual practice, each time a new listener is selected, the listener becomes obligated to find the specified program somewhere on the network using SMDS.

In this case, the “guns” being used were conventional laser pointers which were shot at light detectors. The light detectors were multiplexed into an iRx board connected to the serial port of a computer acting as a target server. The iRx board returns the ID of the target that was selected by the laser pointer to the target server. Since there is more than one iRx board connected to various target servers, a means of keeping track of a global state is required.

An agent-based system handles communication between target servers to manage global state— such as keeping track of which target was last hit. Each multicast receiver has agents listening for messages directing it to receive a specific multicast channel. Agents that register a hit on a listener’s target send messages to the listener’s

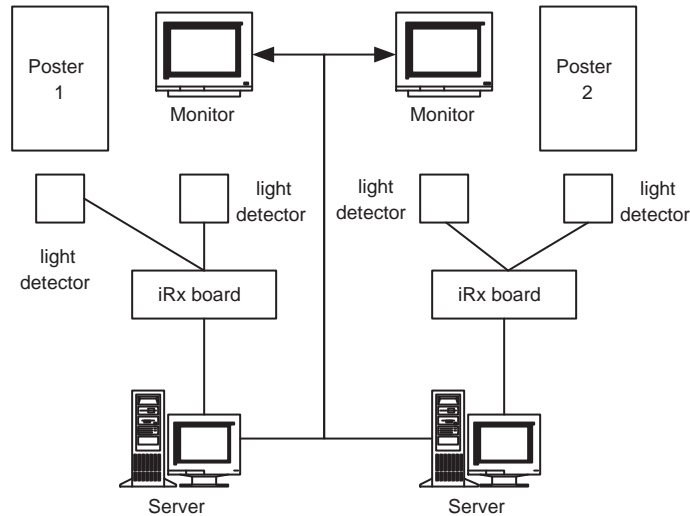


Figure 4-3: Communication between modules in the laser target control system

agents that direct it to start receiving a multicast channel to start playing the video, as depicted in Figure 4-3.

This interface is different from the standard tag-based system because in this case, the physical interface that the user controls— the laser pointer— does not contain any information at all. The system acts as though the laser pointer is being “loaded” with the desired program when it shoots the poster and that the laser pointer is “sending” the loaded program when he shoots the display. In fact, the agents store the “state” of the system and react accordingly. Within the environment, the state is updated when the user selects the poster, indicating a new video selection. The main advantage of the laser pointers is their range to better demonstrate long range selection of video and playback screens. Eventually the laser pointers will have to be replaced with tools that transmit actual information, such as a unique ID. However, in a single-user environment, the system the use of the laser pointer emulates the functions of traditional tags.

4.3 Summary

This demonstrates the integration of the dynamic sessions of SMDS and physical tags to access the media over the network. While previous work has been done with media

tags before, the tags used with SMDS refer to data that can be anywhere over the network. Finally, this work has presented the use of tags that select their data from a distance using iRx boards and laser pointers as transmitters.

These are a few examples of how to handle access and navigation in a world of infinite video. Given a wide breadth of video available to the user, new ways of organizing and accessing such media will become necessary. These tags and pointers we have examined are an example of one way to better browse through such an infinite library. As our access to video scales up from a few dozen television channels to an “extreme” scenario of an infinite selection, our old ways of thinking about accessing media will become obsolete.

Chapter 5

Extending SMDS

SMDS can be extended to provide additional functionality. As previously described, listeners can communicate with each other over pre-negotiated channels over which they are all allowed to contribute. An example of this is how Dynamic Sessions in SMDS are negotiated. There is a handshaking protocol over the IOC channel over which the listeners handle this. New protocols can be defined by constructing packets that are passed over the IOC channels. These protocols can define new functions for the multicast system.

One of the extensions to SMDS that follow this model is the creation of dynamic sessions. The first byte of the IOC packet defines a command that is universally understood by all of the receivers. For example, the Dynamic Session Request Packet has a command number of 7 (Table 3.1). As described previously, when listeners receive this packet, they respond by sending out a bid to fulfill that request. This continues in the manner described in Figure 3.4. It illustrates the point that the IOC channel can be used to allow listeners to provoke a response or an action from other listeners. This provides a framework for adding new features to the multicast system.

Another demonstration of this is a protocol that can provoke a new multicast channel or kill a current multicast channel being sent around the network. One aspect of SMDS that is still somewhat server-centric is that the listeners sit listening for new channels to be announced by a server or other listeners. Why not simply allow the listeners to provoke a new multicast stream if it wants one? We may also

Command Id	Reserved
SessionId	
Multicast Channel	

Table 5.1: Structure of an IOC packet with specifies a session ID and multicast channel

wish to allow either the originator or the requester of that new stream to cancel it. In this case, a new command can be created by adding on a new command ID which normally appears in the first byte of the IOC packet. Including a session ID will indicate to any listeners on the IOC channel which file is being specified. Specifying a multicast channel address along with the session ID could be used to cancel a specific dynamic session (which has multicast channels separate from the channel of the “main session”).

The IOC channel is the pre-negotiated channel for passing control information and can thus be used to add new functions to the listeners and server. By passing new commands between listeners, we can create new ways of allowing listeners to cooperate and share information. In the future, SMDS will use this channel and the model for IOC commands as a means of creating new ways for the listeners to collaborate with each other.

Chapter 6

Conclusions and Further Work

6.1 Conclusions

The multicast system has been optimized to provide performance close to the experience of television. Systems that require such a large amount of data to be displayed in a time-critical fashion cannot afford to be hampered by sluggish code performance. Technologies like Java are inappropriate solutions to use when building such systems like video delivery. Lower-level programming languages such as C provide the performance needed for these applications.

The current implementation is limited to using the Linux operating system and its only available MPEG decoder card, the Stradis MPEG-2 Decoder. However, the software can be easily ported to other platforms, and it is not dependent on any particular decoder card, or decoding method. As more decoding hardware becomes available, or as software-decoding performance improves, these options can be integrated into the system. Furthermore, the hardware fits the model of commercial options, such as Nokia's digital cable box, which runs FreeBSD and uses an MPEG-2 decoder card to output video. The multicast system described here only appears to be hardware and OS-dependent because other options were not available at that time.

Furthermore, this work demonstrates the use of new interfaces to interact with video. This is only possible with intelligent clients that can cache video locally and share it with others. This marks a major departure from mainstream digital video

providers which use set-top boxes without any local caching or ability to communicate with other set-top boxes. Digital video that is re-distributable by anyone turns a broadcasting system into a video-on-demand system. Distributed media provides a new way for users to experiment with interfaces by separating access to media from its physical location. Distributed media makes it ubiquitous and allows us to think about interfaces that are detached from the content itself.

6.2 Further Work

Many possibilities are presented with the video “tags”. The tags can be used to create entirely new interfaces to video editing. The tags are currently used to represent video programs. The tags can also be used to represent portions of video so that video clips can be moved around the network to implement a video editing system.

Digital video is distributed to consumers by telecommunications companies over Asynchronous Digital Subscriber Lines (ADSL). High bandwidth lines go downstream into the consumer’s home, but there is a much lower bandwidth upstream from the consumer’s home. Video sharing between clients thus becomes difficult, since clients do not have the same bandwidth as servers to deliver video. However, if the video is encoded as a multi-layered set of streams, then several clients with can send low-bandwidth streams to another listener and combined into high-bandwidth video. Since many commercial implementations of digital video delivery depend on these asynchronous networks, using multi-layered video to implement real-time video-sharing should be explored.

Also, commercial systems that deliver video over IP networks are quickly becoming a reality. These implementations have access to many more resources than any Media Lab projects can hope to take advantage of. Therefore, the future direction of Media Lab video should be to use these commercial video delivery systems and build new innovative technologies around them. Users can receive their television or other video programs via these commercial systems, and we can develop new video-sharing systems that allow the users to cache and distribute video they have saved to other

users. Implementing Napster-like protocols can create video-sharing systems, and the video delivery systems can be integrated with the video editing or video following systems. However, the *initial* path for delivering video should come over already-available systems, since they are becoming so widespread, rather than building these systems from the ground up. Then we can begin to think about more innovative interfaces and applications for digital video.

Finally, since SMDS provides a framework for reliable file reconstruction, work needs to be done in using it as a filesharing system like Freenet or Gnutella, as well as examining the applications for SMDS and multicasting in distributed file sharing. Since pure peer-to-peer file sharing applications such as Gnutella are beginning to hit a scalability limit, future work will explore if multicast can relieve these scalability issues. By experimenting with SMDS as a filesharing system, scaling problems of current peer-to-peer filesharing systems can be addressed as these systems become more popular.

SMDS provides a framework for examining new applications for multicasting. Combining multicasting and peer-to-peer concepts not only aids in scalability, but also provides new possibilities for examining interfaces to media as well as community information-sharing.

Bibliography

- [Almeroth95] Almeroth, Kevin C., Ammar, Mostafa H.; “On the Performance of a Multicast Delivery Video-On-Demand Service with Discontinuous VCR Actions”, 1995 IEEE International Conference on Communications, pp. 1631-1635
- [Almeroth99] Almeroth, Kevin C., Ammar, Mostafa H.; “An Alternative Paradigm for Scalable On-Demand Applications: Evaluating and Deploying the Interactive Multimedia Jukebox”, IEEE Transactions on Knowledge and Data Engineering, July/August 1999, pp. 658-672
- [Arai95] Arai, T., Machii, K., Kuzunuki, S., Shojima, H., “InteractiveDESK: A Computer-augmented Desk Which Responds to Operations on Real Objects”; CHI '95, pp. 141-142
- [Birman99] Birman, Kenneth P.; “A Review of Experiences with Reliable Multicast”, Software – Practice and Experience, 29(9), pp. 741-774 (1999)
- [Cai99] Cai, Y., Hua, K. A.; “An Efficient Bandwidth-Sharing Technique for True Video on Demand Systems”, Proceedings of the seventh ACM international conference on Multimedia , 1999, pp. 211 - 214
- [Chan98] Chan, Shueng-Han G., Tobagi, F., Ko, T., “Providing On-Demand Video Services Using Request Batching”, IEEE International Conference on Communications, Volume: 3 , 1998, pp. 1716-1722

- [Chen99] Chen, Dean; "IP Multicast Video with Smart Network Caches", Master's Thesis, MIT Department of Electrical Engineering and Computer Science, May 1999.
- [Clarke99] Clarke, I., "A distributed decentralised information storage and retrieval system," technical report (unpublished), Division of Informatics, University of Edinburgh (1999). Available at <http://freenet.sourceforge.net/>.
- [Floyd95] Floyd, S., Jacobson, V., Liu, C., McCanne, S., Zhang, L., "A reliable Multicast Framework for Lightweight Sessions and Application Level Framing, extended report," LBNL Tech. Rep., URL <ftp://ftp.ee.lbl.gov/papers/wb.tech.ps.Z>, Sept. 1995
- [Floyd97] Floyd, S., Jacobson, V., Liu, C., McCanne, S., Zhang, L., "A Reliable Multicast Framework for Lightweight Sessions and Application Level Framing," IEEE/ACM Transactions on Networking, 5(6), Dec. 1997
- [Gemmell00] Gemmell, J., Schooler, E., Gray, J., "Fcast Multicast File Distribution", IEEE Network, Vol.14, No.1, (Jan/Feb 2000), pp. 58-68
- [Gronvall99] Gronvall, B., Westerlund, A., Pink, S., "The Design of a Multicast-based Distributed File System," Third Symposium on Operating Systems Design and Implementation, 1999, pp. 251-264
- [Hua98] Hua, K. A., Ying, C., Sheu, S.; "*Patching*: a multicast technique for trust video-on-demand services", Proceedings of the sixth ACM international conference on Multimedia, 1998, pp. 191-200
- [Ishii99] Ishii, Hiroshi, et. al.; "musicBottles", Proceedings of the 1999 SIGGRAPH conference on Conference abstracts and applications, page 174
- [Kermode98] Kermode, Roger; "Smart Network Caches: Localized Content and Application Negotiated Recovery Mechanisms for Multicast Media Distribution", Doctoral Dissertation, MIT Media Lab, June 1998.

- [Levine96] Levine, B., Garcia-Luna-Aceves, J.J., “A Comparison of Known Classes of Reliable Multicast Protocols”, Proc. IEEE International Conference on Network Protocols, October 1996
- [Lippman2001] Lippman, A., personal communication to author
- [Macker96] Macker, J., Klinker, J. E., “Reliable Multicast Delivery for Military Networking”; IEEE Milcom 1996, pp. 399-403
- [Nickell99] Nickell, J., “Victoria’s Secret: We’re Boring”, Wired News, Feb. 4, 1999, <http://www.wired.com/news/business/0,1367,17725,00.html>
- [Ohta98] Ohta, Masataka; “Static Internet Multicast”, IEE Colloquium on Shaping Tomorrow’s Networks: Trends in Network Architectures for Delivering Present and Future Service, pp. 2/1-2/7
- [Papadopoulos98] Papadopoulos, C., Parulkar, G., Varghese, G., “An Error Control Scheme for Large-Scale Multicast Applications”, Proc. of IEEE Infocom 1998, pp. 1188-1196.
- [Ramakrishnan87] Ramakrishnan, S., Jain, B. N., “A Negative Acknowledgement with Periodic Polling Protocol for Multicast over LANs”, Proc. IEEE INFOCOM ’87, pp. 502–511, March 1987
- [Smith95] Crampton Smith, G; “The Hand That Rocks the Cradle”, I.D., May/June 1995, pp.60-65.
- [Sheu97] Sheu, S., Hua, K., Tavanapong, W.; “Chaining: A generalized batching technique for video-on-demand”, Proceedings of the International Conference on Multimedia Computing and System, June 1997, pp. 110-117
- [Ullmer98] Ullmer, B., Ishii, H., Glas, D., “mediaBlocks: physical containers, transports, and controls for online media”; Proceedings of the 25th annual conference on Computer Graphics, 1998, pp. 379-386

- [Vicisano97] Vicisano, L., Crowcroft, J., “One to Many Reliable Bulk-Data Transfer in the MBone”, Proceedings of the Third International Workshop on High Performance Protocol Architectures, 1997
- [Want99] Want, R., Fishkin, K., Gujar, A., Harrison, B., “Bridging Physical and Virtual Worlds with Electronic Tags”; CHI '99, pp. 370-377
- [Yavatkar95] Yavatkar, R., Griffioen J., and Sudan M., “A reliable dissemination protocol for interactive collaborative applications,” Proceedings of ACM Multimedia, 1995, pp. 333-44