An Architecture for Networked Multimedia

by

Jonathan C. Soo

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degrees of Bachelor of Science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

May 1995

Author _____
Department of Electrical Engineering and Computer Science
May 25, 1995

Certified by _____
Dr. Lee W. McKnight
Thesis Supervisor

Accepted by _____
F.R. Morgenthaler
Chairman, Department Committee on Graduate Theses

An Architecture for Networked Multimedia

by

Jonathan C. Soo

Submitted to the

Department of Electrical Engineering and Computer Science

May 30, 1995

In Partial Fulfillment of the Requirements for the Degrees of Bachelor of Science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science

Abstract

An open architecture for the distribution of networked multimedia was designed, with the goal being to to allow interoperability between diverse networked multimedia distribution systems built for a range of applications. The approach taken was to use existing standards and technologies as much as possible, especially the World Wide Web and the Internet. A key component, the NMIS NetPlay media player, was developed to facilitate using streaming multimedia on personal computers running popular World Wide Web browsing software. This thesis describes the architecture, compares it to other similar systems under development, describes several applications using this system, and proposes future research in this area.

Thesis Supervisor: Dr. Lee W. McKnight

Title: Lecturer, Technology and Policy Program

Acknowledgements

I would like to thank a number of people that have made it possible for this thesis to exist. First, Professor David Tennenhouse got me started in this area and created a research environment from which many of these ideas originally came. All of the past and present members of the Telemedia, Networks, and Systems group provided support and room to grow. In particular, Dr. Christopher Lindblad and David Weatherall were the source of many ideas that are the foundation of this thesis.

On the other side of campus, I must thank the whole NMIS team, but especially Professor Shaoul Ezekiel, who manages to keep NMIS alive; Paul Bosco, for envisioning and creating the project; Kip Compton, without whom this project (and much of the content of my thesis) would also not exist; and Tom Lee, for keeping it all together, and for his much needed moral support.

Finally, Lee McKnight deserves much of the credit for guiding me through this thesis, and nagging just enough to make sure that it was done.

The NMIS Project was funded by the Advanced Research Projects Agency (ARPA), the National Science Foundation (NSF), International Business Machines (IBM), and Turner Broadcasting. The NSF contract number is NCR-9307548.

Table Of Contents

Introduction 4

Today's Environment 4
The Internet - Design Goals 5
HTTP - Hypertext Transfer Protocol 8
HTML - Hypertext Markup Language 9

Background 14

Entertainment Video On Demand 14
Specialized Internet Client - Server 16
General Distributed File Systems` 18

Architecture Description 21

Implementation 24

Primary Media Server 24
Caching Proxy Servers 24
LAN Client 26
NetPlay Media Player 27
The MediaObj Media Pipeline 29
MediaSources 31
MediaPipes and Filters 32
MediaSinks 33
Caching and Dynamic Pipeline Manipulation 35

Applications 36

CNN Internet Newsroom 36
Mechanical Engineering 2.70 Instruction Project 38
CMU Informedia Interchange 37

Conclusions 39

Future Development 40
Conclusion 45

References and Bibliography 46

Introduction

Networked Multimedia is a technology that has been evolving rapidly, in both industry and in academia. As research results in real commercial products, it becomes more and more important to take the steps necessary to insure that these products have the features necessary to integrate them into a NII - level distribution architecture, rather than create isolated islands of connectivity.

This paper presents an architecture for Networked Multimedia distribution that attempts to satisfy these goals using tools and infrastructure available today, where possible. The architecture is compared to several other systems of a similar nature that are currently under development. A preliminary implementation of this architecture is also described, as well as a number of applications using this system. Finally, the architecture and implementation are evaluated, and possible directions for future research and development are described.

Today's Environment

This situation is similar in many ways that of the computer networking community in the late 1970's and early 1980's. At that time, much of the technology needed to support the worldwide internetwork known as the Internet had already been developed. LAN and WAN products were being sold by a number of manufacturers such as IBM and Digital. Many leading customers understood the business need for integrated enterprise-wide technologies, and were rapid adopters and drivers of those technologies.

However, those networks were provided by very vertically integrated firms. IBM provided all of the elements of its network, from the hardware to the operating systems, network protocols, and applications. DEC similarly had a broad range of products. Although there might be provisions for private networks to interact with other private networks, this usually happened only between systems from the same vendor.

There occasionally was a need to communicate with a competitor's system, and those cases were usually handled one at a time, with custom designed hardware and software used only for a single application.

The best aspect of this system was that it worked. If a company bought into the vision of a single vendor and used their equipment exclusively, there was a good chance that the combinations of equipment and software had been designed and tested together. If they didn't work, the vendor could be held responsible for finding the problem and fixing it.

There was little need for external specifications, and in fact, there were significant commercial reasons not to divulge them. Releasing these external specifications would allow competitors to design equipment that could compete with a vendors equipment, reducing profits. In addition, having a public specification could reduce the ability of a vendor to make changes to their products.

However, economically, this was very bad for customers, especially smaller ones. Once a particular architecture was selected and implemented, the customer would be dependent on the vendor for the lifetime of the architecture. Because of the lack of competition, the customer would pay more for the system, and would be limited to the capabilities designed in by the vendor. If a competitor designed a better system, it would be difficult to implement it incrementally, because it would not be able to interoperate with the original system.

The Internet - Design Goals

The Internet was created in part because the US Department of Defense, one of the largest buyers of computers and network equipment in the world, was facing this problem and had the funding and influence to design a system to solve it. The result was the construction of a suite of internetworking protocols which would allow systems using several different networks to function as a single entity. Systems designed using this technology proved to be robust and useful, and the Internet has grown to be one of the largest data networks in the world, used for a wide variety of purposes not envisioned by its creators.

There were several important design goals of the Internet Protocols which made it possible to be so successful [Postel81]:

- Interoperability - They had to be usable on many of the existing, heterogeneous networks in use at the time. This feature was important because it made it possible to "piggyback" on top of existing systems instead of requiring that entire new infrastructures be built. This also meant that the Internet's acceptance did not depend on any single network architecture becoming dominant.

- Ease of implementation - It was relatively easy to implement a working Internet Protocol stack on existing platforms. This meant that IP quickly became available on a wide range of systems, increasing the effective "market size".

- Robustness - The Internet was designed to be robust in the face of failure of its underlying components. This meant that it was reasonable to depend on interconnections with systems that one did not have administrative control over, allowing the it to span a potentially much larger user base.

- Simplicity of programming - The transport protocols of the Internet proved to be easy to use, so many applications (such as electronic mail and file transfer) were developed. In addition, the protocols were designed to be flexible enough so that a wide range of applications could be supported. This allowed the Internet to be used for services that the original designers never intended or even imagined. This also allowed a larger, more diverse community of developers to exist.

- Performance - TCP/IP and UDP were sufficiently efficient so that there was no need to implement and deploy competing protocols, because they would probably provide less of a performance improvement than devoting an equal effort to additional tuning of TCP/IP [Clark88]. The result of years of experience and experimentation has led to many improvements, and has also resulted in good performance in many situations (such as high congestion) that might be difficult to achieve otherwise. The universality of TCP/IP has also led to important advances in several areas, such as operating system level mechanisms for protocol processing.

By the early 1990s, the Internet was well established as the global internetwork. However, the Internet was not entirely successful from a commercial perspective. A number of problems still existed:

- The applications used on the Internet were difficult to use. While this was not a problem for the original designers and the existing user community, this was a serious barrier for the much larger community users outside of the computer science area.

- There were no effective, widely available navigation tools. Finding information on the Internet was a challenge even for the most experienced users. While there were many repositories of useful data on hosts around the Internet, the only way to find out about them was through other channels such as electronic mail, network news, or personal communication. There were a few prototype services available such as Archie or Gopher, but they each had flaws; Archie was only a mechanism for locating, not retrieving files, and Gopher forced a rigid information structure [ANKL].

- There were few presentation layer standards. Text was only available in plain ASCII text, which was not attractive and could not support advanced formatting or graphical images. Postscript produced high quality output, but was not suitable for on-line viewing because of performance and licensing requirements. For raster images, GIF (Compuserve Graphics Interchange Format) was widely used, but was not well suited for some types of images, and a number of other formats were also popular. In addition to all of this, there was no standardized way to determine the format of any particular file.

- "Publishing" was technically demanding. To set up a publicly accessible archive required the skills of an UNIX systems administrator. For instance, FTP (File Transfer Protocol) is a very popular method of transferring files, and still accounts for a large portion of the information transferred over the Internet. However, setting up an FTP server requires administrative privileges, and mistakes can result in dangerous security holes, which are fairly common.

- There was no mechanism for accountability or commercial transactions. While some applications were secure, they were not widespread. Publishers had no way of ensuring that they could get any kind of economic return from their efforts.

Starting in the early 1990s, many of these problems were addressed by a set of standards and applications collectively known as the World Wide Web. Two standards are a critical part of the World Wide Web: HTTP [Berners-Lee94a] and HTML [Berners-Lee94b].

HTTP - Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP) is the protocol used to transfer the majority of information transferred over the World Wide Web. It runs using Transmission Control Protocol (TCP) as a transport protocol, although it could theoretcially work over any reliable stream transport. HTTP shares several of the characteristics of TCP/IP that have made it very successful:

- **Interoperability** - HTTP operates on an extemely wide variety of platforms, ranging from personal digital assistants to supercomputers. Because of this, HTTP can leverage improvements in underlying protocols, network architectures, and host systems.

- **Simplicity of implementation** - HTTP can be implemented in a few pages of C, and even more easily in higher level languages because it is a text based protocol. Because of this, clients and servers have rapidly become available on nearly every platform that can be connected to the Internet. In fact, on many platforms, several independent HTTP implementations are available, resulting in a great deal of competition and improvement.

- **Robustness** - Although this is no longer as important a factor because of the increasing reliability of computing systems, HTTP is a stateless protocol, and is resilient to network failure. This potentially could be more important as more clients are "occasionally connected", and also makes it easier to scale servers up to handle large number of clients.

- **Performance** -The transaction overhead of HTTP is reasonably small, especially when transferring relatively large files over slower connections. Because it uses TCP, the actual data transfer is very efficient, especially over links congested with TCP traffic.

- **Security** - Although there is not yet a single standard for performing secure transactions using HTTP, HTTP has mechanisms to ensure that the competing standards can coexist and number of mechanisms have been proposed.

HTML - Hypertext Markup Language

Hypertext Markup Language (HTML) is a hypertext structuring language used for formatting most of the documents currently on the World Wide Web. It is based on Structured General Markup Language (SGML), a standard for describing text structure. It was instrumental in solving, at least partially, two of the problems.

- **Uniform Presentation** - HTML is a easy to use language for structuring and presenting text. It can be used for a wide range of documents, and allows those documents to be viewed on a wide range of platforms, including text-only displays, high resolution graphics monitors, and printed paper. The directives are simple enough for non-experts to use, and can be edited even without complicated editors. This has enabled the growth of a large population of HTML-literate authors.

- **Simple Navigation** - HTML includes directives for networked host hypertext links. These usually are implemented as highlighted text or graphics which, when clicked, cause the client's browser to retrieve the referenced document from the proper host. This makes the World Wide Web self-navigating, because a user can jump from place to place without the need for external tools. As the WWW became more popular, ordinary users began creating their own lists of useful references using HTML, and in fact, many sophisticated information-locating databases use HTML brewers as a user interface.

Because of these characteristics of HTTP and HTML, and because of several excellent, freely available web servers and clients, the World Wide Web has been popular for electronic publishing, by professionals and non-professionals. However, at this time, the World Wide Web is mainly used for distributing text and images. There are a number of reasons for this; many of the clients on the World Wide Web cannot view "multimedia" documents, simply because they do not have the proper equipment and software. Many also do not have sufficient bandwidth to transfer these multimedia documents easily. But, there are also some limitations in the way that the World Wide Web is actually implemented today that prevents widespread use of networked multimedia, even for clients with sufficient functionality and bandwidth [Soo94].

There are several architectures that are trying to address this problem and others. There are a wide range of target applications and constraints, and the participants come from a wide range of backgrounds. The general goal of all of these architectures is to provide Network Multimedia. There are a few broad target classes of usage today:

- **Entertainment Video On Demand** - This is the driving force of the Cable Television industry, a natural evolution of the usage model currently in place today. In this model, consumers tend to watch a relatively small group of "hot" titles, and interact mainly when deciding when and what to watch, with some ability to pause or scan back and forth in the title. There may be thousands of simultaneous users watching high quality video streams, and the cost of the terminal, the "set-top box" is extremely important [Chang94].

- **Digital Video Libraries** - This usage model is in some ways opposite from the VOD model. Users of these libraries tend to access a very wide range of material, with fewer simultaneous accesses. Although the search engine needs to be much more sophisticated than in the VOD model, after the title is found, the amount of interaction is fairly low. Because the users of this system may have some economic gain from the information that they find, their equipment can expected to be not as cost sensitive as the entertainment consumer.

- **Specialized Interactive Services** - This class of service covers the range of services where interaction takes place within the title, such as networked multi-player computer games to medical training programs. While they do not need to have tremendous amounts of data flowing from clients back to servers, or to satisfy tight real-time constraints (such as are necessary for videoconferencing applications), there must be a way of ensuring smooth, uninterrupted service and reasonable response to interaction.

Development effort can mostly be categorized as being in one of these classes, and the architectures generally do not interoperate. In addition, there is a large amount of fragmentation within each of the systems, resulting in architecturally similar but incompatible systems that require a large amount of redundant development. Table 1 illustrates some important characteristics of these three usage classes.

	Video Quality	Aggregate Usage	Cost per User	QOS Requirements
Entertainment Video On Demand	**High**	**Very High**	**Low**	**High**
Digital Video Libraries	**Med**	**Med**	**Med**	**Low**
Specialized Interactive Services	**Low**	**Low**	**High**	**Med**

Table 1. Usage Classes

For end-users, these incompatibilities may or may not have a direct effect. Because many of these services, especially Video On Demand systems, are clustered geographically, a consumer may not have to face choosing among multitudes of systems; there may be only a handful of information providers that he or she can physically connect to. In addition, the low cost of the end-user terminal makes this decision easier; in fact, in many situations, the information provider may actually own the box and lease it to the consumer, essentially making the actual system used opaque to the user, at least economically.

Users of more sophisticated systems potentially face more problems, because the cost of the access device may be much higher. In that case, the user faces a choice, and will probably determine his or her actions depending on the utility of the information that is being accessed, and the price of the equipment.

Content providers, on the other hand, will face serious economic problems if there are a large number of incompatible systems being used. Many content providers are not geographically limited in distribution; they need to reach as large of a market as possible for their own economic reasons. If they operate at a national (or international) scale, this means that they must consider all of the systems in place in the entire US, potentially dozens. To reproduce content for even a few of the largest of these systems could increase the lead time and the effort required, seriously raising the cost of production.

For that reason, it is desirable to create an infrastructure for the wide-area distribution of Networked Multimedia [Bosco93]. From experience with the Internet and the World Wide Web, we can determine a number of characteristics that this infrastructure should have:

- **Interoperability** - This system should be capable of interconnecting several types of Networked Multimedia distribution architectures. One approach to a Networked Multimedia infrastructure is to construct a single, scaleable architecture. While this would simplify the design of this architecture, it is not feasible because it would waste too much of the tremendous effort already invested in Networked Multimedia distribution architectures. In addition, it would be not be politically feasible to make all Networked Multimedia users to adopt a single specification. Allowing a heterogeneous infrastructure has many important benefits; it allows many different architectures to compete, forcing implementers to innovate, it also allows the adoption of unforeseen new technologies.

- **Simplicity** - Competitive system implementors will evaluate the cost benefit of any new functionality required for interoperability. The easier it is to add this functionality, the more likely that it will be adopted. Without significant external pressures such as regulation, this could play an important part in determining whether an internetwork is widely adopted. The complexity of the

new functionality needed can also affect the quality of the implementations and determine how well the final system actually performs.

- **Robustness** - System implementors are unlikely to depend on any system that is not robust, especially as technology moves from academic research to commercial distribution. Any system should depend as little as possible on the correct operation of components outside of administrative control. In many industries, failures could result in the loss of revenue from thousands to millions of customers. As the size of a system increases, this becomes more and more of a factor.

- **Scalability and Flexibility** - Because of the rapid pace of change in the involved industries, it is important that a distribution system be scaleable in several directions . Some of the more important characteristics that must scale are image quality and usage levels. However, some other less obvious qualities should also be scaleable. For instance, some users may want a larger degree of fault tolerance than others, or may have different security and billing requirements. It should be possible to support a wide range of applications, especially the general categories described above.

- **Support for commercial usage** - Because this infrastructure will be used for commercial purposes, it is important that it provide mechanisms to support this. For instance, it should be possible to perform distribution securely, and it should be possible to support billing transactions through this system, and it may be important to support some form of intellectual property protection. It should also be possible to recover costs fairly from users, perhaps with support for bandwidth reservation or quality of service guarantees.

- **Content-provider friendliness** - The system should insulate content providers as much as possible from the characteristics of the distribution system, but should also allow them flexibility. This is a difficult balance; giving content-providers details about end systems necessarily exposes

their differences. It may be possible to use some type of negotiation (such as in HTTP) to simplify these decisions or delay them.

- **Open Access** - As on the World Wide Web, it should be technically possible for content-providers of all sizes to easily publish information. This also means that where possible, the technical and regulatory requirements should not discriminate against smaller content-providers. End-users should be able to access easily as broad a range of content as possible.

It is important that an interoperable infrastructure be developed at this time, while distribution architectures are still under development. After these systems are fully developed and implemented, it will be much more difficult to modify them to allow them to interoperate.

Background

There are currently several popular approaches to providing networked multimedia services, each specialized for a class of applications described above. These systems are distinguished by their balance between client and server expense, their dependence on underlying network characteristics, and the amount of specialization in the protocols and hardware used.

Entertainment Video On Demand

Entertainment Video On Demand systems are distinguished by several characteristics. The applications themselves are typically fairly simple, consisting of simple playback of audio and video, but the user communities can be extremely large, with thousands of concurrent users. In addition, these user communities are extremely cost sensitive; there may be tens of thousands of clients in a system, so there is tremendous pressure to reduce the cost of client hardware. On the other hand, cable systems have the advantage that they operate over a fairly limited geographic area, and to some extent can specify and install the underlying network infrastructure that they need.

Because of the cost constraints, it is preferable in these systems to push as much complexity from the end user to the centralized server. Set-top boxes have the minimum hardware possible; they need a transceiver to decode the incoming data, a module to decompress the video stream and convert it to an analog video signal for output to a television monitor, and a control unit to handle user interface, billing, and other local features. Although there will be some memory needed to decompress the video, and to handle the user interface, this amount will be small compared to the amount needed to buffer a significant amount of video information. As a result, the set-top box depends on a very structured incoming data stream that satisfies time constraints. This requires a great deal more complexity further upstream in the system [Federighi94].

The place where this is most apparent is in the centralized video server. The server needs to feed hundreds to thousands of video streams with very strict timing requirements from physical storage. Most of the storage in these servers is in the form of rotating magnetic media, possibly with a larger archive of streaming tape. Because the rotating media and streaming tape are not true random-access media, complex mechanisms are needed to schedule these media to meet the timing requirements, and possibly to replicate resources to meet overall bandwidth requirements.

To reduce the demand on the servers, one technique used is "batching" video segments together [Dan94]. This takes advantage of the fact that video is isochronous, that the underlying medium can broadcast or multicast without additional overhead, and that the user is mostly passive. By delaying the start of playback, many requests can be satisfied simultaneously. It is estimated that server load can be reduced by 60% using this technique, with typical television viewing patterns.

It is interesting to note some of the reasons why modifying end-user behavior is feasible here. The scale of the system is definitely a factor; because of the large number of users, it is possible to treat their satisfaction statistically. Also, the customer base is relatively insensitive to initial delay because it is used to existing, passive, broadcast services. Finally, there are enough "hot" titles to make batching worthwhile. Other systems may not have all of these advantages.

At the same time, the end-users are much more sensitive about other aspects of the service; a delay at startup might be acceptable, but glitches once a program starts might not be tolerated. Fortunately for the system designers, their task is made easier by the size of their systems and the fact that they are closed. Because of those two reasons, they can allocate their resources ahead of time, and have a very reasonable expectation of usage because of the large number of users.

Specialized Internet Client - Server

A great deal of effort has been concentrated on creating video distribution and manipulation systems using more general computer networks and workstations. Here, the goal is to leverage continuing improvements in networking and computing technology. The changes that can be made are much more limited; usually it

is not possible to design a completely new network or workstation for this purpose, although it is definitely possible to influence the direction of development. In the last few years, workstation architectures have gradually been including features designed to make multimedia computation more efficient, and network architectures such as Asynchronous Transfer Mode (ATM) have been created with the intention of supporting this type of usage. The development has mainly proceeded in the area of software and protocol development.

Typically, the budget allocated per user is much higher than in a cable VOD system, or in a system running a generic network protocol. Partly because of this, the developers of these systems have focused on providing a much higher level of functionality than in the other types of systems. For instance, several of these systems have a very high level of user interaction, providing nearly real-time response. A number of these systems also provide access to a much broader base of material than in the cable VOD systems. In general, there is a focus on being able to scale functionality (overall bandwidth, data set) rather than on increasing the number of users or improving the cost performance.

The Berkeley Continuous Media Player [Rowe92] is an example of such a system. The distinguishing characteristics of the CMP are the use of software MPEG video compression and decompression, and a priority based bandwidth allocation system. The CMP is a software-only platform; it does not specify any particular network hardware. The CMP itself uses a protocol built on top of IP as a transport protocol; it is a video frame based system that has been designed specifically for video unicasts, with provisions for running on overloaded networks. When running on highly loaded networks, it has the ability to drop video frames based on a priority system so that some classes of video get better treatment than others. This frame dropping happens "cleanly" and the CMP is able to maintain synchronization between multiple parallel streams of audio and video.

One of the reasons for this "frame-dropping" functionality is the fact that the CMP is built on top of an internetworking protocol, rather than any particular type of network. Because of this, the CMP can not rely on any quality of service guarantees. At the same time, the CMP has the flexibility on running on many

workstation platforms, and on top of many networks, including WAN links. For many types of applications, especially ones distributed around the Internet, this is a desirable attribute.

Many of the CMP's assumptions and design choices are debatable. For instance, a great deal of development has gone into the priority-based transport protocol. The assumption here seems to be that these applications will be running at a good fraction of the available network bandwidth. Considering the data rates supported by the software decoding (a few Mb/sec on high end workstations), it seems that the bandwidth needed would not tax even a single 10 Mb Ethernet segment, and the simple assumption that many of these systems would share a single low bandwidth network segment ignores the trend towards switching networks and away from bus-based networks. At lower utilization, a much simpler transport protocol (such as TCP) together with some additional buffering might result in the same performance with much lower complexity.

Secondly, it is not clear whether the reliance on software video decoding is the best choice in the future. Although high end processors and memory subsystems will be able to support real-time video decompression in the near future, relatively inexpensive hardware solutions exist and provide much better cost performance, and will probably always cost much less than a software implementation because bandwidth is much less expensive on a single chip than on a circuit board. In addition, the production volume of these hardware solutions may be much higher than that of high-end microprocessors because they will also be used in mainstream consumer products such as digital televison receivers. The main remaining motivation for software decoding would exist if the decoding standard was still in a state of change; however, in the last year, compression techniques have standardized, removing this rationale.

General Distributed File Systems

Another approach to networked video distribution is to use general purpose network file systems to distribute video. This is usually fairly easy to implement, because new protocols do not need to be designed and tested. In fact, it is usually straightforward to take an application that has been designed to be run off of local storage on a workstation, and run it on a network file system. It also has potential performance

limitations because video applications may have different usage patterns than those that most network file systems have been designed for.

For instance, most file systems have caches that can take advantage of spatial locality within a file. For many applications that use random access within files, such as databases, this can provide a significant performance advantage. However, video is typically accessed in a serial fashion, often straight from beginning to end. While individual video files may be re-accessed frequently, the size of these files may be much larger than any local cache, and in many cases may be larger than the entire available local storage. In these cases, having local caching may actually decrease performance if it imposes significant overhead.

In contrast, a system specialized for video playback may be able to take advantage of usage characteristics. For instance, assuming serial playback would make it much easier to implement prefetching, which could be very helpful for reducing latency. Another aspect of video playback is that, at this time, video files are not often modified by end users. A system that could assume read-only or read-mostly usage could then ignore many of the more difficult problems faced by regular network file systems, which have to do with maintaining some sort of coherency and preserving the write semantics of regular file systems. This at the least would make the system much easier to implement, and probably would make it much more efficient. It would probably also make it easier to take advantage of mechanisms that have not yet found much usage in distributed file systems, such as broadcast and multicast capabilities of the underlying network.

One candidate for use as a wide area networked multimedia distribution system is the Andrew File System (AFS) [Satya90]. There are several desirable characteristics of AFS, the most important one being a universal naming system. This is important for enabling access to information that may be distributed across many systems. However, there are many disadvantages to AFS that are typical of any file systems, most of which are mentioned above. Additionally, partially because AFS is difficult to implement, and partially because most network file systems are implemented at the kernel level, there are many systems for which it is not available.

There are several important reasons why video applications running over generic networked file protocols will remain important. The first is that it preserves compatibility with existing applications that run off of local storage, many of which are in any case not designed to be used by multiple simultaneous users. Other video applications may require the ability to jump within video files [Wactlar94]. Another advantage is that it leverages hardware and software development in the easiest possible way. If it is necessary to upgrade the network or workstation to use video applications over the network, it may improve the performance of other applications also. Finally, for many sites, it may administratively be very difficult to install and use specialized protocols, especially on multiple platforms. In that case, improving the performance of a generic networked file protocol may be the only possible solution for distributed multimedia.

It may also be feasible to use these file systems as a part of a multiple level media distribution system, as described below. In this scenario, where the file system operates on a high-speed LAN, the inefficiencies of using a general purpose file system may be outweighed by the ability to use regular software, especially for applications that use random access within video files.

Architecture Description

The Network Multimedia Information Services (NMIS) Media Distribution Architecture uses HTTP as the main protocol for transferring data between systems on the Internet, and HTTP servers and clients as servers, caches, and clients. HTTP is well suited for multimedia distribution because it is inherently a streaming protocol; once a connection is opened and negotiated, data is transferred from one point to another at the maximum rate allowed by TCP, with no additional overhead.

One characteristic of HTTP that this architecture takes advantage of is the fact that it is easy to build caches that operate transparently and efficiently. Because of this, many levels of caching storage can be implemented with size and location determined (possibly dynamically) by engineering needs and economic considerations, rather than by rigid constraints.

The various levels of storage can be broadly divided into three types; primary servers, proxy caches, and clients. These names primary describe their logical functions, rather than physical characteristics, so it is possible for a component to be a primary server and proxy cache at the same time, for instance.

Primary servers are the places where original material is archived and distributed. These server essentially the same purpose as regular HTTP servers do on the World Wide Web. In many cases they will be operated by the actual content providers. The most important aspect is that, like any other HTTP servers, they may be globally accessible on the Internet using HTTP. This is important in that it allows any host on the Internet to become a multimedia content provider, regardless of their geographical location.

The next storage level consists of caching proxy servers. These are intelligent parts of the network, and help to decouple the end users from the content providers. Instead of sending requests directly to primary servers, clients on a single subnetwork can forward their requests to a caching proxy server on the same subnetwork or between the subnetwork and the primary server. The caching proxy server then acts as a client to request the information from the primary server, and forward it back to the original client. It can

also store a copy of the data in its local cache. If another client on the subnetwork forwards it a request for a copy of the same information, the caching proxy server can check to see if the data is up to data, and then forward the data from local cache to the client, saving a request to the primary server.

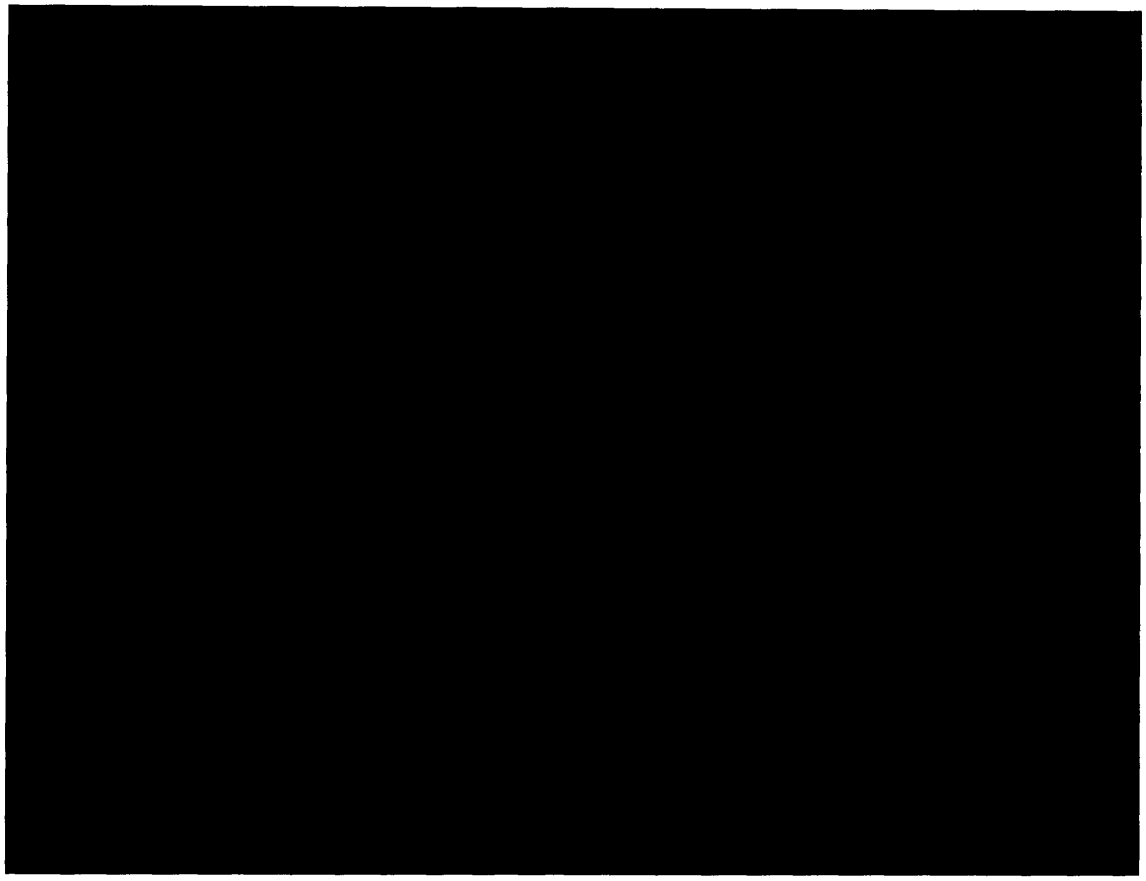

Figure 1. NMIS Media Distribution Architecture

This has many advantages; if many clients on a subnet request the same information, this could significantly reduce the load on both the primary server and the connection between the caching proxy server and the Internet. It can also improve performance If the connection between the caching proxy server and the client is faster than the external connection.

This provides a single point of administrative control for usage of external network resources; because networked multimedia can consume significant bandwidth, this is very important. An administrator can easily allocate the usage of bandwidth, and limit the multimedia services that are available. As primary

servers start charging for their networked multimedia services, this will also allow for more sophisticated billing systems, where, for instance, an organization could purchase a single "subscription" that it could then share internally.

From an engineering perspective, the decoupling of the end user from the primary server makes many issues much simpler. As described above, because the primary server does not have to deliver data individually to all clients, it may be able to serve many more clients without as much load. Also, because there is not direct end-to-end connection between the primary server and the end client, timing and latency problems are greatly reduced. For instance, the primary server can stream data to a proxy cache at a slower than real-time rate, without impacting playback between the proxy cache and the end user. More aggressively, primary servers could send material before it would be used, or use a broadcast or multicast extension to HTTP to reduce load even further.

The local distribution system has much different engineering requirements from the wide area distribution system. The most important difference is the fact that the local distribution system is more likely to be under a single administrative control. This allows engineering tradeoffs to be made at a different scale, and allows the system to be more specialized. For instance, if the local distribution system is geographically local, networking resources will probably be much less expensive than WAN connections. There will probably be a more limited number of end users, or at the least, the number of end users will be known approximately. On the other hand, since the local distribution system will have to deliver content directly to end-users, latency, jitter, and other quality of service requirements will probably be much more stringent.

Because of those reasons, it is desirable to allow a variety of systems for local distribution, depending on the application and client capabilities. The key element is the presence of a gateway between that system and the HTTP proxy cache. Fortunately, because HTTP is a relatively simple protocol to implement, it should be relatively easy to construct these gateways.

Implementation

Multiple levels of this architecture already exist or have been implemented. In particular, the higher levels of the hierarchy (the Primary servers and Proxy servers) are already parts of the existing World Wide Web architecture, and have been used almost unmodified.

Primary Media Server

At the top level of the NMIS hierarchy are Primary Media Servers. Primary servers store the "orignals" of the distributed media. These servers are implemented using standard HTTP server software, such as CERN httpd 3.0. No modifications to the server software are necessary, because HTTP itself is a streaming protocol. There are no special requirements for the server machine, but typically it will have large amounts of storage and a fast network connection because of the storage and bandwidth requirements of digital video.

An important aspect is that the HTTP server does not need to interpret the stored video data in any way. This means that the server can store and transmit video data encoded in any format, given that it has enough storage and bandwidth. Because of this, new video formats can be added to the system at any time, with modifications only to the clients. This also means that existing HTTP servers can be used to store video as just another data type, so video files can exists along with text, graphics, and other multimedia.

A number of hosts have been tested as primary media servers. The current NMIS server runs on an IBM RS/6000 with 81 GB of rotating storage ad a 10 Mb/s connection to the MIT campus network, which it is capable of saturating with video data.

Caching Proxy Servers

The next level of storage in the NMIS media architecture is the Caching Proxy Server level. This is a term for HTTP servers that exist primarily to act as an intermediary between the client and other HTTP servers.

Rather than issuing a request directly to an HTTP server, the client sends the request to the proxy server. The proxy server parses the request, and itself issues an HTTP request directly to the desired server. It then passes the data received from that server back to the client.

There are a number of historical reasons for the existence of proxy servers. Many security-conscious sites operate "firewalls" which prevent data from entering and leaving a private network except at designated, controlled points. HTTP proxy servers allow users to access external HTTP servers while maintaining a single point of control.

Another advantage of having a single point of control is that data returned from HTTP servers can be cached. This means that, data from a popular URL only needs to be retrieved once from the original server, with all following requests being satisfied by the caching proxy server. This can significantly reduce the bandwidth to the external world used by HTTP transactions.

Because, to servers, media streams look like any other HTTP transactions, these caching proxy servers can also be used to cache media streams. The only requirement is that the proxy server be capable of streaming the data to the client at a fast enough rate. There are two categories of proxy servers; store-and-forward, and cut-through. Store-and-forward servers must complete an HTTP transaction before they return data to a client. When the external link is slow or the size of the object returned by the transaction is large, this may result in a significant delay before the object finally arrives at the client. Streaming is not useful with this type of client, because there will always be a delay before the start of streaming.

Fortunately, to reduce latency for even small HTTP transactions, most caching proxy servers today are designed as cut-through servers. This means that as soon as the proxy server starts receiving data from the original HTTP server, it forwards the data directly to the client. Here, streaming is possible, and the proxy server only adds a small amount of latency while forwarding the data.

The hardware requirements for these proxy servers is usually less than that of primary media servers, depending on the number of machines served and the usage expected. A caching proxy server will typically cache only a subset of the media on the primary server, managed in using a LRU cache

replacement algorithm. Much of the cache is also shared dynamically with other, non-video documents, if the proxy server is being used to cache non-video HTTP transfers. Currently, NMIS uses RS/6000 and Intel Pentium class personal workstations with approximately 1 GB of disk each, running AIX 3.2 with CERN httpd 3.0, or Windows NT with the Purveyor proxy server. Both of these platforms are capable of saturating a 10 Mb/sec Ethernet, although when operating in cut-through mode, singly-connected to a LAN, this 10 Mb/s must be split between 5 Mb/sec from the primary server to the proxy, and 5 Mb/sec from the proxy to the client.

LAN Client

The particular local playback mechanism chosen for this implementation uses HTTP running over TCP/IP as the Presentation/Session/Transport protocol, 10 Mb/s Ethernet as the LAN architecture, and relatively low-end (ranging from Intel 486 to Pentium) personal computers running Windows for Workgroups 3.11 (with TCP/IP) as the operating environment. In addition, the computers were configured with hardware MPEG audio/video decompression cards (Optibase PCMotion Pro [Optibase94]), to allow playback of VHS quality, full-motion video. An estimated cost per client is $2000.

There were several reasons for the selection of this configuration:

- **Hardware** - The configuration of the clients used in this implementation is representative of a very broad segment of the existing computer population. Besides making it much easier to procure client systems, this means that the techniques developed should be very useful in the near term. The one "new" component used is the MPEG video decoder board; as MPEG becomes even more of an established standard for the encoding of digital video in many fields such as broadcast digital television, as well as traditional computer multimedia, this component will become less expensive and more common, much in the way that advanced graphics or sound functionality has become a standard part of computers today.

- **Software** - Although Windows 3.11 is far from a perfect environment, it does have that advantage that it is widely available, has a wide range of well-tested development environments, and has an

excellent TCP/IP implementation. In addition, some important software components such as drivers for MPEG video decoder boards are very difficult to find on other systems.

- **Protocol** - HTTP is a protocol that is widely used and easy to implement, and because it is implemented on TCP, its data transfer performance is excellent over LANs and the Internet. In addition, there are several advantages to using the same protocol for the LAN distribution as for the WAN distribution; one is that the client can play back directly from a primary server located anywhere on the Internet, given enough network bandwidth, without requiring an intermediate caching proxy server. This makes the system more attractive for sites that initially have only a few clients, and more importantly, makes caching proxy servers an important optimization, rather than required components.

NetPlay Media Player

The key application developed to support streaming video playback over HTTP was NMIS NetPlay. NetPlay is an application that opens an HTTP 1.0 connection to either a primary HTTP server or a proxy server, and then streams the received data to the client's output device, in this case the Optibase PCMotion MPEG decoder card. Although it is capable of operating as a standalone application, usually NetPlay is invoked as a "helper" application by a Web browser such as NCSA Mosaic or Netscape Navigator.

Currently, helper applications are invoked as child processes of a Web browser when objects are retrieved with content-types that cannot be handled internally by the browser. For instance, if a browser retrieved an object that was a Postscript file, and the browser did not support viewing or otherwise manipulating that type internally, it might spawn a postscript viewer program and pass that program the object to be viewed.

Unfortunately, the mechanism used to pass that object to the helper application is usually very limited. The standard technique is to receive the object entirely and copy it to a temporary file, start the helper application, and pass it the name of the temporary file on its command line. This causes several problems, especially for large, streaming multimedia files. Enough temporary storage space must be kept available to

store the largest object expected, which, in the case of digital video, might be hundreds of megabytes or even gigabytes.

Second, the helper application is not started until the entire object to be transferred, even if it can make use of parts of the object. For instance, the data representing the first few minutes of a large video file cannot be played back until the entire file is transferred, which may take hours. This makes any sort of "live" playback all but impossible.

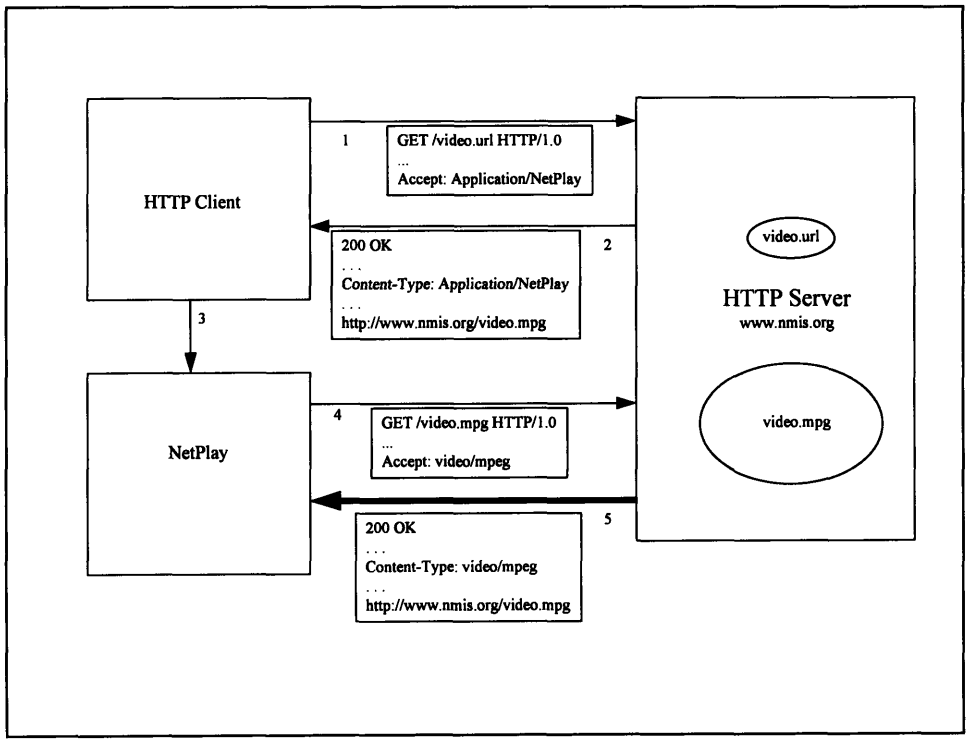

Figure 2. URL Indirection Sequence

The technique used by NetPlay to solve this problem is called URL indirection. This technique is fairly generic and has been independently developed in some other systems. In an HTML document, when a hypertext link to a video file is desired, a link is instead made to a plain text file (informally known as a URL file) which consists of one or more URLs, each on its own CR/LF delimited line. These URLs in turn point to the actual location of the video file referenced. The HTTP server is then configured to give this

type of files a MIME content-type, such as application/x-netplay, and the browser is configured to start NetPlay as a helper application for that content-type.

When the URL file is retrieved from the server, the browser copies the file to a temporary file in local storage. It then starts NetPlay with the command line argument being that temporary file. NetPlay then opens the file and parses it, and is then free to open its own HTTP connection to retrieve the video file.

The MediaObj Media Pipeline

NetPlay itself is composed of components implemented in C++ called MediaObj objects. The output of an object can be connected to the input of another to build a media pipeline through which the video stream logically flows. This structure is similar to that of the MIT Telemedia Networks and Systems group's VuSystem media processing system [Houh95], which has been shown to be a useful abstraction for building media applications.

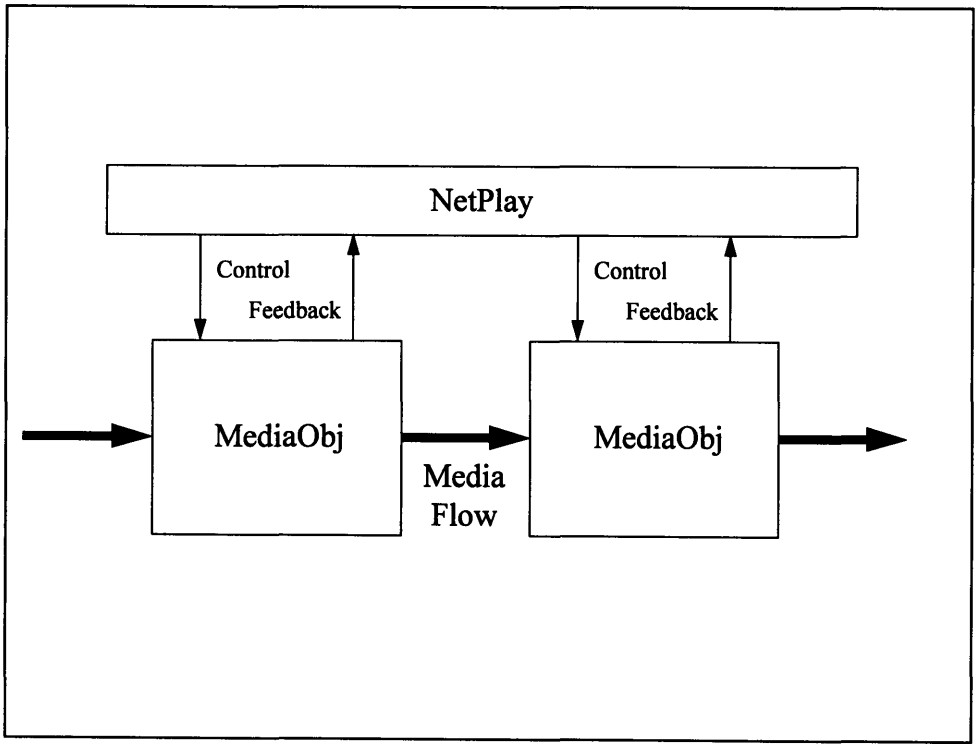

Figure 3. MediaObj Media Pipeline

Each MediaObj object supports a number of basic methods. These can be divided into two categories: in-band and out-of-band. In-band operations are those involve the transfer of media data from the object's inputs to its outputs, or from lower level services such as networks or file systems. In-band operations can be expected to take place very often, and often can involve the movement of large amounts of data. Because of this, these operations must be implemented with special attention to efficiency. For instance, rather than making multiple copies of data, pointers to memory areas are passed between objects where possible. In contrast, out-of-band operations are those that deal with controlling the MediaObj objects themselves and providing feedback to the overall application. Because they are used relatively infrequently, performance is not as much of an issue.

Some in-band operations:

- **Read** - A MediaObj object calls the READ procedure of the object upstream of itself in the pipeline when it requires more data to process. For most types of MediaObj objects, this happens as a result of a READ call on itself made by the next object downstream, although MediaSinks may make this call because of external events (such as hardware interrupts). The default behavior is to recursively call the next object upstream, and return the result to the caller. For MediaPipes, which can operate as filters, processing can take place before the result is returned. For MediaSources, which are the furthest upstream, this call may result in a system call to some data source outside the MediaObj pipeline, such as a network connection or filesystem. At this point, the call may block until information is available to pass down the pipeline.

- **Write** - Write is the complement of READ. When a source has data that is ready to pass through the pipeline, it calls the WRITE procedure of the object immediately downstream. The default action is for an object to recursively call the object immediately downstream, and return the result, but a MediaPipe might process the data before calling the next object, or a MediaSink could either accept the data immediately or block until it was ready.

Some out-of-band operations:

- **Start and Stop** - These operations let an application start and stop the flow of data through a MediaObj pipeline. When stopped, most MediaObj objects will block on a Read or Write until the Start operation is performed.

- **CallBack** - Each MediaObj object can notify the application about certain events by using a callback function. Each class of object has a unique set of events that the application must be prepared to recognize. A commonly implemented event is the "End-of-stream".

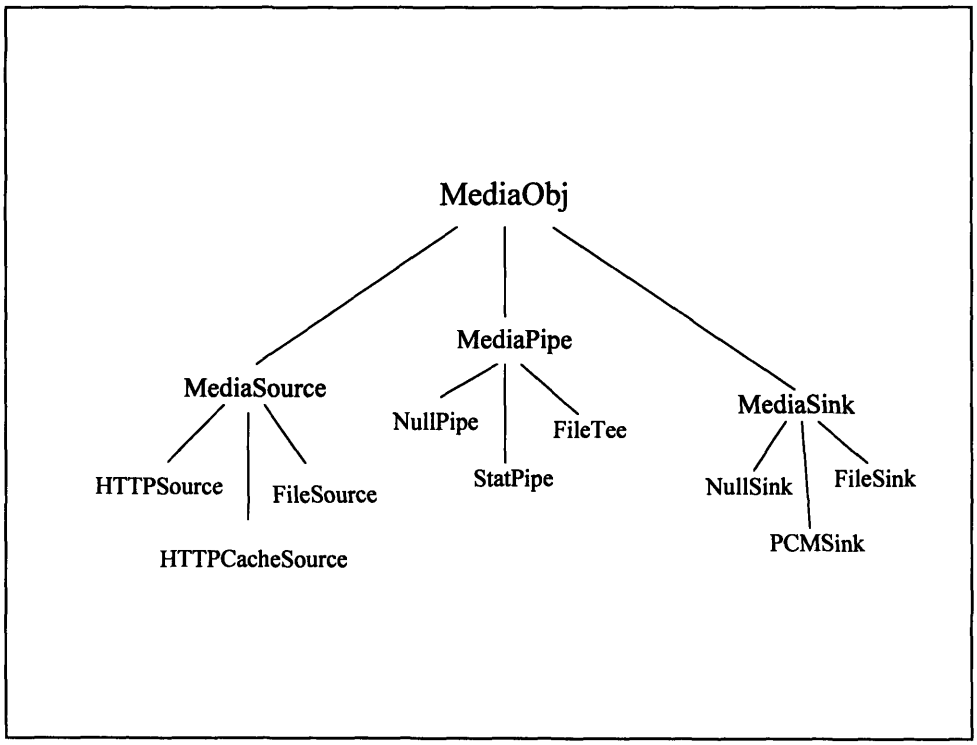

Figure 4. MediaObj Class Hierarchy

MediaSources

There are a number of MediaObj sources. These objects are generally specialized so that they retrieve data from outside of the MediaObj pipeline, typically from a network connection or from a disk file. In normal usage, they will be positioned at the front of a MediaObj pipeline.

- **HTTPSource** - HTTPSource is an object that opens an HTTP connection to an HTTP server on a TCP/IP network. Upon initialization, it sends and receives the appropriate HTTP request and

response headers, and then forwards the data stream (consisting of the object) to the rest of the MediaObj pipeline. Currently, the HTTPSource does not attempt to perform any prefetching of data from the server, and only responds to READ requests. If the TCP connection blocks, the READ request will also block until more data arrives. HTTPSource also implements the HTTP proxy protocol, making it possible to use HTTPSource through caches and firewalls.

- **FileSource** - FileSource is an object that reads a data stream from a file in the local filesystem.

- **HTTPCacheSource** - HTTPCacheSource is a special object that enables local caching of files. It keeps a table of mappings from URLs to local files. At initialization, the object first checks to see if the file requested is available locally. If it is, then it replaces itself with a FileSource pointing to that file. If not, an HTTPSource is created with the proper URL. In addition, a FileTee (to be described later) is connected after the HTTPSource, so that the new file can be copied to local storage during retrieval. An important aspect is that this happens transparently; any objects located after the HTTPCacheSource will not know what object was actually used as a source. HTTPCacheSource does some additional bookkeeping so that the disk files can be maintained as a LRU cache.

MediaPipes and Filters

MediaObj pipes and filters are objects that perform operations on the media stream while remaining transparent to objects ahead of and behind them in the MediaObj pipeline. The pipes perform a minimal amount of in-band processing so that they will not impact the performance of the system as a whole.

- **NullPipe** - NullPipe simply passes data from its inputs to its outputs, without performing any additional processing.

- **StatPipe** - StatPipe is a simple pipe that provides statistics about the in-band data flow to the application. Currently, it keeps a count of the amount of data that has passed through it, and passes this data back to the application through a callback. In addition, it keeps a simple time-average of

the rate of data transfer. This information can be used by the application to provide the user with feedback about network performance.

- **FileTee** - FileTee is a pipe that writes a copy of the in-band data passing through it to a local filesystem. It is used by HTTPCacheSource to transparently copy a media stream to a local file.

MediaSinks

MediaSinks are objects that act as final destinations for media streams. They are connected at the end of the MediaObj pipeline, with no objects further downstream.

- **NullSink** - NullSink is a simple sink that allocates a buffer, and then pulls data as fast as possible from the MediaObj Pipeline. It is useful for testing NetPlayer functionality on hosts that do not have MPEG decoder boards. In addition, because the NullSink is not rate controlled, it can be used to test NetPlayer throughput.

- **PCMSink** - PCMSink is a data sink representing the Optibase PCMotion MPEG decoder board. The sink is a Pull sink that initializes the MPEG board and transfers data from the MediaObj pipeline when the MPEG board generates interrupts for more data. The object is also responsible for creating and destroying any buffers that it uses to pass data through the pipeline. The object also understands some simple commands such as Play, Pause, Resume, and Stop, and upcalls to the application on some events such as End of Media Stream. In the future, this may be enhanced to provide information about characteristics of the MPEG stream being decoded, or timing information. The PCMSink requests data at 1-3 Mbits/sec, depending on the video stream.

- **FileSink** - FileSink is a MediaObj object that writes the data received from the MediaObj Pipeline to a file on the local file system. It is approximately equivalent to a FileTee object connected to a NullSink object.

These MediaObj objects can be arranged in a number of useful configuration. NetPlay supports several of these, selected at run-time:

- **Simple network player** - An HTTPSource object is connected to a PCMSink object through a StatPipe object. This configuration is what is usually used to play MPEG streams over a network. The StatPipe is used to provide feedback to the user about performance of the system.

- **Throughput tester** - An HTTPSource object is connected to a NullSink object through a StatPipe object. This configuration is used on hosts that do not have MPEG decoder boards, and to test throughput of a client, server, or network. The StatPipe output can be used to record these performance statistics.

- **Copy to disk** - A FileTee object is added to the simple network player, allowing the client to record the stream to disk as it is being viewed. This is usually used as part of a media caching system that will be described later, controlled by a HTTPCacheSource.

- **Read from disk** - The HTTPSource in the simple network player is replaced by a FileSource, so that the stream is read from disk instead of over the network. This is also usually used as part of a media caching system.

- **Write directly to disk** - A FileSink object replaces the PCMSink object in the simple network player. This functionality is used by the caching system to preload the media cache.

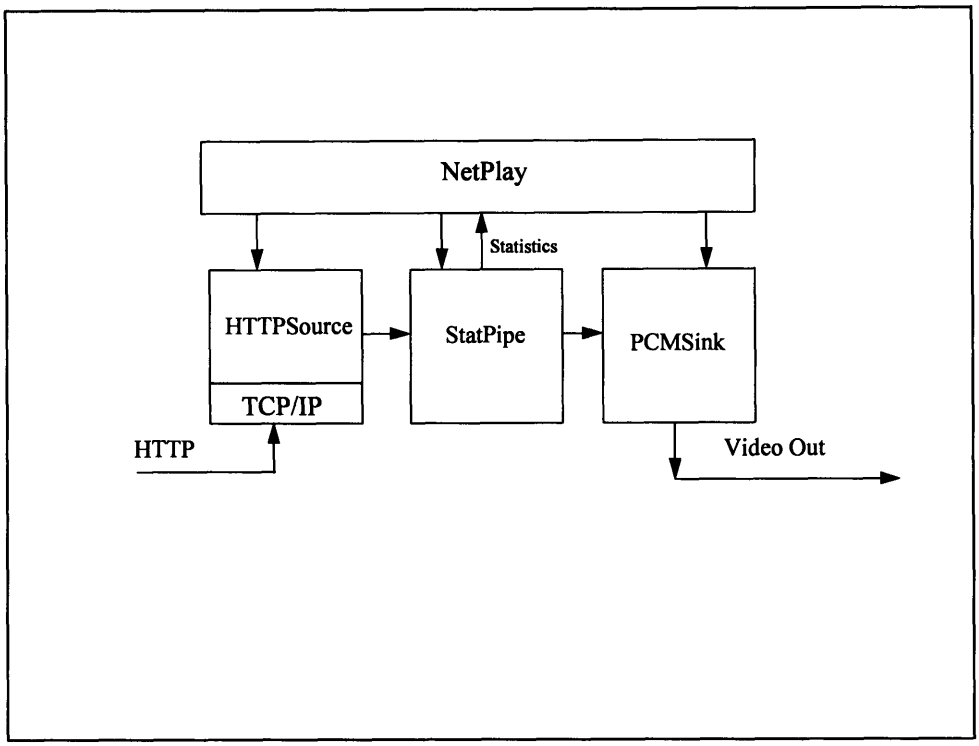

Figure 5. Pipeline configuration for simple playback

Caching and dynamic pipeline manipulation

NetPlay can perform limited caching of media streams on a client's local filesystem. This involves maintaining a simple LRU table, and configuring the MediaObj pipeline appropriately at run-time. This functionality is provided in the HTTPCacheSource object.

When a URL is passed to the NetPlayer, the HTTPCacheSource first checks to see if it is listed in its table of streams that are stored locally. If it is, the HTTPCacheSource creates a HTTPFileSource which it connects to the head of the MediaObj pipeline, and playback continues from the local storage. If not, HTTPCacheSource creates an HTTPSource and retrieves the header of the object, which includes the object size. If the object is past a certain threshold size, it is played directly without copying it to disk. If the object is smaller than the threshold, a FileTee is created and inserted between the HTTPSource and the rest of the pipeline. At this time, the cache table is consulted and media stream files are evicted from the cache if needed to keep the size of the entire cache below a specified limit. The stream is then played and copied to disk simultaneously.

Because the files are never modified locally, when they are evicted from the local cache, they are simply discarded. In addition, only complete media stream files are cached. If a media stream is interrupted during playback (a fairly common occurrence), the partially stored file is discarded, and no attempt is made to capture the rest of the stream.

Because of the relatively small size of most client caches, and because there tends to be little locality of reference for most video clips, the local cache tends not to be useful as a performance optimization. The main motivation was so the client could be preloaded with a small set of video segments, and then be disconnected from the network and remain useful. Also, for sites without secondary caching (described below), this would enable playback of video even with network connections slower than the playback rate by preloading media clips at less than real-time speeds, and playing them back from local storage.

Applications

There are a number of applications that have been developed using this architecture:

CNN Internet Newsroom

CNN Internet Newsroom [Compton95] is a networked, digital version of the CNN Newsroom television program. CNN Newsroom is a daily news program targeted at students in middle and high school current-events classes. It is broadcast via satellite daily at 3 am. In addition, CNN creates a teacher's guide which is e-mailed to subscribers over the Internet.

Teachers use the current television show in many ways in their classrooms. One typical usage is for the teacher to videotape the program automatically early every morning, and then review the content and teaching guide for that day's class. Teachers also can mail requests for particular segments from the CNN archives in Atlanta, Georgia. After a turnaround time of approximately two weeks, they receive a videotape of the segment requested that they can then use in class.

CNN Internet Newsroom provides a version of CNN Newsroom that is available to teachers and students over the World Wide Web. Much of the research and development behind the application has been focused on reliably converting regular programs into a hypertext digital format with as much automation as possible.

Currently, CNN provides the video, time codes marking the beginning and ends of segments within the program, and an electronic copy of the teacher's guide. An application uses the time codes to segment the program, digitizes and MPEG compresses the segments automatically, and then creates a hypertext document including the teacher's guide and the close-captioned text. These items are then placed on the Media Server and made available through the World Wide Web. They are also linked into the CNN Internet Newsroom Library, and the text is indexed.

The application is accessed using World Wide Web browsers and NetPlay. The user is presented with a simple interface allowing one to watch the current day's program, browse the CNN Internet Newsroom library, or perform keyword searches on the library text.

When a user clicks on the URL of a video, HTTP server returns an URL reference object pointing to the real video file, with a special content-type that is recognized by the browser. The browser then start NetPlay and passes the URL reference object to it. NetPlay then opens a connection to the URL and streams it to the appropriate output device, as described above. When the video segment is completed, NetPlay exits. Meanwhile, while NetPlay runs, the browser may be used to examine other parts of CNN Newsroom. The startup latency is usually half a second or less.

CNN Newsroom has been deployed to several locations with a wide variety of network connection types. Streaming playback of 1.2 Mbit/sec video files is possible over 1.5 Mbit/sec T1 lines, and non-streaming playback is performed with reasonable amounts of delay over 1/2 T1 - speed connections.

Mechanical Engineering 2.70 instruction project

An prototype instructional application was developed for the Mechanical Engineering department. The purpose of this system was to use the World Wide Web to teach students how to use various machine tools properly. For this project, it was desirable that students be able to access video from campus UNIX workstations not running NetPlay, so lower bandwidth (0.5 Mbit/sec) video-only files were digitized that could also be played back using software-only decoders. In contrast to the immediate playback possible using NetPlay, using regular World-Wide-Web tools forced the user to wait for the entire clip to download before playback, which resulted in a wait of nearly a minute, even for these extremely short video clips on the high speed campus network.

CMU Informedia Interchange

A demonstration gateway between the CMU Informedia digital video library and the NMIS CNN Newsroom system was developed and tested. This involved building a gateway at the library searching level, and at the video distribution level [Soo95]. The approach taken at the video distribution level was to

use an HTTP caching proxy server as a gateway between the two systems. The CMU Informedia system is similar to the NMIS system in that the video is encoded using MPEG-1 at approximately 1 Mbit/sec. The main difference is that the current implementation uses the NetBIOS protocol for transport of the video on the local area network, which does not allow the system to operate over the Internet. To solve this problem, an HTTP Server running on Windows NT Advanced Server was configured so that the server could deliver files using both HTTP and NetBIOS protocols. Informedia video files could then be accessed either through the Informedia browser on the LAN, or using NetPlay over the Internet.

Conclusions

The NMIS networked multimedia distribution architecture is a useful one for delivering multimedia over the Internet efficiently. The NetPlay Media Player is a tool that is easy to integrate into existing systems, and can solve many of the problems involved in distributing networked multimedia over the Internet. The NMIS networked multimedia distribution architecture, and the NetPlay Media Player conforming to that architecture, satisfy a number of the goals that were described in the introduction, among which the following are the most significant:

- **Open system architecture** - This architecture builds on existing open protocols and standards, most importantly, TCP/IP, HTTP, and MPEG. Importantly, the system defines interfaces, rather than implementations; many of the parts, such as the video format, are replaceable. Some of the advantages of this are that multiple implementations have been shown to work with each other, such as HTTP servers from multiple vendors. A related advantage is that by using existing standards where possible, it is possible to leverage off of existing products, such at TCP/IP protocol stacks and HTTP servers.

- **Strong interfaces** - The interfaces that have been designed have been shown to be well-enough defined to be used as a starting point for interoperability. The CMU Interoperability demonstration shows that it is possible to use this architecture as a way of allowing transparent access to different systems, with a single user interface [Soo95].

- **Generality** - The implementation has been shown to be general enough to support a range of applications, from a digital news magazine to instructional video to digital video libraries. This has occurred without having to make any modifications to the actual video distribution

mechanisms in the system. There is are not limitations in the system regarding the types of streaming multimedia that can be delivered.

- **Efficiency** - The implementation has been shown to be able to deliver video efficiently over a wide range of bandwidths. It also appears that it should be capable of scaling up significantly to support much higher quality media streams. In the opposite direction, the client implementation is capable of offering full performance on low cost platforms.

- **Ease of implementation** - Because this architecture leverages so strongly off the World Wide Web, it appears to be very simple to implement. Because this level of functionality does not require any modifications to HTTP servers, it is conceivable that any current Web server could become a video server provided that the bandwidth and storage requirements of networked multimedia are required. Similarly, existing HTTP Caching proxy servers appear to be implemented in such a way that they are easily used as local video caches, given enough bandwidth and storage.

On the client side, the MediaObj pipeline architecture appears to be very powerful and flexible. It should be fairly straightforward to implement new MediaObj objects such as sinks for other video decompression cards. Because it runs on a popular platform and interfaces easily with all types of HTTP browsers, it is fairly easy for NetPlay to be used with any Web browser desired.

Ease of use and authoring - Because the NMIS architecture is Web based, it has the advantage of being a familiar environment to electronic publishers and of having a large base of authoring tools. The video distribution is an incremental addition to the system, rather than a revolutionary change.

Future development

There are a number of areas that need to be addressed in future development, some of which are architectural issues, and some of which are implementation issues.

Browser - NetPlay Interface

In the current implementation, an extra HTTP transaction is required beyond what is strictly necessary. This is results in additional latency for the user, and also additional load on the server. For the server maintainer, this also results in the need to maintain an extra file, the .URL file that contains the reference to the actual video. In addition, it is confusing to users that are not using NetPlay to access the video files, because they are not provided any references to the URL of the video file without interpreting the .URL file. An approach that was attempted but not completely successful was to use content-type negotiation between the client and server. Ideally, the browser could use the Accepts HTTP header to indicate whether they were configured to run NetPlay. Unfortunately, most browsers do not pass enough information about what content-types they can handle to the server, so it was not possible to do this reliably.

On some platforms, notably Microsoft Windows, browser manufacturers are starting to provide better interfaces between browsers and helper applications such as NetPlay. Hopefully a future version of NetPlay will be able to communicate with browsers and receive streaming data directly, rather than having to use their own HTTP connections. This would reduce the transaction overhead, as well as simplify NetPlay itself.

Advanced Browser Functionality

Currently, NetPlay can only support playing a single video file from beginning to end. It is possible to simulate "Rewind to beginning" by opening a new connection to the server, but this is a very limited form of control. In Internet CNN Newsroom, users are provided limited random access by physically segmenting video files into relatively short pieces, typically 30 seconds to a few minutes long. Unfortunately, this means that it is not easy to view the entire program at once, in an uninterrupted manner, and often a finer granularity of control is desired. There are two opposite approaches to solving this problem; one is to provide mechanisms to segment video files into smaller and smaller pieces, and provide mechanisms to play back sequential pieces seamlessly. One advantage of this approach is that, because the individual objects are smaller, it may be easier to cache them at various levels of the hierarchy. Because entire files would still be transferred, it would require no modifications to the server-side software. On the

other hand, playing back multiple small objects would certainly increase the overhead and management cost, and would still not provide arbitrary random access; entry points would still need to be computed at encoding time.

The opposite approach is to add support for starting play from an arbitrary point within a file. This would allow truly random access to the material at playback time, and simplify storage management. However, this would require some modification to software on the HTTP server, and the design of a protocol to describe where to start access. Additionally, there are some serious caching issues involved because current caching proxy servers only cache on a file by file basis.

Probably some combination of the two approaches is necessary to provide the performance and flexibility desired. For most material, there are probably "Chapters" of information that can stand alone and make sense as a single isolated material, and it is desirable to be able to play back a series of these objects. However, many applications will want access at a finer grain than that, and that should be supported in an efficient manner.

Application / Scripting Support

NetPlay is currently viewed as a standalone application that can be called by Web browsers to provide video playback support, and the usage model is user-initiated browsing. This is a needed and very usable first step. However, many applications will require much more sophisticated interaction between the user, playback mechanism, and application. For instance, the Dartmouth Interactive Media Laboratory is converting an interactive laser-disk based application to the NMIS architecture. This application is designed to teach rural medical practitioners how to treat infant respiratory problems. The model of usage is more like a video game than a magazine; there are many possible paths through the application, and at some times the application itself will initiate interaction with the user, such as during a quiz.

It is necessary to provide applications access to the functionality of NetPlay through a more flexible mechanism than is available today. One approach is to create an interface to the MediaObj objects through a scripting language. In this approach (give reference describing Jimmy's work), MediaObj objects are

encapsulated in higher levels in a dynamic, object-oriented language. Using this mechanism, MediaObj objects can be created and manipulated at run-time by higher level applications. To the user, the video streaming mechanisms become completely and transparently integrated into whatever application they are using. The applications programmer simply sees a way to use those tools flexibly, without depending on any particular arrangement such as that used in NetPlay.

This development is necessary to create more advanced applications, much as the BSD Sockets library made it easy for programmers to create new data networking applications. If programmers need to write these low level components from scratch, or use fixed arrangements such as in NetPlay, development will proceed much more slowly and much work will be duplicated.

Wider Media Support

Currently, NetPlay only supports a single data type, streaming MPEG video. This is not because of an architectural reason, but because that is the only type of data that programming resources have been focused on to date. Streaming playback is feasible and desirable for many other types of content, notably audio, but also including data such as closed-captioning information. In some cases, it will be desirable to have multiple streams of data in parallel, such as when streaming audio and text during a multimedia presentation. More development needs to be done to expand the MediaObj library to support these data types.

Advanced Cache Management and Replication

The NMIS architecture currently only supports user-initiated demand based caching. Because caching proxy servers are capable of serving data as soon as it is received, and because we have a small number of users, operating on high-speed links, this is acceptable. However, there are situations when a more sophisticated cache management system is desired. One is when the bandwidth between the main server and proxy servers simply is not enough to sustain real-time playback. In that case, for some material, it is desirable to stream data at a slower rate well before it is needed. For instance the CNN Newsroom application, a "subscriber" should be able to get the daily news feed as soon as it is available early in the morning, as long as there is enough bandwidth to send it before it is needed. In this instance, a "Push"

mechanism initiated by the server is a means of doing this. This currently is not supported by existing HTTP proxy servers, and some modifications of client and server software may be necessary to do this.

Another important case is when a large number of clients from different sites wish to access the same information at the same time. This situation could arise when CNN Internet Newsroom is deployed to a much larger population. The problem here is that the main server simply may not have the bandwidth to transmit that much same information at once.

If the demand for that information is predictable (such as if many of these requests were for the current day's headline story), it may be possible to coalesce many of these requests into a single multicast or broadcast. For example, if most of the requests are by daily subscribers to CNN Internet Newsroom, one strategy would be to use Internet Multicast or some other multicast or broadcast technology to send all of these subscribers the data at once, with only one request from the main server. This would allow video servers to support a much larger number of users than otherwise possible. One difference between this approach and the cable television VOD and MBONE Internet multicast approaches is that the broadcast is used to replicate cache entries, rather than to do final distribution to end users. This has the advantage that, first, a smaller set of hosts must be changed to support this usage, and second, end users are not forced to access the material at the same time.

Broader interoperability

The NMIS media distribution architecture has been shown to interoperate well with a range of systems designed for an array of uses, from digital libraries to simple video on demand. However, there are many types of systems that this implementation has not yet interfaced with. For instance, work needs to be done to show how it might be interfaced to a state-of-the-art cable or telco video-on-demand system. A simple interface to the MIT analog cable system is currently being implemented; it would also be interesting to build an interface to an advanced commercial video server.

Live Media

Although the current implementation of the NMIS architecture does not support "Live" media streams, there is no inherent reason why they could not be added. The current limitation is in the MPEG encoding system itself, which does not yet support real-time networked encoding. Live media transport would make it possible for the NMIS architecture to support a broader range of applications, notably interactive live remote learning.

Conclusion

It is important today to take the lessons that have been learned from the design, implementation, and growth of the Internet, and apply them to a broader set of applications. The NMIS Media Distribution architecture shows that it is possible to provide support for a wide range of network applications over the existing infrastructure, with a few key improvements to existing components. At this time, a limited realization of this architecture has been implemented and tested. Although there are several areas where further development is necessary, there do not seem to be any insurmountable obstacles to providing the desired service, which consists of user-friendly, efficient multimedia distribution, and open access and standards. The implementation currently provides support for several interesting applications, and hopefully, new applications will influence the development of the architecture, and the architecture will influence the develop of new applications.

References and Bibliography

[ANKL] Anklesaria, F. et al "The Internet Gopher Protocol", Internet RFC 1436

[Berners-Lee94a] Berners-Lee, T., Fielding, R. T., and H. Nielsen. *Hypertext Transfer Protocol (HTTP)*. Internet Draft, 1994

[Berners-Lee94b] Berners-Lee, T. and Daniel Connoly. *Hypertext Markup Language: A Representation of Textual Information and Metainformation for Retrieval and Interchange*, Internet Draft, 1994

[Bosco93] Bosco, P. et al. *Research In Networked Multimedia Information Services*, Proposal to the National Science Foundation and Advanced Research Projects Agency, 1993.

[Chang94] Chang, Yee-Hsiang et al. "An Open-Systems Approach to Video On Demand", *IEEE Communications Magazine*, May 1994

[Clark88] Clark, David D. "The Design Philosophy of the DARPA Internet Protocols", Proceedings of the 1988 SIGCOMM Symposium, pp. 106-114

[Compton95] Compton, Charles L. *Internet CNN Newsroom: The Design of a Digital Video News Magazine*, Master of Engineering Thesis, May 1995

[Dan94] Dan, Asit, Sitaram, Dinkar and Perwez Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching", *Research Report*, IBM Research, 1994

[Federighi94] Federighi, Craig and Lawrence A. Rowe, "A Distributed Hierarchical Storage Manager for a Video-On-Demand System", *SPIE Symp. On. Elec. Imaging Sci. & Tech*, San Jose, CA, February 1994.

[Houh95] Houh, Henry H., Joel F. Adam, Michael Ismert, Christopher J. Lindblad, and David L. Tennenhouse, "The VuNet Desk Area Network: Architecture, Implementation and Experience", *IEEE Journal of Selected Areas in Communications*, First Quarter, 1995

[NMIS] *Networked Multimedia Information Systems Home Page*, http://www.nmis.org/

[Optibase94] *PCMotion MPEG Playback System, User's Manual*, Optibase, Inc. 1994

[Postel81] Postel, Jonathan B, Sunshine, Carl A. and Danny Cohen. "The ARPA Internet Protocol" *Computer Networks 5*, Vol 5, No. 4 1981

[Rowe92] Rowe, Lawrence A. and Brian C. Smith, "A Continuous Media Player", *Proc. 3rd Int. Workshop on Network and OS Support for Digital Audio and Video*, San Diego CA, November 1992

[RPCP] *Research Program on Communications Policy*, http://farnsworth.mit.edu/

[Satya90] Satyanarayanan, M. "Scalable, secure, and highly available distributed file access" *Computer*, May 1990

[Soo94] Soo, Jonathan C. "Live Multimedia over HTTP", *Proceedings of the Second International World Wide Web Conference*, October 1994

[Soo95] Soo, Jonathan C. Private notes, ARPA meeting, April 1995

[Wactlar94] Wactlar, H. et al. *Integrated Speech, Image and Language Understanding for Creation and Exploration of Digital Video Libraries*, Proposal to the National Science Foundation, 1994