AFRL-IF-RS-TR-1998-101
Final Technical Report
May 1998



VIDEO ON DEMAND TECHNOLOGIES AND DEMONSTRATIONS

Syracuse University

Marek Podgorny and Geoffrey C. Fox

19980911 004

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1998-101 has been reviewed and is approved for publication.

APPROVED: Alan J. Akins

ALAN J. AKINS
Project Engineer

FOR THE DIRECTOR: Warren

WARREN H. DEBANY, JR., Technical Advisory
Information Grid Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFGA, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED	
	June 1998	Final	Sep 94 - Nov 96

4. TITLE AND SUBTITLE

VIDEO ON DEMAND TECHNOLOGIES AND DEMONSTRATIONS

5. FUNDING NUMBERS

C - F30602-94-C-0256
PE - 62702F
PR - 4519
TA - 22
WU - 36

6. AUTHOR(S)

Marek Podgorny and Geoffrey C. Fox

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Northeast Parallel Architectures Center (NPAC))
111 College Place
Syracuse University
Syracuse NY 13244-4100

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFGA
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-1998-101

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Alan J. Akins/IFGA/(315) 330-7751

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This effort demonstrated a functional video-on-demand (VoD) service running over a wide-area high-speed Asynchronous Transfer Mode (ATM) network. This report presents an overview of the efforts key results and accomplishments. the overall architecture of the video delivery service is described, including the implementation of the video search and retrieval methodology. The video server and video client architectures are discussed in detail, describing the options and paths chosen. Also included in the report is an overview of the hardware and network infrastructure that was built to provide the project testbed along with the wide area network (WAN) demonstrations used during the effort. Separate areas of the report provide information on several fundamental research issues such as Variable Bit Rate vs. Constant Bit Rate transmission, wavelet compression technology for video, I/O support for scalable coders, and multimedia networking.

14. SUBJECT TERMS

Asynchronolous Transfer Mode (ATM), digital video, network delivery, video server, video client, video compression, multimedia networking distributed server

15. NUMBER OF PAGES

332

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

1. Introduction 1
2. Project Summary 3
   2.1 Funding sources 7
3. Global Video-on-Demand Architecture 9
   3.1 Interaction of the system components 10
   3.2 Metadata concept and database back-end 12
   3.2.1 Video sever asset management 13
   3.2.2 Indexing and search interface 18
   3.2.3 Data flow model 19
   3.2.4 Contents creation service 24
4. Video Server Implementations 31
   4.1 Early server designs 31
   4.1.1 Network Dataflow in a VoD System 31
   4.1.2 Video Server Configurations 33
   4.1.3 Software Architecture of a VoD Server 34
   4.1.4 Optimizing the VoD Server 35
   4.1.5 Server implementation on SGI platforms 40
   4.2 Production Video server at NPAC 41
   4.2.1 Completion Ports in Ms Windows NT 41
   4.2.2 Production version if the Windows NT Video Server 44
   4.3 nCUBE2 Video Server 45
   4.3.1 NCUBE2 network connectivity 47
   4.3.2 Video Server Implementations on nCUBE 48
   4.4 Distributed Server Architecture 52
   4.4.1 Server Design Issues 55
   4.4.2 Oasis Monitoring Tool 65
   4.4.3 Client implementation for Oasis Media On Demand Server 74
   4.4.4 Implementation status of the distributed video server 84

5. Video Client Architectures 85
5.1 Video client for SGI platforms 86
5.1.1 Digital Media Library 86
5.1.2 MPEG1 Video Client for SGI workstations 88
5.1.3 Low bit rate, H.263 video client for SGI workstations 93
5.2 Video for Windows networked video client 95
5.2.1 Video for Windows architecture 95
5.2.2 MPEG decoders 97
5.2.3 Windows networking tools 98
5.2.4 Media Control Interface Optibase MPEG driver 98
5.2.5 MCI driver in "Video for Windows" architecture 99
5.2.6 Technical details of the driver 100
5.2.7 Execution Flow 101
5.2.8 Parsing metadata file 102
5.2.9 Networked Video Client application 103
5.2.10 Random access 104
5.2.11 AVI Video Client 106
5.3 Codec independent Active Movie video client 106
5.3.1 Active Movie Technology 107
5.3.2 Video Client in Active Movie 116
5.4 Web-specific video client architectures 129
5.5 VoD versus LAN video broadcast: hierarchical multicasting video client 130
6. VoD Testbed Infrastructure 133
6.1 Video Server assets 133
6.2 NPAC ATM LAN 134
6.2.1 Living Schoolbook Project 135
6.2.2 Rome Laboratory 136
7. Selected VoD Research Issues 139
7.1 CBR transmission of VBR encoded video compression 139
7.2 Hybrid wavelet – H.263 video compression 141
7.3 a continuous media file system for multi-resolution data 141

8. Multimedia Networking: Integrated Services Model 143
8.1 Introductory Multimedia Networking Module 145
8.2 Multimedia Networks and Integrated Services Model 155
8.3 Multicast and Switched Network Basics 197
8.4 Reservation Protocol, Parts I & II 211
8.5 Real Time Protocol 243
9. Software packages 271
10. Appendices 273
10.1 Appendix 1: CBR Transmission of VBR Encoded Continuous Media 275
10.2 Appendix 2: Hybrid wavelet-H263 video compression 327
10.3 Appendix 3: A continuous media file system for multi-resolution 349
10.3.1 Introduction 349
10.3.2 Traditional File System 349
10.3.3 The CFS file System 350
10.3.4 Terminology 350
10.3.5 Multi-resolution Support 350
10.3.6 Data Layout Policy 351
10.3.7 Data Placement Policy 353
10.3.8 The directory Service 354
10.3.9 Admission Control Policy 355
10.3.10 The physical Scheduler 357
10.3.11 Interactive Support for Continuous Media 358
10.3.12 Quality of Service (QoS) Support 358
10.3.13 Comparison of CFS and XFS 358
10.3.14 Conclusion 359
10.4 Appendix 4: Video Server Asset Management Screen Dumps 361
10.5 Appendix 5: Video Database Search Interface Screen Dumps 367
10.6 Appendix 6: Video Clients Screen Dumps 373
10.7 Appendix 7: Details of the Ncube2 OS network layer architecture 379
10.8 Appendix 8: Slides of the final project presentation 389

1. Introduction

This document describes the main results of the project "Video-on-Demand Technologies and Demonstrations." It was funded by the USAF Rome Laboratory under the contract number F30602-94-C-0256. The project's time frame was 24 months, starting in October 1994. It was conducted at NPAC, Syracuse University, and was led by Geoffrey C. Fox and Marek Podgorny.

The main objectives of the project were (1) to develop and evaluate existing base technologies and (2) to build a prototype video-on-demand (VoD) server to deliver digital video streams over the NYNET regional ATM network between Syracuse University and Rome Laboratory. More specifically, the following issues were addressed:

1) Hardware and software requirements of the video server were determined and evaluated, including video capture, indexing, random access, searching, retrieval, and storage.

2) Hardware and software requirements of the Video Network Service were determined and evaluated, including

   - A. an investigation of ATM and FDDI technology to evaluate network bandwidth and latency requirements for high resolution VoD;
   - B. evaluation of existing standard transport protocols and their suitability to support VoD applications;
   - C. evaluation of network topology to achieve optimal performance in access to video storage, video server functions, and delivery of VoD services to geographically dispersed users across a wide area network; and
   - D. evaluation of existing network management technology for managing VoD.

3) Evaluate parallel, distributed, and heterogeneous computing to support VoD applications.

4) Evaluate and implement state-of-the-art user-interface technologies for VoD applications.

5) Upgrade existing VoD demonstrations to run in an HPCC environment (parallel servers, WAN ATM network delivery).

6) Evaluate, benchmark, and demonstrate the WAN ATM delivery of the VoD prototype service.

This report is organized as follows:

Section 2 (Project Summary) presents an overview of project's main results and accomplishments, including the logical framework of the project, a description of project progress, explanations research direction decisions taken during the project, technical results of the effort, and a list of project deliverables. The overview is deliberately concise but refers the reader to appropriate technical sections of the report. Subsequent sections provide an in-depth discussion of the research and technical issues, a presentation of the software architectures created in the project, and a detailed discussion of the research results. After reading the Project Summary section, the extent of the research effort should be clear to the reader. Technical sections of the report may be read selectively, as they are largely self-contained.

Section 3 (Global Video-on-Demand Architecture) is the first of several technical sections; it describes the overall architecture of the video delivery service and, in particular, the implementation of the video search and retrieval methodology based on the database back-end. **Section 4** (Video Server Implementations) discusses video server implementation. **Section 5** (Video Client Architectures) is devoted to video client architectures. **Section 6** (VoD Testbed Infrastructure) provides an overview of the hardware and network infrastructure built to provide a project testbed; this section also lists the WAN demonstrations set up during the project. **Section 7** (Selected VoD Research Issues) provides a summary of several fundamental research issues addressed in the project. The subjects include Variable Bit Rate vs. Constant Bit Rate transmission issues, wavelet compression technology for video, and I/O support for scaleable coders. For some of these subsections, the more detailed material is presented in appendices. Finally, **Section 8** (Multimedia networking: Integrated Services Model) documents our study and evaluation of multimedia networking issues in the form of lectures delivered at NPAC.

2. Project Summary

The main task of the project was to implement a functional video-on-demand (VoD) service running over the NYNET wide-area ATM network. This task has been completed; and the completed system exceeds project requirements in two aspects:

First, we have built an operational digital video service, not merely a prototype. This service has been operated continuously by NPAC since its inception in late 1995 and remains operational. Based on its continued availability, the service is being used by the Living Schoolbook (LSB) project and several other ongoing projects. In addition, the service is being improved by addition of new elements, gradual incorporation of certain theoretical ideas developed in the project but not yet implemented, and by integration of the service into other current projects. Most noteworthy, the benefits of the video-on-demand service are being leveraged by its inclusion in the "Collaboration and Interactive Visualization" project sponsored by the Rome Laboratory.

Second, the project deliverables include operational software. Although not of commercial quality, the developed software package is sufficiently robust to warrant limited distribution. Some of the packages have been installed at LSB sites, where they are expected to go into production mode in the near future. <The software packages for the video server, video clients, and video-server database back-end are attached to this report.>

In spite of this successful outcome, the project had its share of difficulties. Before reviewing the statement of work requirements and deliverables, it is useful to assess the project results in relation to its initial goals. As perceived at the project onset, the main tasks were: (1) to build a demonstrable, end-to-end VoD system; (2) to test the system in the context of real VoD applications; and (3) to concentrate research efforts on the ATM aspects on the video network delivery. Looking at this list after the project has been completed, we offer the following observations:

- A complete VoD system has been built. As indicated above, we believe this system to exceed the initial requirement for a prototype system, as the created software can be successfully distributed.
- The application-level testing was minimal. All testing took place at NPAC. The main reasons for this outcome are both technical and sociological. Technically, stagnation of the ATM technology (see below for a broader discussion) made it economically non-viable to provide ATM-based VoD service to the LSB testbed. This situation improved only recently and the LSB project has moved rigorously to take advantage of this progress. In addition, we have experienced difficulties in stimulating broader use of the VoD technology in the LSB project. These difficulties result largely from the number of project participants who, being new to

the Web technology and advanced multimedia technology, used limited video-downloading procedures rather than new streaming technology. It appears that the high bandwidth provided by NYNET actually promoted this approach and that the advantages of a fully interactive, searchable video archive is not regarded as critical from the point of view of curriculum designers. This situation is changing, mostly as a result of technology demonstrations provided to the LSB teachers.

- The ATM aspects of the network delivery did not play any essential role in the project. This does not imply that we have failed to build an ATM video-delivery service; in fact, such a service has been implemented. The main point is that, although the project carried out an extensive evaluation of new ATM solutions, very little creative ATM-related research has been done. The reasons for this situation can be summarized as follows:

  1) During the project time frame, there was almost complete stagnation of the ATM technology. Classical IP and incremental improvements in ATM NICs performance are the only progress we have seen. The seemingly endless negotiations on standards within the ATM Forum as well as vendors' difficulties in implementing early specifications of standards like LAN Emulation made it impossible (or, at least, economically nonviable) to build an extensive ATM-based VoD testbed. We discuss the ATM issues in Section 6.2.

  2) The failure to progress in ATM development has resulted in a lack of ATM applications, i.e., the applications using the ATM Adaptation Layers and bypassing the TCP/IP stack. The dominant mode of using ATM networks is TCP/IP over permanent virtual circuits. Our project took identical route. Due to the rather high bandwidth and very low load on NYNET, the ATM links available to us provided sufficient support for the wide-area video delivery, even for multiple streams of high bit rate codecs. The really difficult problems in implementing the service were not related to ATM but to the software architectures needed to feed the network and then receive and present digital video. Further, had we decided to pursue ATM-oriented protocol research, it would not have been possible within the project's time frame to build an operational prototype system. The research community now recognizes these issues as important and will soon address them.

  3) In contrast to the ATM stagnation, there was an explosive development of the multimedia-related protocols for IP networks. This work, mostly concentrated around IETF, brought new protocols such as RTP (Real Time Protocol),

RSVP (Reservation Protocol), Protocol Independent Multicast (PIM), and RTSP (Real Time Streaming Protocol). These protocols form a foundation for the Integrated Services Model (ISM) of the Internet. ISM will bring quality of service to packet-switched networks. It is not clear whether ATM will regain its leading role as the multimedia network technology of choice. We believe that even if this happens, the current ISM will remain an important mechanism for applications to request quality of service. Most likely, interfaces between the ISM protocols and ATM protocols will be built to utilize ATM's inherent QoS support. In the course of the project, we have evaluated extensively ISM and related protocols. This evaluation guided our implementation of certain solutions for the VoD system., and it convinced us that ATM-related protocol work is not a good investment of human resources in a project that requires an operational prototype as a main deliverable.

- Instead of the ATM research, we have made a number of theoretical contributions to (a) the problem of efficient video-streams multiplexing, (b) the design of more efficient, scaleable video codecs, and (c) the issue of designing an efficient file system for multi-resolution data. This research is important to the design of scaleable video servers.
- Research focused on three areas: video server design, video client technology, and use of Web technology as middleware to build a coherent, distributed digital video library. The Web technology application for implementation of such a system is successful and extremely efficient. In the video server and video client research, we encountered a number of difficult problems which have been only partly solved.

The remainder of this Section briefly outlines project results and deliverables for all the tasks mandated in the original project. The task numbers refer to the task definitions quoted in the Introduction.

1) We identified, developed, and integrated technology elements so as to built a complete digital video-delivery system. The system uses the IP protocol and runs over both broadcast and circuit-switched networks, including ATM. The system entails the following functional modules:
   - Contents-production module: the setup provides video digitization and encoding from multiple sources to multiple video codecs. For a description, see Section 3.2.4.

- Video-server module: video server software has been created for SGI SMP and Windows NT platforms. For a description, see Section 4.1.5-4.2. An attempt to implement a video server for nCUBE failed due to insufficient system performance (Section 4.3)
- Video-client module: network video clients have been created for UNIX (SGI) and PC platforms. We provide support for both hardware and software decoders. Video clients have been implemented as stand alone applications, Netscape browser plug-ins, and even as Java applets (Section 5).
- Database back-end: the heart of the digital video library, the database back-end supports search, random access, and asset management (Section 3.2)

The video delivery system built in this project supports automatic video indexing via closed captioning, random access to video contents based on metadata search, video retrieval using multiple access methods, video server monitoring, and a video-asset management layer.

2) To implement a digital video network-delivery service, we developed and evaluated a family of solutions for network video clients. In addition, we evaluated current ATM support for multimedia traffic and QoS issues for the IP networks (Integrated Services Model). For a summary of findings, see Section 6.2. In the final phase of the project, we designed and implemented an ATM testbed for performance benchmarking of our VoD setup (Section 6). In addition, we have addressed the following research issues related to the future of video-on-demand architectures:

- Variable bit rate vs. Constant bit rate transmission (Section 7.1),
- Wavelet compression for video (Section 7.2),
- I/O support for scaleable coders and adaptive applications (Section 7.3).

3) We evaluated three different platforms for video server implementation, including nCUBE2 MPP, SGI symmetric multiprocessor, and an ATM cluster of Windows NT PC multiprocessors. We implemented video servers on Windows NT and SGI platforms and evaluated and benchmarked different multithreaded video server architectures. Finally, we have provided a blueprint for a low cost, scaleable, distributed, dynamically reconfigurable cluster VoD server. For details, see Section 4.

4) We designed, implemented, and evaluated a Web-based interface to the VoD system. We developed an integrated video delivery system with indexed, random

access. The system entails automatic video indexing capability, relational database back-end, a CGI-extended web server as a database access agent, and a web browser as user interface. We also developed a variety of video-client user interfaces. See Section 5.3.2.8 for details.

5) Demonstrations of the evolving capability of VoD were available over an ATM network beginning almost with the project's inception. We tested and installed all appropriate ATM software upgrades available from the vendors, and we constantly tuned NYNET performance to deliver the video-on-demand service. In the final stages of the project, we moved to ATM LANE topology. At present, VoD service is delivered to participants at the Living Schoolbook Project. For details, see Section 6.2.

6) We demonstrated video delivery over an ATM WAN on a number of occasions and worked with ATM vendors to set up demonstrations from Syracuse to several locations, some as remote as San Diego, Atlanta, and Washington, DC, as well as to the Syracuse OnCenter and to Rome Laboratory.

2.1 Funding sources

This project was funded by the Rome Laboratory under the contract No. F-30602-94-C-0256. A fraction of the work described was co-funded by Rome Laboratory contract No: F30602-95-0-0273 (Collaboration and Interactive Visualization - CIV). The basic technology used in this project supports various aspects of the CIV project and is being developed further.

3. Global Video-on-Demand Architecture

Implementation of a network digital-video delivery system is perceived as a system integration issue. It is often assumed that the component technologies needed to built such a system are available and the only thing that remains to be done is connecting component parts into a working system. In general, we do not share this view. The following parts of this report address the implementation issues for system components such as the video server, the video client, and the asset-management module. For the purpose of this section of the report, however, we will assume that the basic system components exist; and we will proceed with description of our system integration strategy.

The system we implemented is intended to support applications such as education, corporate and military training, as well as the distribution of video information of transient value, such as news clips. Entertainment systems are beyond the scope of this project. This focus implies that the targeted audience is assumed to have functional expectations and technical capabilities significantly exceeding those of an interactive program guide user. In essence, our goal was to build a digital library of documents based on video contents. We believe that our can serve any other stream multimedia information with equal ease.

The basic components of the network video-delivery system are:

- Video Server: an entity capable of delivering multiple concurrent, continuous media streams. In our approach, the video server has the following important properties:

  1) the video-server data repository holds only multimedia data streams, not the metadata describing them;

  2) the server is codec independent, i.e., the media streams are opaque to the server. All random-access functionality is realized on the protocol level. The server may admit and regulate stream transmissions based on the stream bitrate defined in the client stream request;.

  3) the server uses a plain file system to store video files. Video database consistency is implemented in the Video Server Asset Management layer.

- Video Client: a software module capable of interactive playback of the media stream pushed by the video server. Video clients may handle multiple media stream formats. The client is responsible for issuing the stream request, initializing the decoder, presenting the stream, and interactive stream playback control.

- Database Back-end: a repository of the system metadata. The database repository holds all data other than continuous media streams. Metadata are used for (1) technical characterization of the stream, necessary for video clients to set up the playback mechanism properly; and (2) storage of all contents information related to a particular continuos media stream, such as closed captioning, annotations, synchronized URLs, etc. (The concept of metadata will be described in more detail later).

We used Web technology to integrate all components into a working system. Specifically, we used Hypertext Markup Language (HTML) and other client-side programming technologies such as JavaScript and Java to build a graphical user interface for all aspects of system operation. Also, we used Common Gateway Interface (CGI) technology to provide linkage between the HTTP server and the relational database back-end. For certain video clients, we use MIME mechanisms to spawn client instances.

Current Web technology does not provide support for certain functionality necessary to implement a complete video delivery system. For instance, there is no accepted standard for a protocol delivering media streams. In such cases, we implemented our own solutions. Very recently, work has been started to define such standards. We provide our view on how such standards are applicable to our work in Section 7: Selected VoD Research Issues.

3.1 Interaction of the system components

Figure 3.1 illustrates the high-level architecture of the system.

The Video Client component communicates with two server components: Database Server and Video Server. The Database Server is responsible for providing to the Video Client all information necessary for the client to retrieve the video stream. Based on that information, the Video Client sets a session with one of the Video Servers in the system. During the session, the client interactively retrieves and presents the media stream.

The data flow model presented in Fig. 3.2 consists of two consecutive phases of user interaction with the Video-on-Demand system. Phase 1 searches for video material and presents the user with a hit list retrieved from the database according to the user specified search criteria. Phase 2 actually displays the selected media stream. The first phase is realized in communication between the Video Client and the Database Server; the second is realized in communication between the Video Client and the Video Server. The two phases are essentially independent, except that the second phase is invoked by the first one. For videos with associated time–stamped textual data, additional coupling exists between the two phases: each time stamp can serve as a stream access point, allowing for interactive, indexed random access to video streams.

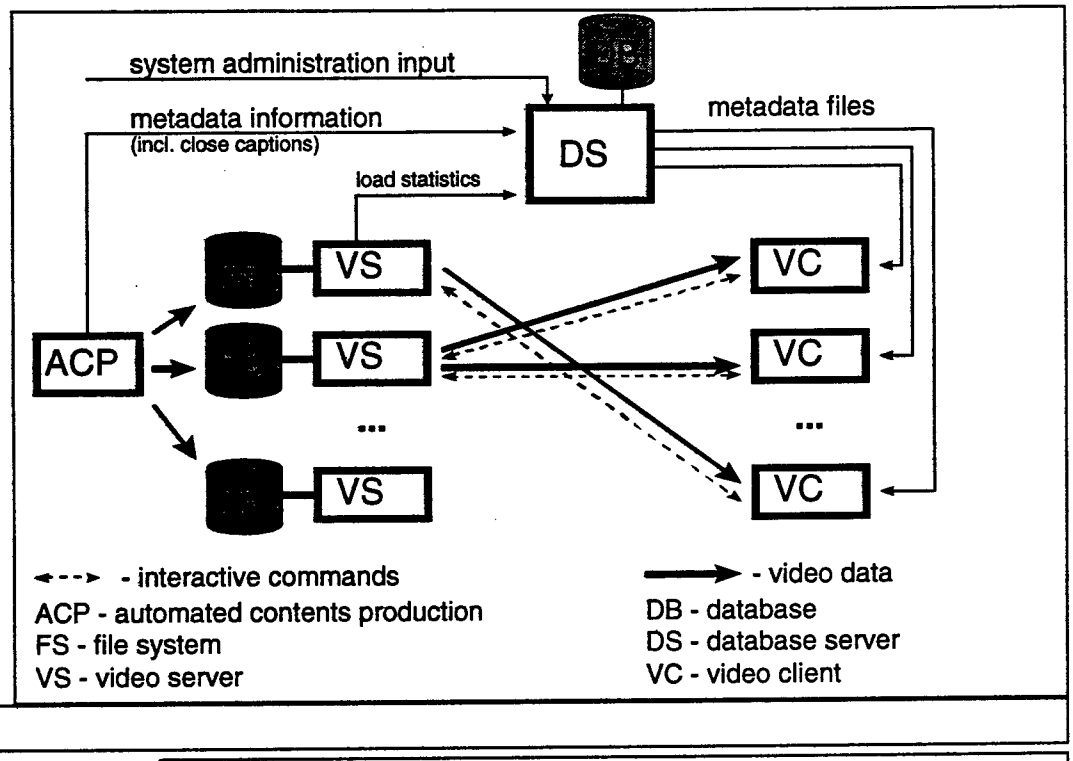


Figure 3.1. Basic components of the video delivery system.

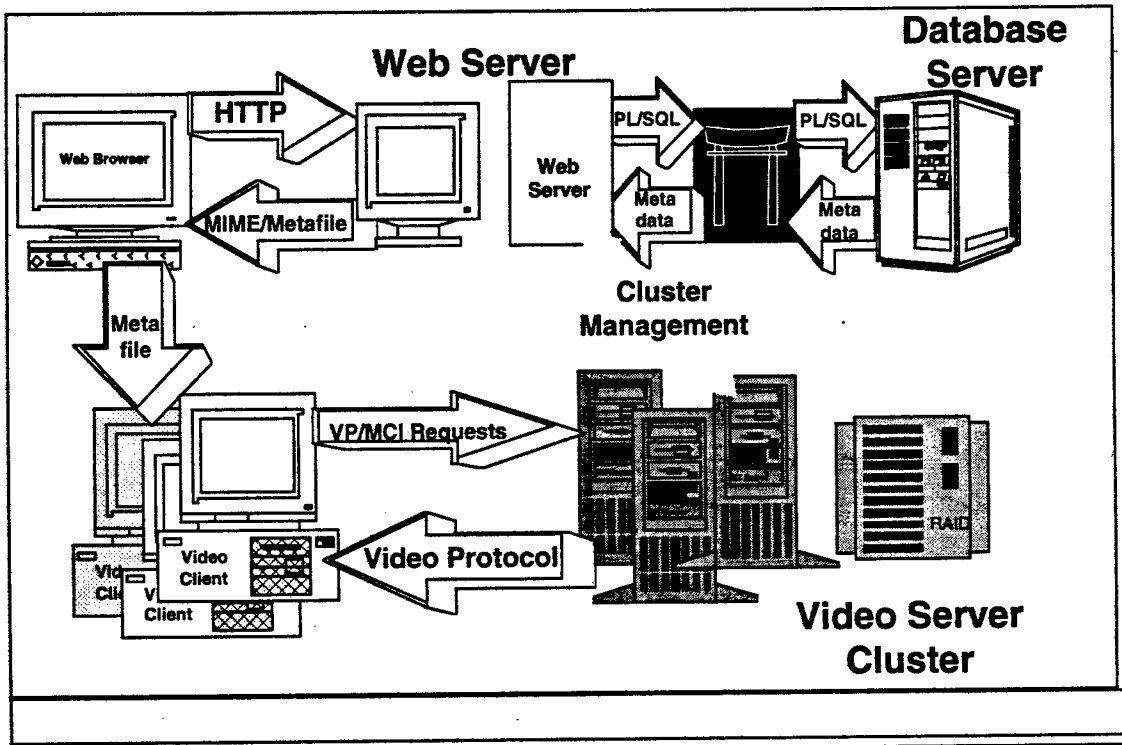


Figure 3.2. Data flow in the video on demand system

More detailed information about system search capability is presented in later sections describing database back-end and video client architecture.

3.2 Metadata concept and database back-end

Any data item characterizing a video clip or otherwise related to a video clip is considered metadata.

Clip metadata can be broadly classified in two categories: clip descriptive attributes and associated clip contents data. Clip descriptive attributes fall into the following subcategories:

1) Technical clip parameters: encoding type, frame rate, bit rate, format, frame–to–file offset translation table, file size, VBR pattern.

2) Clip contents attributes: source, author, producer, copyright, creation date, credits.

3) Clip statistics: availability on the server cluster, location on servers, clip usage patterns, number of copies, insert date, expiration date, access restrictions, access scope, downloadability, charge per one viewing session.

The main use of the clip descriptive attributes is for system administration and asset management. Some of the attributes of type 1 and 2 above may be accessible from the user search interface, especially the clip contents attributes. Data of the type 3 above is used for setting up playback sessions between video clients and video servers.

Associated clip contents data may include: title, annotations (either textual or in a general for of a URL), and closed captions with associated timestamps. This data is critical for video archive searchability. Our video system supports fully indexed, random access to the video material via timestamped closed captioning or annotations.

We are aware of extensive work aiming at other methods of video indexing, including attempts at automatic patter recognition or, at least, recognition of video contents via analysis of signal and color dynamics. Most of these methods are nonviable today. We believe that, within a year or two, continuous speech recognition will provide a powerful video indexing methodology. When this technology becomes available, it can be incorporated easily into our framework as either a replacement or enhancement of indexing via closed-captioning.

As mentioned before, we store video streams in file systems on the video servers. Although certain database vendors (e.g. Informix/Illustra) endorse using databases to store video, we consider this solution to be technically incorrect. Databases lack support for real-time data

delivery. In addition, the current inability to build queries with multimedia predicates for video contents makes the entire idea of placing video in databases a pointless exercise.

A strict division into continuous data and metadata is a fundamental feature of our design. This division cleanly maps the logical modules of the system into separate physical components. It is possible to build a very high performance system by distributing logical modules across multiple workstations. In reality, the system built at NPAC uses a dozen heterogeneous workstations with different networking infrastructures for database access and for video retrieval (see Section 4.1). System requirements for a database server are very different than system requirements for a video server. The modular nature of our design, based on the metadata concept, allows for very efficient resource allocation and for construction of a high performance server from off–the–shelf (COTS) components.

Due to the clear separation of the database and video server functionality, the two phases of the video retrieval process described above (Section 3.1) are not only executed asynchronously but also by physically disjoint sets of components of the VoD system. This solution has additional significant advantages for system administration, performance tuning, and cost-effective system design since it is much easier to tune a group of systems with simple, well-defined workload patterns than a larger system in which each hardware component is running a number of applications with wildly different system resource utilization.

The metadata concept creates one difficulty: video data and metadata may be inconsistent since it is rather difficult to ensure atomicity of transactions in such a heterogeneous database. For example, the operation of adding a clip metadata to the database takes a few seconds, while the operation of copying a GB file from the encoder to a video server can take several minutes, even over a fast network with spare capacities. This problem has not been practically resolved in the current software implementation. It is possible to create a metadata record in the database without having a corresponding video file on the indicated server. Our most recent server design, being implemented outside of this project, will support a protocol extension specifically designed to ensure data consistency.

3.2.1 Video server asset management

Video Server Asset Management, or VSAM, is a separate, self-contained module designed to manipulate the metadata database. The system operates in two modes: manual and automated. The manual mode provides a web-based GUI to all tables in the database. The GUI implements a score of operations on the database, including creation of video server records, logical archive records, and video clips records, as well as later modifications, update, or removal of data. Every single attribute value in the database is editable using VSAM. The automated mode is operated via a special daemon that supports an automated process of video contents creation. This service is described in more detail in Sec. 3.2.4. The service automatically runs the entire process of

video digitization, encoding, placement on the server, and metadata creation in the database back-end. In addition, the latest version of the video server supports a protocol extension allowing statistics about the use of interactive video clips to be uploaded automatically to the database.

Appendix 4 contains a number of screendumps illustrating the most relevant functionality of VSAM.

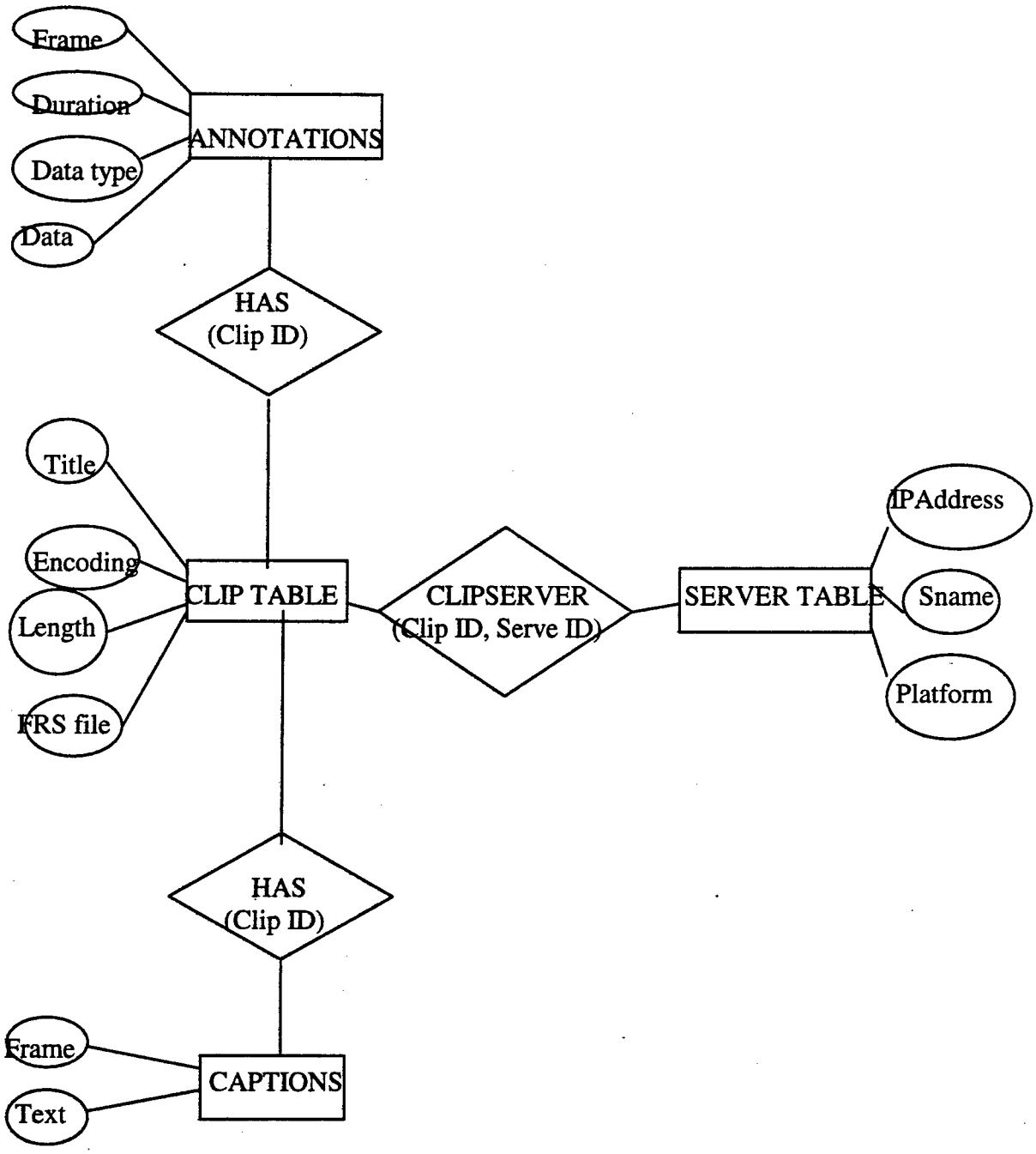
The relational database back-end supports two basic functionalities: video server asset management and indexed video retrieval. Both functions are supported by the same relational scheme holding all system metadata.
While using the same database, data retrieval and data management modules have separate user interfaces. Data search and retrieval interface is designed for the system end-user. The asset management interface is intended to be used by systems administrators and contents creators. Access to this interface is normally protected by a password. Both interfaces are built using Web technology. More specifically, we used following technology components:

- Oracle relational database server.
- CGI technology for http server - database server interface. Perl 4 was used to implement the CGI scripts. The Oraperl Perl extension was used to facilitate Oracle database access.
- GUI forms were built using a combination of HTML3, JavaScript, and, to a small extent, Java.

The following subsections contain description of the relational scheme that is used to store video server metadata

3.2.1.1 Entity Relationship Diagram

A simplified E.R. diagram for the video server database back-end is presented in Fig. 3.3



• Figure 3.3. Entity Relationship for the video server database back-end.

3.2.1.2 Relational scheme: Tables and description

ANNOTATIONS: This table stores annotations for the video clips.

Columns	Type	Description
CLIPID	NUMBER(10)	Video clip ID number
FRAME	NUMBER(10)	Frame number
DURATION	NUMBER(10)	Duration in frames
DATATYPE	VARCHAR2(20)	Data type of the annotation
DATA	LONG RAW	Annotation data

CAPTIONS: Video clip captions are stored in the CAPTIONS table.

Columns	Type	Description
CLIPID	NUMBER(10)	Video clip ID number
FRAME	NUMBER(10)	Frame Number
TEXT	VARCHAR2(2000)	Caption text

CLIPSERVER: This tables lists all video clips stored on a particular video server.

Columns	Type	Description
CLIPID	NUMBER(10)	Video clip ID number
SERVERID	NUMBER(10)	Server ID number
FILENAME	VARCHAR2(255)	Video clip file name in a server

CLIPTABLE : Metadata for each clip is stored in this table. FRSFILE contains contents of the FRS file for a clip in text format (i.e., a frame - file offset lookup table).

Columns	Type	Description
CLIPID	NUMBER(10)	Video clip ID number
TITLE	VARCHAR2(255)	Video clip title
ENCODING	VARCHAR2(10)	Encoding type (MPEG1,MPEG2, AVI)
LENGTH	NUMBER(10)	Length of the clip in frames
FRSFILE	LONG	Frame and File offset table(text)
FILESIZE	NUMBER(15)	Video file size in Bytes
BITRATE	NUMBER(10)	Video clip bitrate in kpbs
FRAMERATE	NUMBER(10)	Video clip frame rate in fps
ARCHIVEID	NUMBER(10)	Archive tag
ACCESSLEVEL	VARCHAR2(10)	Clip access restriction tag
DOWNLD	VARCHAR2(10)	Clip download permit tag
FNAIL	VARCHAR2(60)	Clip thumbnail URL
DATEINSERT	DATE	Date of insertion

SERVERTABLE: stores server metadata.

Columns	Type	Description
SERVERID	NUMBER(10)	Server ID number
IPADDRESS	VARCHAR2(15)	Server IP address
SNAME	VARCHAR2(255)	Server Name
PLATFORM	VARCHAR2(32)	Machine model or OS name

ARCHIVETABLE: stores archive (logical clip collection) definition.

Columns	Type	Description
ARCHIVEID	NUMBER(10)	Archive ID number
ANAME	VARCHAR2(15)	Archive Name
ADESC	VARCHAR2(60)	Archive description

SERVERKEY : To get unique ID for a server automatically, SERVERKEY table is used. This table stores KEYNUM and increases its whenever a new ID is requested. New server ID is the increased KEYNUM. SKEY is just for locking this table.

Columns	Type	Description
SKEY	NUMBER NOT NULL	Key for locking this table
KEYNUM	NUMBER NOT NULL	Server ID to be given
UNIQUE (SKEY)		

CLIPKEY: Similar to the SERVERKEY table, used to generate unique CLIPID.

Columns	Type	Description
CKEY	NUMBER NOT NULL	Key for locking this table
KEYNUM	NUMBER NOT NULL	Server ID to be given
UNIQUE (CKEY)		

ARCHIVEKEY: Similar to the SERVERKEY table, used to generate unique ARCHIVEID.

Columns	Type	Description
AKEY	NUMBER NOT NULL	Key for locking this table
KEYNUM	NUMBER NOT NULL	Server ID to be given
UNIQUE (AKEY)		

3.2.1.3 ID generating for servers, clips, and archives

Server and clip IDs should be unique to maintain consistency. IDs are assigned automatically rather than manually since the information is used only within the database back-end. The simplest way is to start with a number and to increase it when a new ID is needed. The only

problem is the race condition. To solve this problem, we used the database as a locking method. The tables SERVERKEY, CLIPKEY, and ARCHIVEKEY are used for this purpose. The pseudo-code for assigning a new ID is:

```
PROCEDURE GET_ID
BEGIN
  READ ID;
  if  READ_ERROR then EXIT with ERROR ;  # since another
process is using it.
  DELETE ID;
  if DELETE_ERROR then EXIT with ERROR;
  ID := ID + 1;
  WRITE ID;
END;
```

3.2.2 Indexing and search interface

The end user can perform two types of video material searches: the category-based search and the content-based search. In a category–based search, a property associated with a video clip is requested. Implementation of this capability is elementary. The number of different attributes that can be searched depends only on the extent of metadata associated with each video clip. Our system holds a handful of such attributes. This number can be extended on the fly by adding attributes to the database tables. In the current version of our system, video clips are grouped into video archives that can be searched separately. The user can also search for clips with certain technical characteristics, for instance, a user connected via a low bitrate line may be interested only in low bitrate video clips.

The content-based searches supported by the system come in two flavors. A keyword search can specify database fields to be searched. The metadata associated with each clip may be time-stamped. Closed-caption information is a type of time-stamped data, while clip titles and clip annotations (which usually contain a "script" of the story presented in the video clip) are not. Our system allows for both kind of queries. A query acting on non–time-stamped data results in a list of clips satisfying the query, as in a category–based search. Queries against time-stamped data always yield, in addition to the clip list, a list of offsets associated with positions, relative to the clip beginning, of the search keywords in the time-stamped data. In this way, our system not only searches for video clips but also for the relevant video–stream entry points.

We believe this to be a very practical approach to video indexing. The issues of capturing and converting closed–caption date are discussed in Section 3.2.4. As indicated above, we believe that continuous speech recognition technology will mature sufficiently in the next few years to make it useful for indexing. This technology, together with the approach we developed, will allow for a rapid increase of the amount of video material available in searchable digital video

repositories. It should be obvious that, with minimal effort, the results of a speech recognition process may replace current closed–caption support in our system.

It should be mentioned here that the file offsets produced by the database search engine *per se* do not warrant random access to the video streams. It is still necessary to design both the video client and the video server so that the random access requests are seamlessly handled. This is particularly difficult in our case since we requested that the video server remain codec-independent. Therefore, it is up to the video client to process the video stream in a way that ensures correct playback. The methodology to do so differs from codec to codec; this matter is discussed further in Section 5: Video Client Architectures.

The process of creating metadata, storing it in a database, and searching through the database by sending queries, is entirely self-contained. The only requirement for compatibility with our video client-server architecture is a particular form of the metadata packet returned by the database search engine. The details of the data flow between the database back-end and video client-server module is discussed in the following subsection.

3.2.3 Data flow model

The process of selecting and playing a video clip involves a data movement pattern that is quite complex. The entities and data streams involved in the process are illustrated in Fig. 3.2. Figures 3.4 - 3.6 illustrate the functionality of the search interface.

The process starts from a Web browser accessing the video server search interface. The HTML form presented to the user provides a collection of buttons and type-in fields needed to compose a query. Upon submission, the form output is sent to the Oracle RDBMS via a CGI script residing on the HTTP server. The query is executed by Oracle and the results are returned to the browser. The user is presented with a dynamically created HTML page with a list of available video clips. If the search keyword has been spotted several times in the closed-caption data associated with the clip, the user will be given a list of stream entry points. In addition, if the clip resides on multiple servers, the user will be able to chose a video server to deliver the clip (Figure 3.5).[1] In addition, the page contains links to other content elements possibly associated with the clip, for example, annotations. Annotations *per se* can contain a list of other relevant URLs, which can be examined while the video clip is playing.

[1] It is planned that the list of video servers will be ordered accordingly to their load, providing the user with the opportunity to select the server with lowest load. This selection could be automated; but if the user operates behind a firewall, there is a benefit of being able to chose manually.

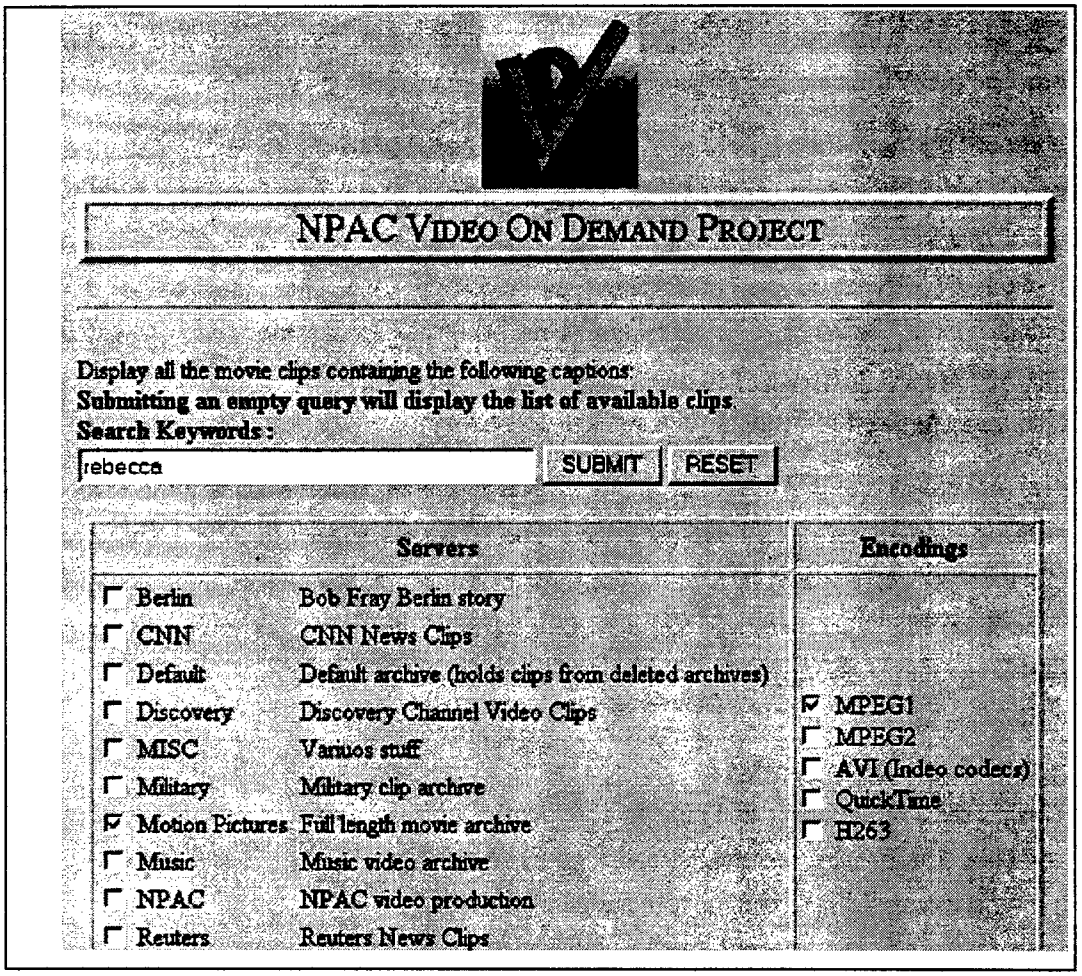


Figure 3.4. Front page of the video server user search interface

After selecting the clip access point, the user presses the PLAY button. Again, output from the form is sent to the database back-end via a set of CGI scripts. The output from the database is a multi-part HTTP document. The first part of this document contains a metafile used by the Web browser to start the video client. Our basic video client is a helper application, although we implemented browser plug-ins and, even, a Java applet (see Section 5.1.4). The type of the helper application started by the browser is determined by the usual MIME mechanism. The contents of the first segment of the multipart document is now passed as an argument to the video client. This content enables the video client to actually establish a link to the video server, an IP address specified in the metafile. The metafile specifies also the clip location(s) on the server. In addition, for certain type of contents, the metafile contains a lookup table providing translation between the frame numbers and the actual file offset on the server. This information supports the codec-independent, random access capability of our VoD architecture.

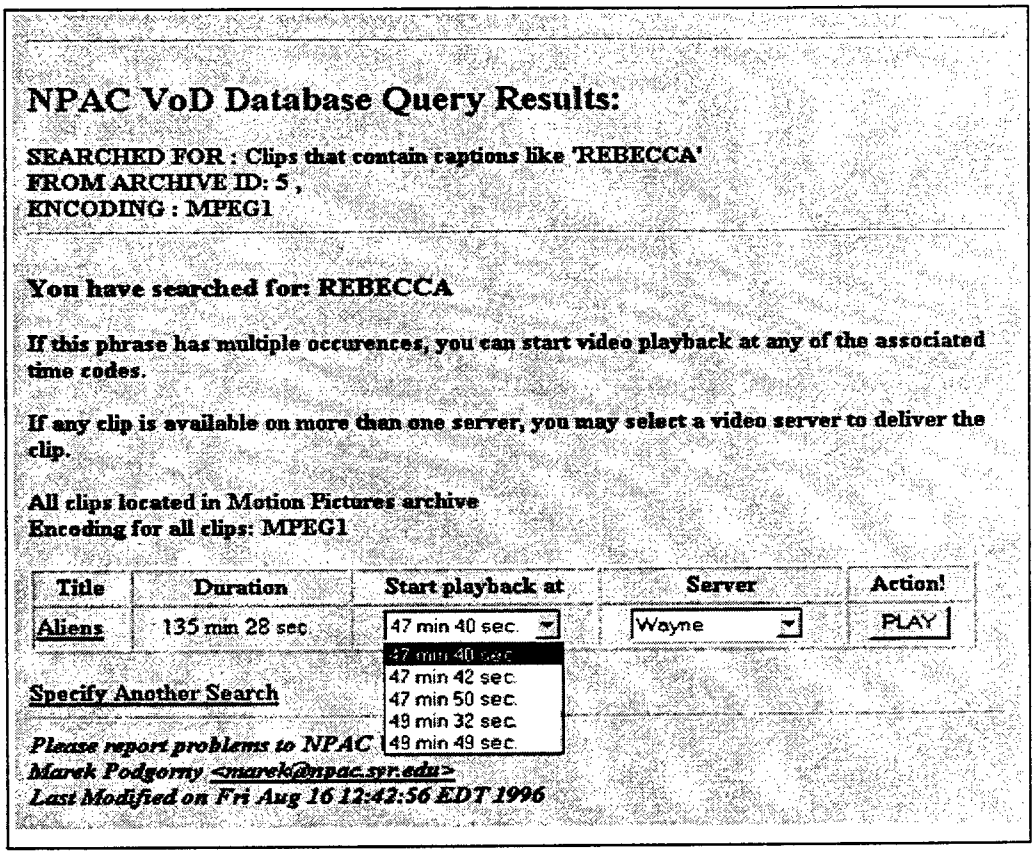


Figure 3.5. Hit list of video clips returned by the database back-end.

After the video client establishes a link with the server, video playback continues independently on the Web browser. This flexibility is one of the reasons we prefer and recommend the helper application model over the plug-in model for video clients. Most of the clients we implemented support fully interactive playback, i.e., they implement VCR-like controls with PAUSE, RESUME, and SEEK buttons. Seek functionality is supported using the offset tables mentioned above.[2]

The second section of the multipart document contains contents of the time-stamped document associated with the selected video clip. This is illustrated in Fig. 3.6. The time stamping information is preserved and can be used as an alternative method for random access. The user might want to skip to a particular part of the video clip, based on the textual contents presented to him by the search engine. He/she does it by clicking the link symbol next to the phrase (see Fig. 3.6). In response, the search engine returns an abbreviated metafile which is passed to the

[2] For some implementations of decoders (for instance, the ActiveMovie MPEG decoder), random access is supported even without the frame-file offset translation table. See Sec. 5.2.10 for details.

existing instance of the video client. The client stops the current playback, searches to the requested stream entry point, and resumes playback from there.

Figure 3.6. Video client and random indexed video stream access.

The above description of the data flow pattern in our video server does not cover details of the protocol used by our video clients and video servers to communicate. This subject is covered later in the report.

3.2.4 Contents creation service

Implementation of a contents creation service for the video server was one of the explicit project requirements. The service we created has the following features:

1) Analog video sources supported by the system include VHS and S-VHS tapes, Betacam tapes, and a satellite link.

2) The basic encoding used by the system is MPEG1. We experimented with Motion JPEG and proprietary SGI formats; but this work was discontinued as the position of MPEG1 stabilized as a video encoding standard. The MPEG1 encoder we installed is a real-time hardware solution. In addition, we have the capability for real-time encoding to the Indeo 4 format. But although Indeo 4 is a quite popular Intel video codec, it is not a standard.

3) For low bitrate video streams, we implemented MPEG1-to-H.263 conversion software. Similar to MPEG1, H.263 is an international standard. We implemented most of the H.263 codec software and the entire conversion utility.

4) The content creation system extracts closed-captions from the encoded video stream concurrently with the encoding process.

5) The entire process of contents creation has been automated. The system is completely distributed and, once started, performs the entire procedure of video stream and metadata insertion into the system repositories.

Contents production is time consuming and rather tedious, consisting of tasks such as digitization and encoding of video data, metadata creation based on video content, or copying video files to remote file systems on Video Servers. The automation speeds up this lengthy process as well as maintaining consistency between VoD file servers and a metadata database. The automation begins from the capturing of the video material from live source: satellite or TV feed, video camera or VCR. Automated contents production updates the database and a Video Server file system each time a new video clip is digitized. Also, this process will digitize, compress and copy a video file to a Video Server, as well as parse the video file to obtain additional metadata information, allowing content-based searches. All metadata created during the process is inserted into the database.

The modular structure of the contents production service is depicted in Fig. 3.7. The service consists of the following modules:

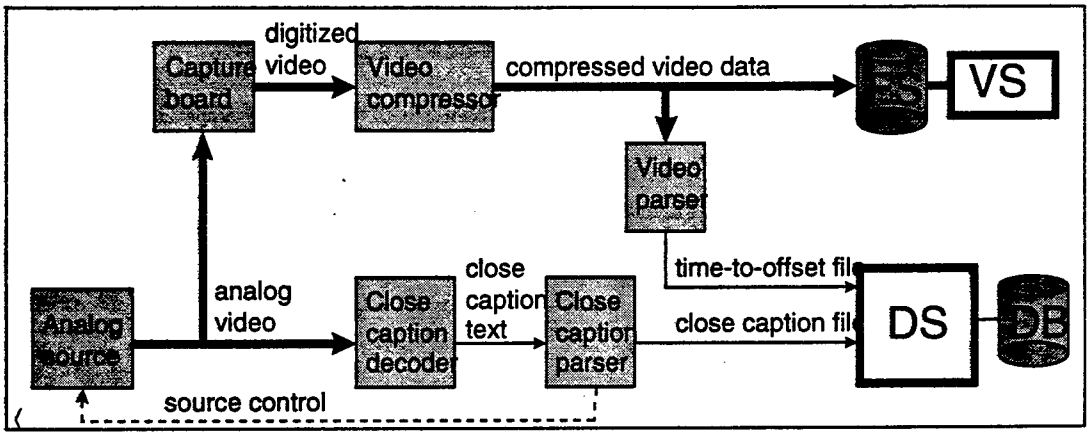


Figure 3.7. Block diagram of the automated contents creation service

- **Analog video sources:** There are two video tape recorders (VTR) and one satellite receiver in the system. Both VTRs are computer controlled, i.e., they connect to a PC or UNIX workstation via a serial port. All aspects of the VTRs operation can be controlled from the encoding workstation via special drivers. The most basic remote control functionality includes tape movement control and tape-counter control/retrieval. Computer control of VTRs is standardized, and the interface is known as SONY 9 pin. The drivers for most high-end VTRs can be purchased from a handful of companies. The driver we installed, SoftVTR from Moonlight Enterprises, is available for both PC and UNIX platforms. The two VTRs installed in the system are respectively JVC model BR-S822UJ for VHS/S-VHS tape standard and SONY Betacam model BR-S822UJ for Betacam tapes. Both VTRs are studio quality and extremely reliable. They support all the functionality needed to provide clean analog video signal for the encoders, including tape pre-roll.

  Satellite input comes from a satellite dish installed at the Newhouse telecommunication facility at Syracuse University. The antenna receives Galaxy V satellite signals. The multi-transponder signal is connected to a Drake satellite receiver model ESR 1252 installed in the Newhouse facility. Output of the receiver is converted from coax to multimode fiber and back by a pair of the Drake converters model 1252. A fiber link transports the video signal from Newhouse to NPAC. The receiver signal carries only programming from one transponder. To have the ability to switch between transponders, we installed a

modem connection between the encoding workstation and the Drake satellite receiver. A simple software driver enables us to change transponders to gain access to multiple programming channels available on Galaxy V. Galaxy V carries, among other channels, Reuters News Service for broadcasters; thus, the satellite link gives us access to unedited news material.

- **MPEG1 video encoder:** The process of encoding MPEG video is so CPU intensive that its software implementations are possible only in batch, non real-time regime. Hardware support is necessary for real-time encoding. We tested and evaluated a number of products before making a purchase decision. One of the most important factors in our decision was the issue of compatibility. Nominally a standard, MPEG1 was sufficiently immature in the early periods of this project to cause interoperability concerns. Consequently, we linked our decision regarding an MPEG encoder with availability of inexpensive decoders from the same vendor. Further, as described in Section 5 on the implementation of a hardware-supported video client, we requested that the decoder vendor make available to us an SDK so that we could proceed with the development of the networked video client. The only company responsive to our requests was Optibase, an Israeli company with a US base in Dallas, Texas. Since the Optibase encoder is one of the most highly regarded on the encoder market, we decided to acquire it.

  The Optibase MPEG LabSuite encoder consists of three ISA cards: the video encoder based on the C-Cube MPEG1 encoding chip, a card combining an MPEG1 audio encoder with an MPEG1 audio and video decoder (PCMotion), and a card for the capture of digital audio (D-5000). The hardware is driven by Optibase integrated encoding software. Written entirely as an application (without drivers), version 3 of the MPEG LabSuite software runs under both Windows 3.11 and Windows 95. Our setup still runs under 16-bit Windows, since this setup proven to be stable and entirely sufficient for our needs. Optibase MPEG LabSuite cooperates with SoftVTR MCI driver for VTR control. The encoder, installed in a COTS Gateway 100 MHz Pentium with SCSI disk drives, supports real-time MPEG1 encoding with stream bit rates up to 2 Mbps. Such a bitrate corresponds to VHS quality and is sufficiently good for viewing and as a source for low bitrate re-encoding. For higher bit rates, we observed instabilities on the encoder operation. MPEG LabSuite software supports a batch encoding mode, i.e., it is possible to select multiple fragments of the tape for encoding and start an automated procedure.

The software environment on the encoder includes two other important elements: the Netmanage NFS server enables us to export the encoder disks so that the contents can be transferred easily to video servers, and the Symantec ScriptMaker utility. The role of ScriptMaker will be described later in this section.

- **Closed-caption decoder:** Captioning is an electronic (or manual) process which converts the audio portion of a television program into written words. Unlike subtitles, captions are designed specifically for the deaf and hearing-impaired; thus, in addition to on-screen dialogue, significant sound effects and off-screen nuances might be described.

  The NTSC broadcast signal consists of 525 lines at 30 frames per second. Individual images forming the motion picture are divided by a 21-line-wide bar called the Vertical Blanking Interval (VBI). VBI can carry additional information, usually used for the synchronization purposes. But Electrical Industries Associate (an American standards body) has defined a standard for the transmission of closed-caption characters in the 21st line of the Vertical Blanking Interval. This service is also known as the "Line 21 service." This standard allows for two characters to be stored for each video frame, for up to 60 characters per second.

  There are 4 closed-caption channels, 4 text channels and one Extended Data Services channel (XDS). Closed-caption channels as well as text channels may carry characters in 4 different channels, each in a different language. Extended Data Services have been designed to carry various kinds of information such as weather conditions, general information about the current television program (summaries, time to end, copyright, author, etc.). However, only one closed-caption channel is currently used by broadcasters.

  Closed-captioned video essentially carries the entire audio layer in textual form. The contents creation service uses an industrial closed-caption decoder from EEG Enterprises, model DE 241DR to extract captions from the video signal and translate them into ASCII code. Via a serial port, the captions are transferred to the encoder machine. We implemented a software module to record the captions, parse them into sentence-size chunks, and time-stamp them. The resulting caption file is inserted later into the back-end database as a part of the metadata. The process of video encoding and caption decoding is concurrent, as explained in the next subsection.

- **Automation of the contents creation procedure:** The elements described above can function as separate entities. In our initial implementation, the video encoding and closed-caption decoding processes were run sequentially. Since

each of these processes is real time, we found this solution to be quite inefficient. Further, after the video has been encoded and caption extracted, it was still necessary to manually transfer video files to the video server and to construct metadata records in the database.[3] This process was intolerably tedious and error prone even in a research prototype. Thus, we designed an automated system. This task was not entirely trivial since we were dealing with an extremely heterogeneous system and with monolithic applications without any communication capability. The solution had two parts: a ScriptMaker driver which controls both the MPEG LabSuite and caption parser on the encoding machine, and a UNIX-style daemon continuously running a watchdog service looking for new digitized video files.

Symantec's ScriptMaker is a rather obscure scripting language that has the ability to mimic Windows desktop controls typically operated by mouse and keyboard. Any kind of API would be preferable to this solution; but in the absence of better tools, ScriptMaker provides a methodology to start and control monolithic Windows applications. We used it to operate concurrently the MPEG LabSuite encoder and our own closed-caption parser. This solution, although revoltingly inelegant, gives us the benefit of a 50% decrease in content preparation time. Technically, the ScriptMaker interface is a simple form with fields corresponding to typical input to the encoder and closed-caption parser. The Go button starts the entire process of encoding, during which the original applications are invoked, parameters set up, and the encoding process started. Upon completion of the encoding process for each clip, the ScriptMaker utility writes a file with partial metadata to the encoder machine's storage. Appearance of this file signals the watchdog service to start processing the newly created files. The daemon runs the following steps:

1) The MPEG file is run through an indexing utility. This process constructs the frame-to-file offset lookup table. The table then becomes a part of the metadata being prepared for the database upload.

2) The video file is copied to the designated location in the video server file system.

3) The metadata connects to the suite of CGI scripts normally used to run the VSAM system (see Sec. 3.2.1) and inserts the metadata into the database back-end.

[3] We attempted video encoding directly to a network disk-drive located on the video server. But this did not work well.

Apparently, the MPEG encoder is extremely sensitive to any delays in moving its data over the ISA bus, and the CPU overhead involved with network transport was sufficient to destabilize it.

4. Video Server Implementations

Originally, we planned to investigate two different hardware architectures as video server platforms. Our initial choices were nCUBE2, an MPP (Massively Parallel Processor) with hypercube interconnect, and the SMP (Symmetric Multiprocessor) SGI server. Our first video server implementation was written for the SGI platform. Also, we expanded the configuration of our nCUBE machine, investing significant amounts of time and human resources in an nCUBE prototype video server. As described in detail in Section 4.2 below, we found the nCUBE2 system software unsuitable for supporting a video server; thus, the server implemented for this platform did not reach application phase. We have since turned to the idea of a video server built from inexpensive COTS components, clustered for performance. This led us to the implementation of the video server for Windows NT platforms. Our production video server in NPAC consists of a cluster of very high multiprocessor Windows NT servers based on the Pentium Pro CPU.

The following section is structured as follows: **Section 4.1** describes the early design of the video server for single CPU and SMP machines. This architecture was implemented on both SGI and NT SMP platforms. **Section 4.2** describes architectural changes made to the Windows NT-based servers to improve their performance and scaleability. This section also describes the architecture and configuration of the current NPAC production video server. **Section 4.3** summarizes the design principles of our distributed video server architecture. The software for this architecture has been implemented only in part during the project and is currently being used as a research prototype at NPAC. Finally, **Section 4.4** provides a detailed account of the effort directed towards implementation of the nCUBE-based video server and explains the reasons for the termination of the project.

4.1 Early server designs

4.1.1 Network Dataflow in a VoD System

Conventionally, dataflow in a VoD system moves from the video server to the client.[4] The client requests data from the server, which results in data being streamed from the server to the client. There are two regimes of data being streamed from the server to the client: the pull regime and the push regime.

- In the pull regime of dataflow, the VoD client determines how much data is needed and sends a request for that amount to the VoD server. The VoD server responds with the amount of data requested. The client then receives this data in

[4] A generalized video server of the future will have capability to "capture" live video streams, in which case the traffic patterns for a video server will be more symmetrical. This capability will be added in the framework supported by the RTSP (Real Time Streaming Protocol)

a variable amount of time and requests more data from the server. This cycle continues until the whole movie is played out. The data, thus, is being pulled on demand by the client.

- The push regime is orthogonal to the pull model. Here, the client does not request any data from the server; rather, the data is furnished by the server on a continuous basis. This form of data flow is especially suitable for Constant Bit Rate (CBR) coded movies. By this we mean that a certain amount of data is always consumed within a certain time period. An example of this encoding is the MPEG1 video standard. This model can be applied also to a Variable Bit Rate (VBR) encoded stream since the server can transform the VBR stream into a CBR stream and transmit the data at this rate.

There are fundamental differences between the push and the pull model. In the pull model, data transfer is initiated at request of the receiver. Hence this can be classed as a receiver-initiated mode of data transfer. The push model of data transfer, on the other hand, is sender initiated. Here, the sender (e.g. a VoD server) determines when and how much data is to be sent.

The pull model contains a significant performance penalty since two communication transfers are needed to transfer a unit of data: a request for the data and the actual data transfer. In a VoD system, we are dealing with isochronous data, i.e. data which has strict limits on when it needs to be delivered and which must preserve the timing relation contained within the data. Thus the request-response structure of the pull model introduces an undesirable delay.

The push model, on the other hand, exploits the properties of isochronous data (i.e., data transmitted by the server at a given rate, on the assumption that it is being consumed at the same rate). This assumption holds for CBR encoded streams and needs to be extended to VBR encoded data. By using the isochronous properties of the data, the push model eliminates the traditional request-response model of data flow, thus eliminating delay and preserving network bandwidth.

As reported in detail in Appendix 1 , we developed a methodology to efficiently transport VBR data over CBR channels. Thus, we believe that the push regime can accommodate both CBR and VBR traffic. In addition, we observe that the push regime is consistent with the two fundamental components of the Integrated Services Model for Internet: multicast and Quality of Service (QoS) support via the Reservation Protocol. For multicast, the push regime is obviously the correct solution. For the Reservation Protocol, even though the client issues reservation requests, the traffic descriptors must be provided by the sender. This is compatible only with the push regime of data transport.

It is important to note that the push regime does not preclude interactivity and random access capability. It is still possible to pause and randomly access the video stream. It is possible also to extend this capability to multicasted sessions, which is described in Section 5.5.

Consequently, all video servers designed at NPAC use the data-push regime for video data transport.

4.1.2 Video Server Configurations

Video servers are basically large repositories for data and information which are provided on demand to clients. The VoD server architecture can be defined by the way the data is stored within the video server. Data can be stored locally within a server or can be distributed across a large number of servers. Thus, two distinct methodologies exist for video server configuration:

- ***Autonomous Servers:*** Video servers of this type can be viewed as stand-alone entities since they store encoded streams locally to themselves and. thus, do not require the cooperation of other servers to serve a VoD client. The actual physical implementation and the logical implementation of the video server are relatively the same. The server usually consists of a CPU, a storage medium (which can consist of a standard storage hierarchy), and a network interface.
- ***Distributed Servers:*** Instead of storing entire video streams locally, encoded streams may be spread out among a larger number of video servers which are usually distributed over a local (but can be a wider) area network. Thus, in order to serve a VoD client, a set of servers need to cooperate. The client may need to be aware of the topology of the VoD servers. The encoded bit stream can be stored in any granularity throughout the system.

Autonomous servers suffer from a well-known problem associated with centralized processing, i.e., the common point of failure. In this case, when an autonomous server fails, all the clients being served by it are affected. In the case of the distributed paradigm, if we assume that some loss of data can be tolerated at the cost of reduced Quality of Service (QoS) to the clients then when a server fails all the clients being served by the distributed system are affected as they all have a part of the encoded stream stored on the faulty server.

One of the driving forces behind the distributed server paradigm is the concept borrowed from Redundant Arrays of Inexpensive Disks (RAID). In order to serve a large number of clients from a VoD server, the server needs to have a large data bandwidth from the storage system to feed the demand from the VoD clients. In the field of scientific computing this large demand for bandwidth was provided by the RAID paradigm. For a VoD server, however, it does not appear that providing a large data bandwidth will necessarily lead to a larger number of clients being serviced. This is due to the fact that RAID was designed for a small number of large bulk data transfers which

achieved the higher bandwidth. A VoD system, however, must service a large number of relatively low bandwidth requests. In the case of MPEG1 encoded video, this is typically about 1.5 Mbits/sec. Thus we might achieve better results by replicating autonomous servers to provide service[5].

The distributed system suffers also from the overhead associated with coordinating resources which are distributed across to service a single request. This overhead could be in terms of network bandwidth (by the use of an extra messaging protocol) or CPU usage. Alternatively, if the client needs to be aware of the server topology, the design may be harder to port to other network topologies. Thus, the best approach appears to be the autonomous server paradigm combined with the optimizing of each server to provide the best service it can by itself. These servers can then be distributed around a wider area network which can then provide service to VoD clients more efficiently. The VoD client can find the closest server storing a specific stream and fetch it on demand from that server.

The philosophy described above has driven our development efforts. Few generations of autonomous servers have been implemented. Next, we moved towards the model of the clustered autonomous server.

The server model for nCUBE2 was somewhat different. The basic difference between the nCUBE and SMP platforms is that nCUBE can be seen a "shared disk" architecture. Such an architecture potentially permits much more flexibility for a video server. We discuss this issue more thoroughly in Section 4.4.

4.1.3 Software Architecture of a VoD Server

The video server must be able to service multiple client requests simultaneously. This might lead us to investigate the potential for exploiting parallelism at the server side, which is discussed further in later sections. A video server should be capable of pumping streams requested by clients.. It should provide also basic interactivity, i.e., support operations such as stop, rewind, play a video, and, possibly, provide random access to the video stream with some sort of fast-forward preview. Thus, the video server should provide two basic functionalities:

- A data pumping module
- An interface to enable interactivity with the stream.

An interface is defined to enable the client to interact with the stream in a consistent manner. A consistent interface leads to cross platform development, i.e., clients can be developed on

[5] Recent research in parallel I/O systems indicates that without complete re-engineering of the underlying file system structure, parallelization of the I/O operations more often than not does not result in any performance benefits. We believe that the issue of file system support for video is an open research issue.

multiple platforms and communicate in a predictable way with servers built on multiple platforms. This leads to transparent execution between cross-party platforms.

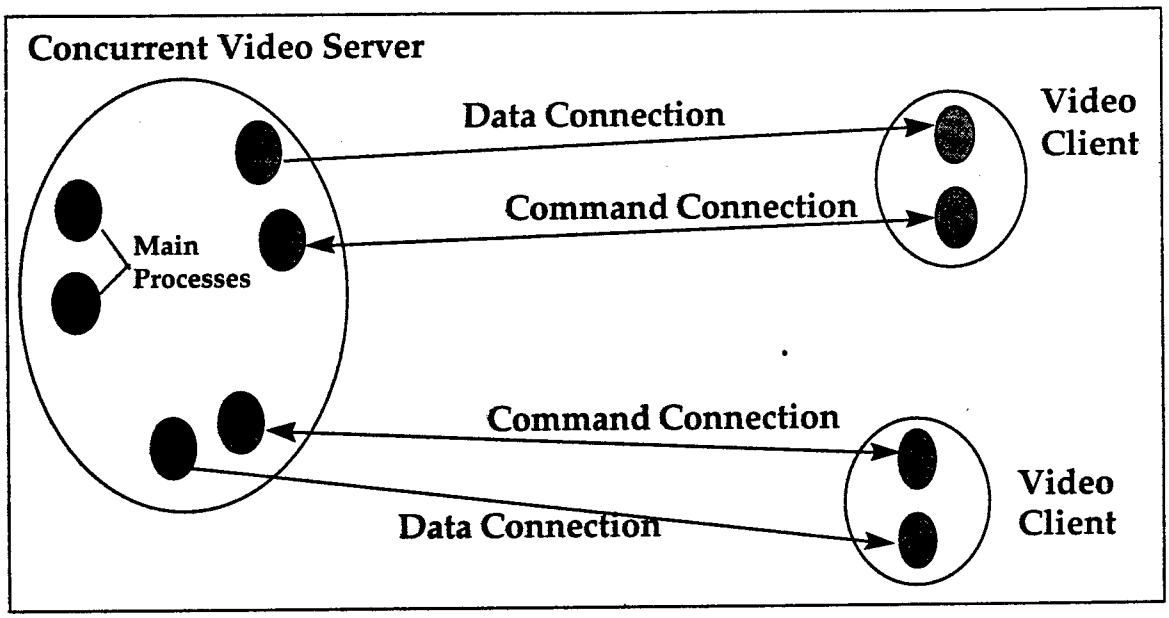


Figure 4.1. Early software architecture of the video server

The architecture of our early video server design is shown in Figure 4.1. The server contains two distinct logical-- the data and the command ports. The data link is logically a unidirectional link with data flowing from the server to the client. The command link is a bi-directional and is used to implement the application level interface between the server and the client. The data link was chosen to be unidirectional for efficiency as no parsing of packets need to be performed at the client end.

4.1.4 Optimizing the VoD Server

Once the architecture of the video server had been decided, we needed to optimize the VoD server. Each video server must consist of three basic elements:

- A Central Processing Subsystem to handle requests from clients and other management functions.
- A Storage System to store the streams. The storage system can be flat or can be a hierarchical storage system.
- A Network Interface used to communicate with the clients and provide the basic service.

One approach to optimizing the whole VoD server is to optimize each element locally with the hope of achieving a globally optimal solution. The following sections outline the issues involved in optimizing each of these subsystems.

4.1.4.1 Processing Overhead

It is clear that some processing will be involved at the server end to manage the requests from the VoD clients. The server needs to order disk and network I/O in a fair manner for the clients it is serving. It also needs to allocate fairly and efficiently the limited CPU time it has at its disposal. Here we must establish the fact that, because the computing power available to the server is limited, a point will be achieved after which no further clients can be serviced without degrading the performance of the currently active sessions. If this situation arises before the storage or the network system is loaded, we say that the server is "computer bound." Since the server needs to manage requests concurrently from multiple clients, it is advantageous to develop a multithreaded server to exploit the inherent parallelism present.

Parallelism can be exploited at various levels. A VoD server can exploit parallelism ranging from fine grained to coarse grained. Issues involved with exploiting parallelism are discussed next.

- ***Fine-Grained Parallelism:*** In this model of parallelism, the ratio of computation to communication is quite small. For example, in the context of the VoD server, there might be a single thread for each of the two communication channels between the server and the client, that is, one thread each for the data channel and the command channel. Thus, the overall number of threads executing at the server for N clients is `(2*N)+2`. There are two extra threads enabling the server to accept a new connection. With a large number of clients, the number of threads executing at the server will rise quite sharply.

  One drawback of this approach is the context-switch overhead, which results from the threads being scheduled. This might tend to overload the system with the managing of threads. Thus, the VoD server may become CPU bound while supporting only a small number of clients. It should be noted that the command thread does relatively little work compared to the data pumping thread.

  The advantage of using fine-grained parallelism is the overall reliability of the VoD server. In the case of a runaway client, which might cause either of the data or command threads to crash and die, the VoD server can still provide unhindered service to the other clients.

- ***Medium-Grained Parallelism:*** In this model, the amount of computation is slightly more with respect to communication, etc. In the model of the VoD server, since the command thread performs relatively little work compared to the data

thread, it might be sensible to have only a single thread which manages command requests for all the clients. In addition, we have a multiple data threads, that is, one for each client. Thus, for N clients, there will be N+3 threads executing at the server side.

This is substantially less than the fine-grained model and will lead to a reduction in context switches. This might lower the CPU utilization at the server, lowering the probability of the server system being computer bound. Also, it might allow support for more clients per server, thus raising the efficiency of the system.

However, a drawback associated with this approach is reliability. The VoD server will not be as reliable as the fine-grained model. For example, if a stray client issued a spurious request on the command link resulting in the server command thread crashing, then all the clients would be affected since there is only a single thread to manage all the clients. In the case of data thread crashing, though, the server will still be able to provided an unhindered service to remaining clients.

- ***Coarse-Grained Parallelism:*** In this model, the amount of communication, etc. is relatively small compared to the computation. In the VoD server model, we might have one data and one command thread for all the clients. Hence for N clients, the server will consist of only 4 threads. Context-switch overhead will be virtually eliminated compared to the other two schemes. However, this model has some drawbacks.

  The first drawback is reliability. This provides the least reliable service when compared to other schemes because either of the threads crashing will result in a disruption of service. Another drawback is that the complexity of implementing the server increases. because the server needs to perform multiplexing/demultiplexing of data and command dispatches.

Taking all these issues into account, our early implementation of the video server incorporated the fine-grained model of parallelism. The main reason behind our decision was that reliability of service was deemed an essential factor at the cost of a slight performance penalty. In the later versions of the server, we moved to coarse-grain parallelism[6], taking benefit of certain system-level solutions offered by the Windows NT system.

4.1.4.2 Storage System Issues

The pros and cons of using RAID have been discussed previously in the subsection on a distributed VoD server (4.1.2). As noted, RAID systems are not suitable for VoD systems. The

performance of the storage subsystems is critical to the efficient functioning of the VoD server. The storage system needs to support multiple I/O requests and provide the data back to the server so that it can be pumped over the network to the client.

In order to increase the bandwidth from the I/O subsystem, asynchronous I/O requests are issued. Thus, anticipated demands for data can be queued to disk so that the data is available when it is needed by the clients. This pre-fetching of data before it is needed results in seamless flow of data and the effect of jitter observed by the client. This interleaving of storage and network I/O is dealt with in later section.

Due to the fact that most of the video streams play continuously, pre-fetching is an efficient and effective way of screening disk I/O latency. In the video servers we have implemented, we have never observed our system to become disk I/O bound. This conclusion contrasts with the popular belief that the video server's performance is critically dependent on I/O performance.

4.1.4.3 Network I/O

Network I/O is the last link in the chain for data to be delivered to the client. Traditionally, blocking calls to the network were made when the server had to wait for a request to complete before continuing execution. However, in the case of a VoD system, we are dealing with isochronous data which has certain inherent timing relations built in. This property of isochronous data can be exploited to perform asynchronous network I/O. This should improve further the performance of the network I/O subsystem since the server can queue all the network requests without blocking.

A main component of the network I/O subsystem is the transport protocol. Practically this should be a light-weight, high performance protocol. Data delivery does not have to be guaranteed, but it should be ordered. For the early implementations of the video server at NPAC, we chose the TCP protocol running on standard IP networks. This protocol guarantees error-free data delivery. Although this was needed, it was chosen for simplicity in the prototype system. The TCP protocol is a relatively heavy protocol; and it might be better to use a lighter weight protocol, like UDP.

4.1.4.4 Interleaving Disk and Network I/O

To further improve the throughput of the video server, the network and storage system I/O were interleaved and a dual buffering scheme was used Initially, the first buffer is filled from the disk. Then, in the next time slice, this buffer is written out asynchronously to the network while the next block of data is fetched from the disk asynchronously into the second buffer (pre-fetching of data). In the next time slice, the second buffer of data is written out to the network while the first buffer is filled with the pre-fetched data.

---

[6] More precisely, the production server has an additional independent thread handling disk I/O operations. It is also possible to start a pool of worker threads, to take advantage of multiple processor on an SMP server.

This scheme improves overall throughput of the storage and network subsystem by the pre-fetching of data and asynchronous I/O requests. Thus, the whole VoD server system supports the asynchronous I/O model of execution.

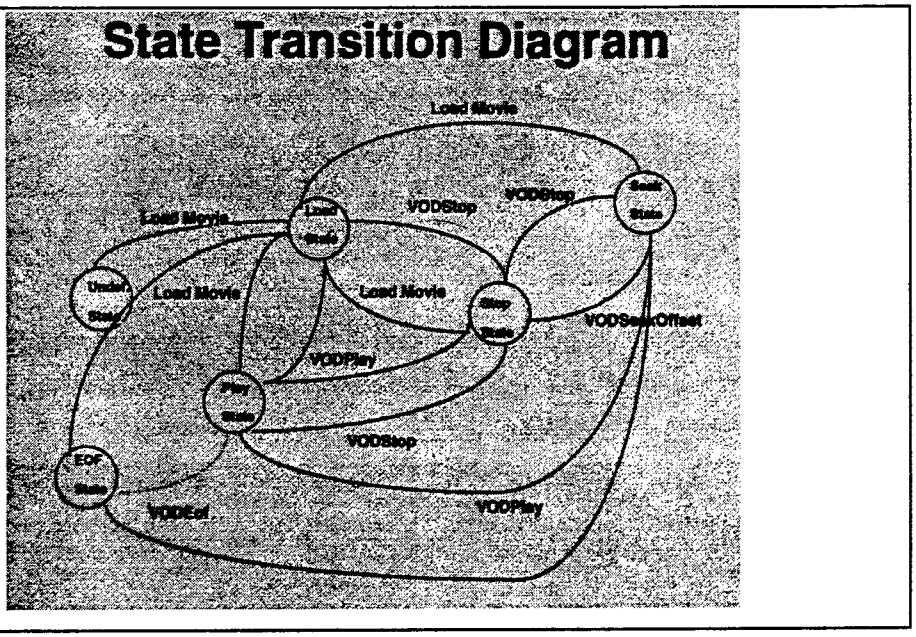


Figure 4.1a. State diagram for a video server with random access capabilities

4.1.4.5 State diagram of the Video Server

We have designed all generations of our video server to handle fully interactive video playback. In addition, if the structure of a media stream supports multiple entry points, the server is capable of random access to such streams. As described in Section 3, our system uses this capability to provide indexed access to the video.

A server must implement a quite complex state machine to provide random access. The state machine of the video server is depicted on Figure 4.1a.

4.1.4.6 Performance Estimates for a PC-based server.

The VoD server with all elements described above was implemented on the Windows NT operating system running on a Pentium 90 PC. The configuration consisted of two Ethernet interfaces connected to an Ethernet switch, providing a peak network bandwidth of 20 Mbits/sec. Storage consisted of a set of four drives configured as a striped logical volume using standard Widows NT system tools. The server was tested in this configuration with multiple client access and was able to saturate the network with 10 clients playing MPEG1 streams. The effective

network usage was 15 Mbits/sec. However, the network is saturated at this level due to the overhead associated with the TCP protocol.

CPU utilization vs. number of clients was nearly linear as the number of clients increased. This is expected since the fine-grained model of parallelism was implemented.

At this stage of the project, there were not enough network resources available to attempt more that 10 clients simultaneously. The implemented server provided seamless service to the clients. The disk load was relatively light since the asynchronous I/O requests were handled by the operating system. Standard optimization was used by the operating system to reduce the seek time for disk accesses and to increase throughput. We estimate that the current server can support 20 MPEG1 clients if enough network resources are available. We also estimate that the server will be CPU bound at this level of use.

Our analysis documents that, using careful design principles with independently optimized server subsystems, it is possible to achieve respectable performance of video servers even on COTS equipment. The architecture described above was also implemented on the SGI platform (described below). An improved server architecture has been implemented on Windows NT platform as a production server for the project (see Section 4.2). We have used this relatively stable architecture as a starting point for the design of a distributed server (Section 4.3).

4.1.5 Server implementation on SGI platforms.

We also implemented the architecture described above on the SGI platform. The port has been relatively difficult since the IRIX operating system is not multithreaded, in spite of SGI's claim. For SGI SMP servers, multithreading in the video server is critical for efficient system utilization and load balancing. The server architecture described above benefits from concurrent processing. For symmetric multiprocessors, this benefit is even higher as different threads can be executed on physically different CPUs. Multithreading is the simplest way of implementing a truly concurrent server. On SMP machines, we rely on the operating system's scheduling to achieve load balancing.

As mentioned above, IRIX does not support kernel multithreading in the strict sense (as defined in DCE). One can, however, use an SGI-specific system call which gives the developer the ability to create a new shared group process. The `sproc` system call is a variant of the standard `fork` call. Like `fork`, the `sproc` call creates a new process that is a clone of the calling process. The difference is that after a `sproc` call, the new child process shares the virtual address space of the parent process rather than simply being a copy of the parent. The parent and child each have their own program-counter value and stack pointer, but all the text and data space is visible to both processes, as well as to multiple child processes of the same parent. The inter-process communication (IPC) is being provided by the operating system. Thus, functionally, the `sproc`

mechanism is nearly equivalent to kernel threads, except the process switching is somewhat heavier. This provides a basic mechanism upon which parallel programs can be built.

Using `sproc`, we could map the Windows NT video server architecture directly to the SGI server architecture. However, we have encountered a number of problems in the implementation process, the most serious being incompatibility of the SGI asynchronous disk I/O with the `sproc` environment: with two asynchronous I/O processes started concurrently in two shared `s-processes`, neither returned. This has been identified as a bug in the SGI asynchronous library, for which SGI was able to provide a system patch.

The SGI server is protocol compatible with all other servers implemented in this project. The server is available on a 4 CPU SGI Onyx machine. Its architecture basically corresponds to the "early design architecture" described above. No further modifications to the SGI server were made during the project time frame.

4.2 Production Video server at NPAC

The production server at NPAC runs on a cluster of Windows NT machines. Before we proceed with a description of the infrastructure, let us take a detailed look at the technical improvements made to this server as compared to the "early" models described above. The changes were made possible by Microsoft's introduction of the "completion ports" in their Windows NT operating system. This facility is not available on UNIX machines yet. Note that "completion ports" are not merely non-blocking socket calls known from UNIX and Winsock socket implementations.

4.2.1 Completion Ports in MS Windows NT

Simplicity in programming and making use of multiple CPUs simultaneously are just some of the reasons for writing multithreaded servers. Let us consider again two models of multithreaded servers. The models described above correspond to the high and low granularity parallelism described above in Section 4.1.4.1, with the extension that the 4 threads in the coarse parallelism model are replaced by a pool of available worker threads.

In the first model, the main thread creates a thread for handling each client. This way, each thread is responsible for taking care of the state of a single client, which makes implementing the server much simpler.

In the second model, a pool of worker threads is created to handle client requests. The main thread then does a select on all the connected sockets and passes any new requests to the worker threads to handle.

These two models have been used in many servers, but they do have problems. The first model does not scale well for high number of active clients. Creating multiple threads takes advantage of

multiple processors but uses excessive system resources, thus causing scheduling overhead. As the number of threads increases, the system spends too much time context-switching threads in and out of their running state. This is one example in which a multithreaded server on a SMP machine may prove to be more inefficient then a single-threaded server on a single processor machine.

The second model suffers mainly from the lack of fairness in servicing clients. A very active client may block other, less active clients. Although this problem can be solved by an application level-scheduling algorithm, it does make the main thread and the worker threads more complicated than they should be. The second model does not use as many system resources as the first, but it still has to make a context-switch for every request between the main thread and a worker thread. This is not a big problem for infrequent requests; but as the frequency of requests increase, the overhead of context-switches could overwhelm the system.

We can now see the possibility that a multithreaded server could be inefficient. Therefore, to produce optimal performance, the programmer must design the server very carefully.

Asynchronous I/O is a very powerful mechanism for any real-time application. It is favorable for a real-time server to use asynchronous I/O when the completion of an I/O instruction may take an undetermined amount of time. Using asynchronous I/O, a server can queue I/O (such as reading on a socket, waiting for a request from the client) and continue processing without having to wait for the completion of the call.

In the first model described above, use of asynchronous I/O might not be necessary; but the design of the second model mandates that we use asynchronous I/O. We will now discuss a new asynchronous I/O mechanism that can be used to better implement the second server model.

A mechanism called "I/O-Completion Ports" was introduced by the Microsoft Corporation in their Windows NT 3.5 operating system. I/O completion ports are used with asynchronous I/O, better known as "Overlapped I/O" in the MS Windows development environment. During or after the creation of an I/O-Completion Port, one or more file handles are associated with the port handle. After an asynchronous I/O operation completes, an I/O completion packet is queued to the I/O Completion port associated with the file handle. At this time, if a thread is waiting for completion, it is awakened to handle the completion of the asynchronous I/O call.

To use a completion port with a file handle, the file or socket must be created with the `FILE_FLAG_OVERLAPPED` flag. This allows asynchronous calls to be made with the file handle and allows the file handle to be associated with a completion port.

A completion port is created with a call to `CreateIoCompletionPort()`. This function is also used to associate an open file handle with an I/O completion port opened earlier. This allows us to dynamically associate new file and socket handles with an I/O completion port.

A thread capable of handling a completion, calls the `GetQueuedCompletionStatus()` function to wait for an I/O completion packet to be queued to the I/O completion port, rather than waiting directly for asynchronous I/O to complete. If there are no completion packets queued at the port, then the thread blocks and waits to be awakened by the system upon the arrival of a completion port. Threads that block their execution on an I/O completion port are awakened in last-in-first-out (LIFO) order, while I/O requests are serviced in first-in-first-out (FIFO) order. This means that when an I/O completion packet is queued to the I/O completion port, the system releases the last thread to block its execution on the port. In this way, a context-switch can be avoided, the cache is not invalidated, and the system can make efficient use of its processing power.

One of the parameters passed to `CreateIoCompletionPort()` is the maximum concurrency level. This value is very important because it limits the number of runable threads associated with the I/O completion port. When the total number of runnable threads associated with the I/O completion port reaches the concurrency value, the system blocks the execution of the threads until the number of runnable threads associated with the I/O completion port drops below the concurrency value. The best concurrency value to specify is the number of CPUs on the machine if the computation for all transaction will take the minimal amount of time. However, if most of the transaction are computationally intensive or have to wait for other events, then a larger concurrency value would be beneficial.

The most efficient scenario occurs when I/O completion packets are waiting in the queue, but the concurrency limit has been reached, with possible threads being blocked. At this point, if a running thread calls `GetQueuedCompletionStatus()`, it will immediately pick up the queued I/O completion packet. No context switches will occur, because threads are used in last-in-first-out order.

If a concurrency value of zero is specified, the system makes an intelligent choice of the number of threads to use for handling completions concurrently. In this case, the system will allow a maximum concurrency of handling completions to be the total number of threads that are running. This makes implementation easier for the programmer, who can rely on the system to automatically increase or decrease the concurrency level.

A thread which has access to the I/O completion port can call the `PostQueuedCompletionStatus()` function to queue its own special-purpose I/O completion packets to the I/O completion port without making any asynchronous I/O calls. This is a very useful tool for notifying worker threads of external events. For example, a worker thread, which is executing a tight loop to handle completions, can be notified to get out of the loop and stop execution.

An I/O completion port is freed when there are no more references to it. The port handle and every file handle associated with the I/O completion port reference the I/O completion port. To

free a completion port, all the file handles must be closed along with the port handle by calling the `CloseHandle()` function.

If we were to implement a multithreaded server using I/O completion ports, our main thread would create an I/O completion port along with a pool of worker threads to wait on the port. This model is similar to the second model discussed above, but it would be more efficient because it does not suffer from the context-switching overhead. A worker thread waiting on the port would find a completion and service it.

We mentioned that fairness is the main problem of the second model. In this model, fairness is built in to the completion port mechanism, since I/O completions are satisfied in FIFO order.

4.2.2 Production version of the Windows NT Video Server

The current production version of the server has been implemented using completion ports, as described above. The server has been tested for almost a year and has proved to be rather stable. While its implementation is "bare bones" (no server setup and management tools), the server has been a workhorse in the project, enabling us to work on implementation of other parts of the system, such as video clients and the metadata database.

The current video server facility at NPAC, built as a result of this project, consists of the following elements:

1) A small video server on which system modifications are tested before going into production mode. This server is based on a 2 CPU Micron PC. The server is an EISA/PCI machine with dual Pentium CPU running at 133 MHz. The memory configuration is 64MB RAM and 512KB secondary cache. Disk storage consists of a 2GB IDE system disk and a 5 disk array connected to an ADAPTEC SCSI controller. The disks are SCSI2, single-ended, narrow drives with total capacity of ~18GB. They are configured into 3 logical NTSF volumes, two of them striped two ways. Network connection is provided by a 10Mbps Ethernet EISA 3COM controller and by an ATM adapter. The ATM card is the 155 Mbps (OC3c) PCI card with multimode fiber interface from FORE Systems, controlled by FORE driver version 4.0. This version of the supports only LANE 1.0 (no support for FORE SPANS or Classical IP). The operating system on this server is Windows NT 4.0 Workstation.

2) The main video server consists of two ALR servers, which are 4 CPU EISA/PCI machines with 4 200 MHz Pentium Pro CPUs each. Memory configuration is 256 MB RAM and 512 KB secondary cache per CPU. Disk storage for video contents is provided by 3 Buslogic fast/wide SCSI adapters, connecting ~200GB of the fast/wide Seagate 10GB drives. Each machine is networked two ways: via 100

Mbps Ethernet card and via the FORE ATM OC3 adapter, identical as the one used in the test server. The operating system is Windows NT 4.0 Server.

For networking details, see Section 7.x.

All servers can run single or multiple copies of the video server described above. The entire facility has been integrated after this project was formally completed. There are no performance data available at this time. It is our intention to complete the implementation of the distributed server (see next Section) and install it on the infrastructure just described.

4.3 nCUBE2 Video Server

The nCUBE2 muiltiprocessor was our primary target for the parallel server platform. An nCUBE2 machine has been installed in NPAC since 1993. The platform has been configured for a parallel database server, i.e., it had significant storage and memory. External network connectivity was weak, but we have acquired a number of network adapters to remedy this, as described below.

In theory, the nCUBE2 platform seems ideal for implementation of a video server. The salient points of nCUBE's architecture are depicted in Fig. 4.2 below. The main processor array forms a hypercube. Each processor sits in a vertex off a hypercube and is connected to the neighboring vertices via a proprietary communication mesh (nCHANNEL). The nCHANNEL links in the nCUBE2 have a nominal bandwidth of 17.5 Mbps. All processors in the main array are identical. The NPAC machine has a 32 MB RAM per array node.

The main array does not connect directly to any I/O devices. The I/O subsystem forms an independent hypercube with CPUs identical to the main array CPUs. Each CPU in the I/O subsystem had 4 MB of RAM. Each I/O node was also connected to a specialized I/O controller: a SCSI controller for the nodes serving disks and network interfaces for the nodes running the TCP/IP stack. NPAC's nCUBE2 has 64 array nodes and 32 I/O nodes supporting SCSI controllers. Each I/O node was connected to two array nodes using the same nCHANNEL technology. In addition, there were 4 I/O nodes with Ethernet adapters and 4 other I/O nodes supporting tape backups and a VME connection to the control workstation. The 32 SCSI controllers were connected to 96 2GB disk drives, with 3 disks per controller, for the total capacity of nearly 200 GB.

Looking at the nominal performance numbers, this configuration is well balanced. The standard SCSI controller is expected to pump data at 3-4 MB/s. This number corresponds nicely to the total bandwidth of connections between the I/O node handling this data stream and the array (2*17.5 Mbps ~= 4 MB/s). The total outgoing network bandwidth was nominally too low, but it was deemed sufficient for modest scaleability experiments. A larger number of network nodes was beyond the fiscal capability of NPAC.

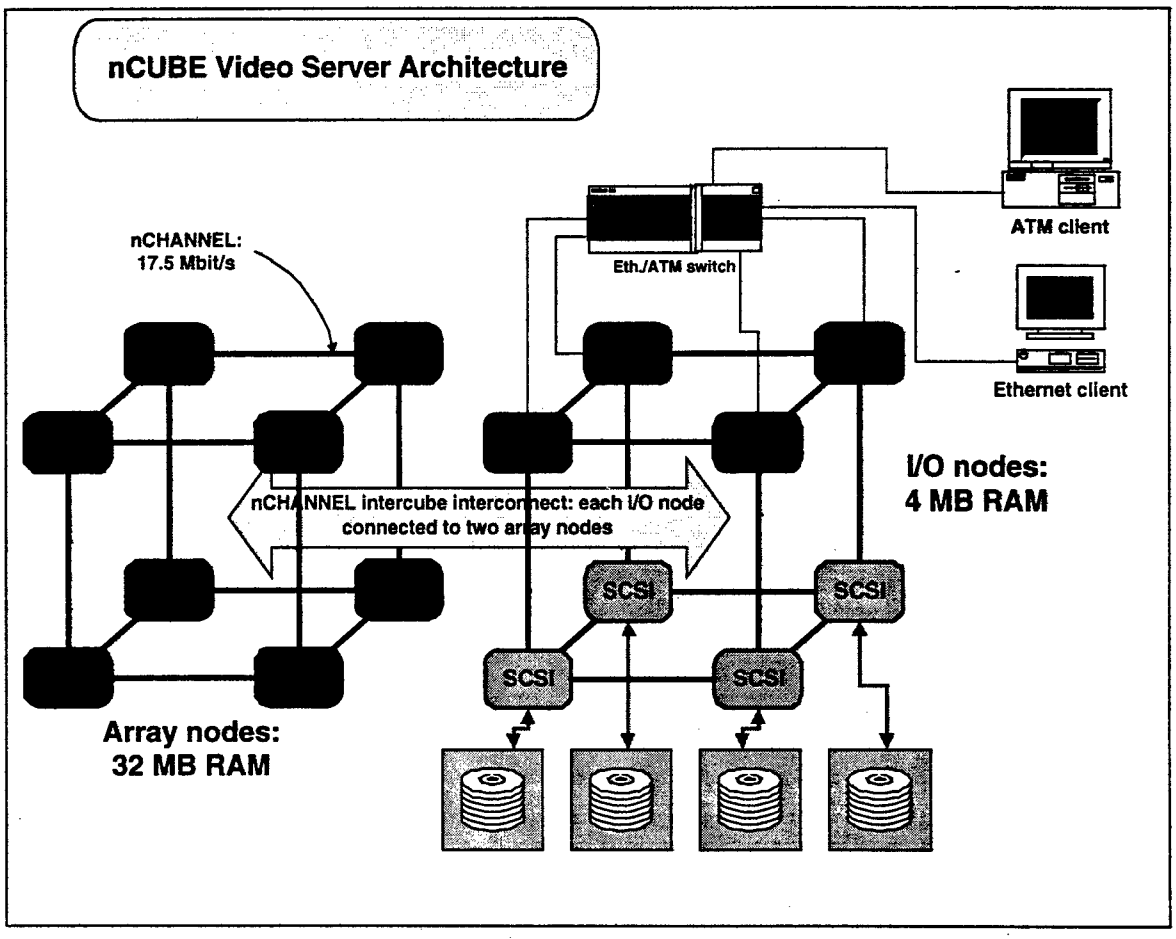


Figure 4.2 Architecture of the nCUBE multiprocessor

The appeal of the nCUBE architecture is easily explained. In the preceding section we have explained the modular structure of the video server and the asynchronous nature of connectivity between them. On nCUBE, these server modules can be mapped onto physical processors and made completely independent. Due to the very high level of concurrency in the video server, we expected a significant performance from the nCUBE platform in spite of its rather outdated and slow CPU chip. Another exciting feature was the shared disk architecture. In the distributed server model described in Section 4.3, we worried about video contents distribution on the disk arrays which are local to the server pumps. But, on nCUBE, multiple video-server processes running on the array have access to all disks connected to the system. This functionality is supported via wormhole routing of messages on the system level: the processor does not need to be directly attached to the I/O node to be able to read its disk contents. Access to the disks is entirely transparent. We hope that this architecture will enable us to experiment with different configurations of the video servers and to investigate performance tradeoffs.

In the remainder of this section, we document the effort to realize the initial vision. The outcome was negative since we were unable to implement a server capable of serving even one continuous MPEG1 video stream from one network interface. We will explain in detail the reasons for the failure in the "lessons learned" section.

4.3.1 NCUBE2 network connectivity

Our initial plan was to purchase an OC3 ATM interface that nCUBE engineers planned to feed from 4 doubly connected I/O nodes. This plan turned out to be unfeasible because nCUBE decided that their ATM NIC would use an ATM adaptation layer 1. This created an incompatibility problem with our ATM infrastructure, which supports only AAL3/4 and AAL5. We have analyzed and considered several options, including:

1) a SCSI to ATM converter. After analyzing this architecture, we dismissed it as an ad hoc, non-scaleable solution without the mechanisms necessary to support quality of service requirements.

2) use of a HiPPI interface and connect to NYNET via GTE switch. We found this solution unfeasible since the GTE ATM switch only provides bridging for HiPPI signal, whereas our project required HiPPI-to-ATM conversion.

3) use of multiple Ethernet connections to nCUBE along with a LAN access switch used as a multiplexer/concentrator. We analyzed and tested the LAX20, a device from FORE Systems and determined that, in order to have all requested functionality, the device will need a number of improvements in software, most notably the LAN emulation; otherwise it offers an acceptable solution which implements server connectivity over a number of dedicated Ethernet lines multiplexed into an ATM trunk. For this implementation, the dedicated Ethernet lines can be seen as permanent virtual circuits with guaranteed bandwidth.

We adopted option 3 because NPAC has acquired three additional nCUBE I/O nodes with Ethernet controllers and the add-on 4 Ethernet card for our LAX-20 device. The controllers have been connected to the dedicated Ethernet segments that, in turn, were connected to the LAX-20 switched Ethernet ports. With the FORE company promising LANE implementation any day, we hoped for direct nCUBE-to-ATM client connectivity. This, incidentally, has never happened since FORE dropped the LAX-20 product without providing LANE implementation. Nevertheless, we were able to use video clients connected to Ethernet ports of another LAX-20, with the ATM link between them providing bridging capability.

4.3.2 Video Server Implementation on nCUBE

This section summarizes our efforts to implement an operational video server on nCUBE. These efforts alone accounted for at least 30% of the total project, so we feel that a relatively detailed report is justified.

As a first step, request-based video pump software was ported to nCUBE. The request-based video pump denotes a TCP/IP system where the data (video file) is transferred from the server to the client on demand, i.e. the client maintains a buffer and requests data from the server as needed.

We have found a number of undocumented idiosyncrasies of the nCUBE system software. Although we started from the fully functional software written for the standard UNIX platform, the porting process went through multiple stages during which nCUBE-related difficulties were discovered, analyzed, and, when possible, circumvented.

- A prototype TCP-UDP/IP system was constructed to test existing protocol performance over Ethernet channels. The initial implementation of the video server contained a single network/disk server. Upon receiving a request, the disk server would retrieve the data from disk and send it to the client.

  Findings: This design led to "hiccup" effects at the client end (an effect of freezing the video stream at regular time intervals). Clearly, the video decoder buffers were starved by the server, which was unable to provide data at the requested rate. We suspect that the disk delay at the server was responsible.

- We then modified the server by adding a circular buffer and allowing it to pre-fetch data immediately upon satisfying a request. In this manner, we hoped that network latency and disk latency would be overlapped.

  Findings: This design did not ameliorate the "hiccup" behavior. We found that requests arrived with an inter-arrival time that exceeded the pre-fetch delay time. Therefore, a further alteration was needed.

  One of the hurdles in the design procedure was the inaccessibility of the nBSD server process (nCUBE implementation of the Ethernet driver and the TCP/IP stack). Thus, exact measurements of network latency within the nCUBE was difficult to obtain.

- The server was modified by separating the network and disk server functions into separate processes running in parallel on different processors of the nCUBE. The disk server was equipped with a pre-fetch buffer which would be kept full constantly. The network server was equipped with a double buffer to which it pre-fetched data from the disk server. The latency between the disk and network

server is low and communication takes place at about 2 MBps. All buffers are filled during initialization (after a movie has been selected by a client); all subsequent prefetching is done after requests are satisfied. This design offers the minimum latency from the disk. Due to prefetching and initialization, all accesses have delay equivalent to memory accesses (i.e., disk access times overlap completely with other accesses).

Findings: The design change had minor effect upon the "hiccup" behavior. Fortunately, the design facilitated the approximate measurement of network latency on the nCUBE by separating network and disk accesses. We discovered that network write access on the nCUBE took place at about 11.5 KBps on average, which is much less than disk access bandwidth. This was a completely unanticipated result. Contributing to this latency is the requirement that network transfers be performed at 32 KB increments (the network daemon malfunctions otherwise—this requirement is undocumented),

- Next, we modified the client to allow it to pre-fetch data from the server. The client would then simply send requests for data in advance of the actual use of that data. We expected that this mechanism should reduce access times at the client-to-memory transfer times, approximately at the expense of the kernel buffer space allocated to a socket connection.

- Also, we changed the protocol to one based on non-request; that is,- data is pushed to the client on a pre-scheduled basis. (Note that this is the architecture adopted for other servers, although at this junction of the project, the architecture for other platforms was not defined yet). In the context of nCUBE server difficulties, the advantage of the push approach is that almost any latency can be hidden by increasing buffer size and by taking advantage of pipelining effects.

  Findings : We have not achieved requested performance parameters. To clarify the reasons for this failure, we have designed and implemented a number of low-level tests that benchmark the performance of the nCUBE networking infrastructure. These tests show that the performance problems persist even for the simplest, low-level data transfer operations involving nCUBE implementation of the socket library.

These findings were inconsistent with the satisfactory performance of the Oracle Media Server software on the nCUBE2 platform. We attempted to clarify this puzzle with nCUBE representatives and learned that the Oracle developers have rewritten significant parts of the nCUBE OS level software to achieve the performance necessary for a video server. nCUBE does not own these changes and they are not being distributed with the standard operating system.

Based upon these findings, we decided that the *nCUBE system code as currently provided will be unable to support VoD requirements.*

Given the accumulated research investment into the hypercube architecture for VoD, we have decided not to abandon the platform but rather to improve the software performance by rewriting the system networking code. Since system source access was necessary to implement the improvements, we have negotiated with nCUBE a source code license for the relevant parts of the operation system software.

After receiving and compiling the nCUBE OS source code for networking, we conducted a detailed review of this code. The findings are summarized briefly below:

- Network Library: The nCUBE network library consists of primarily BSD network code (e.g., socket library) at the upper layers up to the application interface. The lower layer of the network library architecture consists of modules of nCUBE code which provide the glue between the lower layer of the BSD code and the nCUBE communications library. These layers contain code to map socket descriptors, port ids, server ids, request sequence nos., etc. into structures with some semantic significance to the nCUBE communications library.

- nBSD (network) Server: The nBSD server is multithreaded, supporting either synchronous or asynchronous network I/O. The scheduling discipline has not been completely analyzed, although we can say that it makes use of a prioritized round-robin schedule. The nBSD server contains modules to: manage threads and provide thread services, maintain server (system) tables which map from socket structures to nCUBE communication structures, perform packetizing functions, and perform low-level driver functions. The nBSD server treats all connecting processes as socket clients. This includes both connections to external clients and connections to the processes running on the nCUBE array nodes. This is a highly unusual situation: the TCP/IP layer is known to introduce performance bottlenecks due to multiple copy operations. On nCUBE, the TCP/IP stack is accessed by the server code and twice by the nBSD code - this is two times more than on a standard workstation. As far as we can see, no provisions are made to optimize access for the array processes. Further, we determined that nCUBE uses a reliable protocol (TCP) over a completely reliable link layer (nCHANNEL). This is obviously wasteful. From this preliminary investigation, it is clear that the network architecture results in unnecessary overhead during communication between nCUBE array nodes and nBSD server nodes. The overhead here is located on both ends of any such connection.

As a result of the above analysis, we designed a light-weight path between the nCUBE array nodes and the nBSD server in order to decrease overhead on the nCUBE array node, but also on the nBSD server node. We expected significant performance increases resulting from bypassing the original nCUBE architecture.

In addition to the design of the light-weight path between the video pumps and the nBSD server, we identified a number of performance bottlenecks within the nBSD server itself. To remedy these, we implemented our own version of the nBSD Ethernet driver.

Next, we inserted the light-weight communication path into the operating system layers. Although we had access to the source of the networking layer, we had no documentation. Thus, we did not know exactly how this layer is called by other parts of the OS. We assumed that the networking layer routes end-to-end all messages that enter it. Unfortunately, a detailed analysis of the original nCUBE code revealed further complications: in order to support the NFS protocol, all networking calls are routed via the so called "virtual file system" layer. This solution supports a completely transparent file access interface on nCUBE, but it imposes a heavy performance penalty on all networking operations. We were unable to obtain source code for the VFS layer.

Appendix 7 contains a detailed analysis of data flow through the layers of the nCUBE operating system.

Our continued effort to send the light-weight protocol messages through the VFS layer ultimately failed. We were able to send and receive messages initially; but, as described in Appendix 7, the communication process broke, with reports of state errors or the continued transmission of corrupted messages.

At this point of the project, we became concerned with two issues: (1) the amount of work involved in rewriting the nCUBE OS was imposing a heavy load on the project resources; and (2) the stability of the company had been questioned recently by inside sources. The company's survival depended clearly on their ability to introduce a new family of machines. But in either case, the nCUBE2 platform would became obsolete. It was clear also that the company was no longer interested in helping us solve the performance problem. As a result of this consideration and after consulting project management in Rome Laboratory, the work on nCUBE was discontinued. The final conclusions can be formulated as follows:

nCUBE2 architecture was an extremely appealing platform for implementation of the VoD server. However, nCUBE2 has been built using outdated chip technology. The nCUBE2 CPU corresponds roughly to the Intel 386 chip. CPU performance per se would not preclude using nCUBE2 as a video server, although the cost-performance ratio is questionable. However, a combination of its low processor performance with disastrous design of the nCUBE2 operating system makes this platform unsuitable for implementation of a video server except perhaps for low bit rate streams. Specifically, three most serious performance bottlenecks are:

1) the nBSD server treating array processes as IP clients;

2) stacked use of reliable protocols on top of each other; and

3) filtering of all network calls via a notoriously slow VFS layer[7]

To implement an operational video server, it is necessary to (1) write a completely new file system on top of the existing SCSI drivers, specifically removing the entire VFS layer, and (2) rewrite the network layer, replacing the IP over nCHANNEL links with native message passing. We felt we could tackle the first task easily enough (in fact, we have implemented the double buffering scheme described earlier using raw disk access); but a complete implementation of a new networking layer, while possible, was well beyond limited goals of this project. Consequently, the effort has been discontinued.

4.4 Distributed Server Architecture

As described in the previous section, our attempt to implement a scaleable video server on a typical MPP platform failed due to poor quality of the system software. Now, we observe that the issue of software quality has been one of the reasons that brought the federal HPCC program to an abrupt end.. One of the lessons learned from the HPCC program is that, while it is possible to build expensive hardware for custom HPCC applications, it is unlikely that a community of software developers addressing a relatively small market of very sophisticated users will develop truly robust and functional software .

The initial VoD thrust was stimulated by the telecommunication industry's desire to deliver new kinds of service to the homes of their subscribers. The model behind this scenario is that of a large contents provider feeding a network serving a city or a town. One example of a medium size installation is the Time-Warner Full Service Network in Orlando, Florida, which initially served ~ 4,000 customers. In comparison, a small video server would serve a few hundred customers

It is our belief that the only video server applications that warrant use of the HPCC-class platforms are the entertainment-centered, consumer networks. It is only at this scale of demand that the expense of building a VoD service around an MPP is warranted. Moreover, the recent demise of the Bell Atlantic VoD service, which remained operational for only few weeks and cost the company more than $100M, suggests that even large-scale use may be unjustifiable.

For educational, intranet, or Internet applications, use of MPPs as a VoD platform is economically unfeasible. For example, the cost of the entire nCUBE platform at NPAC - was nearly $2M. In addition, we paid nearly $20K for three additional Ethernet nodes for nCUBE2, only to discover

[7] To further verify our diagnosis we obtained and installed an implementation of the Network File System for nCUBE2. During extensive performance tests we discovered that maximal bandwidth for NFS transfers from nCUBE never exceeded 0.5 Mbps with transfers over Ethernet lines. This performance is easily exceeded by workstations such as low-end PCs or Apple's Macs.

that these modules are unable to transmit even a single MPEG1 video stream. For the price of this expansion alone we could have purchased 4 high-end PC machines that could easily serve up to 80 MPEG1 streams.

Obviously, one cannot make a blanket statement that MPP machines are not useful as video servers. On the contrary, we believe that parallelism is critical for scaleable servers. The next generation of the nCUBE machines, series 3000, is reported to support very high performance video servers. The main problem is not that these platforms won't scale up, but rather that they do not scale down. The expense for a lowest-end HPCC platform, with great expansion potential but with performance only marginally better than a COTS machine, is an order of magnitude (or more) higher than the price of a COTS installation.

Confronted with the failure of the nCUBE effort in this project, we felt we need to propose an alternative solution. NPAC's hardware inventory offered two other HPCC platforms: the IBM SPA machine and the DEC Alpha cluster. We considered both and decided that both are unsuitable because both represent the recent trend in the HPCC industry to build HPCC platforms from UNIX workstations. In the context of video server implementation, we note the following:

1) Both the storage and network modules in the workstations use exactly the same technology that is available for PC platforms. The performance of these modules in UNIX and PC platforms is comparable[8]. In many cases, however, the cost of disk drives and network cards from UNIX vendors is significantly higher.

2) We have observed (see Section 4.1) that video servers can become CPU bound. As it is well known, the newest CPU technology is first implemented in midrange servers and desktop machines. The HPCC platforms are often built from outdated CPUs. The most notable example is the IBM SP2, IBM's HPCC flagship, which uses four-year old CPU technology that does not match the power of a commercial grade PC.

3) Except for the SMP platforms, architectures such as SP2 and workstation clusters do not offer any architectural advantages over a cluster of PC machines. For a video server, a shared disk architecture is clearly advantageous, as discussed in Section 4.3. Although IBM Virtual Shared Disk emulates such functionality, its setup is completely inflexible and very difficult for dynamic performance tuning.

[8] In at least one casewe found the performance of PC modules better. The early ATM interfaces on UNIX platforms offer only a fraction of the nominal bandwidth. The first interface (hardware and software drivers) we have seen that approached the nominal performance numbers was the FORE OC3 PCI card for Windows NT platforms, which, using the IP protocol, exceeded a of 100 Mbps.

Based on the above observations and on our experience with video server implementations on PC platforms, we decided that, given the applications targeted by this project, the only feasible short-term solution is a distributed video server implemented on a cluster of Windows NT PC machines. We also believe that scaleability up to the traditional HPCC range will be provided by platforms utilizing Intel CPUs to build highly parallel machines entirely from commodity components. Examples of such machines have been built recently in national laboratories and are offered commercially. A good example is Parsystec's CC platform.

The architecture we propose below assumes that each instance of the server will be locally optimized according to the lines described in Sections 4.1 and 4.2. We assume also that, for a given hardware configuration, each instance will have defined a performance metrics that will allow the distributed server to apply a sound admission policy and load balancing. The two basic mechanisms for providing load balancing are the ability of the server to move client requests from one server to another and the ability to dynamically change media contents in local server repositories in response to statistical load analysis. The architecture we propose does not postulate striping of media streams across multiple servers.

In the following section we describe the distributed video server architecture. This architecture was designed in the final stage of the project, mostly as a result of the experiences we gathered by implementing earlier versions. Prior to the end of the project time frame, we defined the overall architecture of the distributed server and we created a detailed system specification. As for implementation, the core of the server code has been written and tested with dummy video clients. Other tools, like server administration and monitoring, have been partly implemented. It is our intent to continue our research into distributed video servers and to use this architecture as our next production server software.

The main differences between the new server and the existing implementations are:

- the new server's data structures have been designed so that it is possible to monitor all aspects of server operation via a remote monitoring tool, and to administer the server by the same means
- the new server supports security in terms of access control
- the new server has an interface for direct interaction with the database back-end. This interface can be used to implement more reliable data consistency measures. In particular, the new server can respond to queries regarding the existence of the media files in its data repository, and can store utilization data in the database
- a collection of new servers can act as an entity. Multiple instances of the server can run on the same machine (especially on an SMP) or on different machines. In

the later case, the servers do not share a media database. The load balancing measures allow servers either to move media requests from one server to another (hand-off capability) or to transfer a copy of a media stream from one server to another in order to circumvent statistically significant imbalances between admission requests and the capacity of other servers to serve particular contents.

- the new server has a slightly modified thread architecture as compared to the current production server.

The new server has been code named Oasis and is referred to in this way in the remainder of this section.

4.4.1 Server Design Issues

4.4.1.1 A Scaleable Server

As the computing age continues to mature, millions of clients will need hundreds of thousands of servers. Most servers will be tiny, but some will need to be large. With the belief that this spectrum of different sized servers can be built using small modular components, we decided to implement Oasis using small modular components that work with each other, while maintaining abstraction amongst them.

Oasis had to run on a typical home PC with a small amount of disk space, a single processor, and possibly an Ethernet LAN connection, functioning as a personal server or as server intended to service a small office of people . But Oasis also had to run on multiprocessor machines with ample amount of disk space and memory and possibly multiple network interfaces, functioning as a video server intended to service an Intranet. Moreover, if needed, Oasis had to service many thousands of people, as in a large residential area.

4.4.1.2 A Distributed Server

One way of scaling a service is to distribute it, allowing multiple machines to work with each other to improve the service. It was our task to allow for distribution of a single Oasis Server among multiple machines and to make an it work cooperatively with other Oasis Servers. Oasis needed to be configurable (1) so that parts of it could run on other machines, (2) to decrease the computing requirements, and (3) to increase the network bandwidth for servicing the clients.

A pool of Oasis servers working together had to balance the load produced by all the clients. To do this, an Oasis server had to pass and receive active clients from and to other Oasis servers easily and seamlessly. In addition, an Oasis server, if needed, had to act as a proxy server between a client and another Oasis Server that are geographically far apart.

Multiple Oasis servers had to be able to work together to service a single client. While a client could initially connect to a single server, it could then request media located on other servers in the pool.

4.4.1.3 Architectural Model

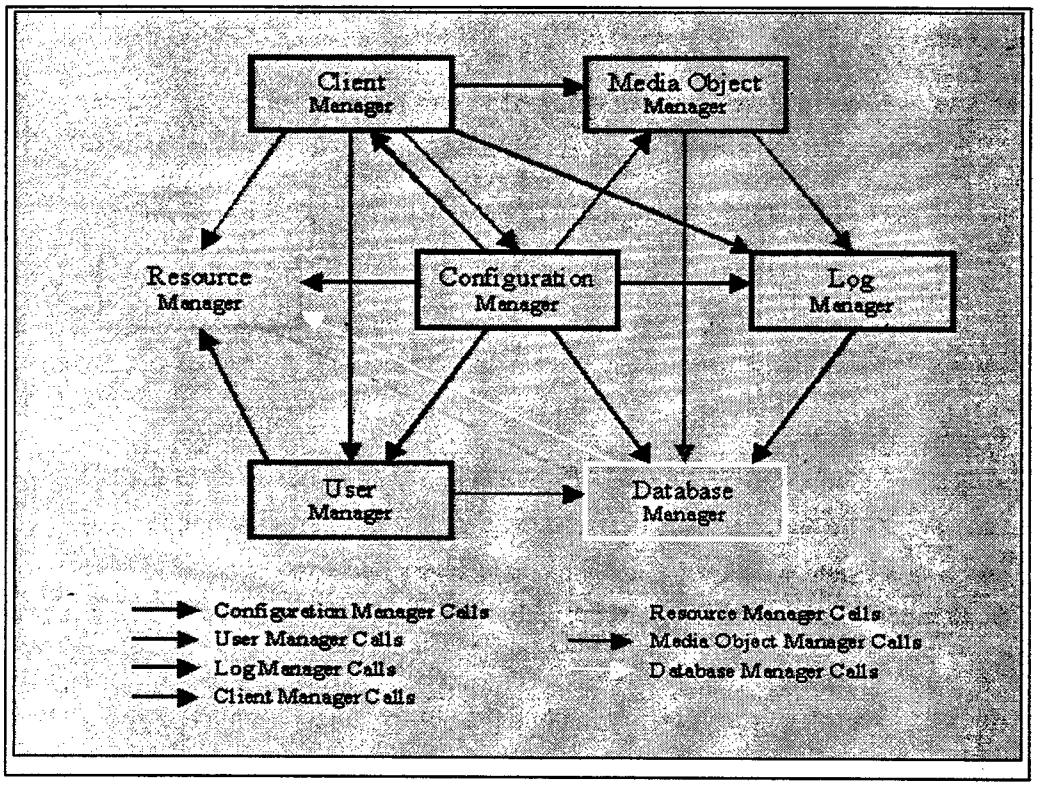


Figure 4.3 General architecture of the Oasis server.

Oasis can be broken up into seven main modules. Each of these modules provides a set of interfaces for other modules to call. Each of the modules has a module interface and something we call the module core. The module core contains the main functionality of the module while the module interface is just the communication mechanism between the modules. Any installation of Oasis that contains all the module interfaces on a single machine we call a *parent* Oasis server. Any installation of a set of module cores on a machine that is not a parent Oasis server, we call a *child* Oasis server. For each parent installation of Oasis, all the module interfaces must exist on the same machine, but not all the module cores. The core of some of the modules can be configured to reside on different machines. This gives Oasis the ability to conform to many different types of server architectures, enabling scaling of the service. We will now outline the general structure for a module.

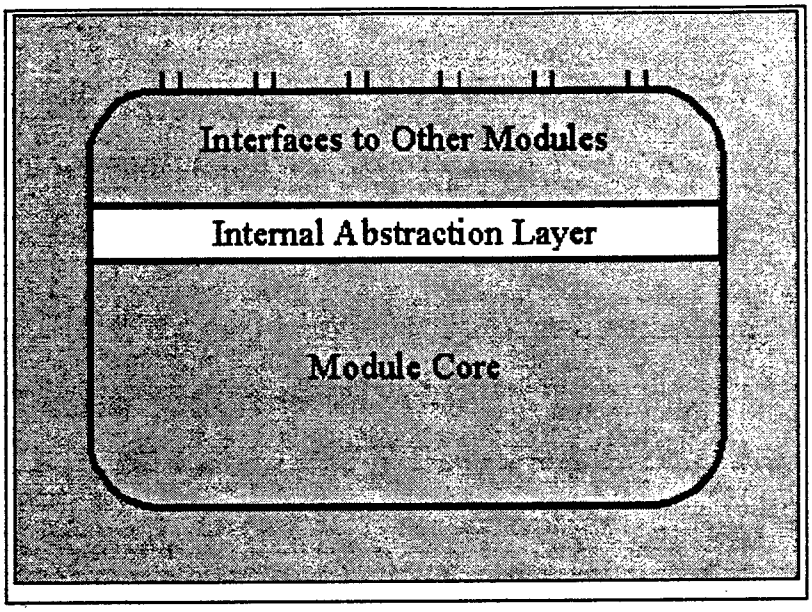


Figure 4.4 Structure of the Oasis module.

As we mentioned earlier, each module has a Module Interface and Module Core. For some of the Oasis modules, the Module Core does not have to reside on the same machine as the Module Interface. The Interface provides a standard way for communicating with other modules, namely by using function calls. The interface passes any received functions to the Module Core, which is responsible for the processing requests from other modules. To allow the Interface and the Core to reside on different machines, we have to provide an abstracting communication mechanism between the Interface and the Core. This *Internal Abstraction Layer* within the Module enables communication between the Modules' Interface and the Core over many different types of communication media, whether it is ATM, Ethernet, or Shared Memory. Ideally, different Abstraction Layers would be available; and, depending on the configuration of the server, appropriate Abstraction Layers would be used.

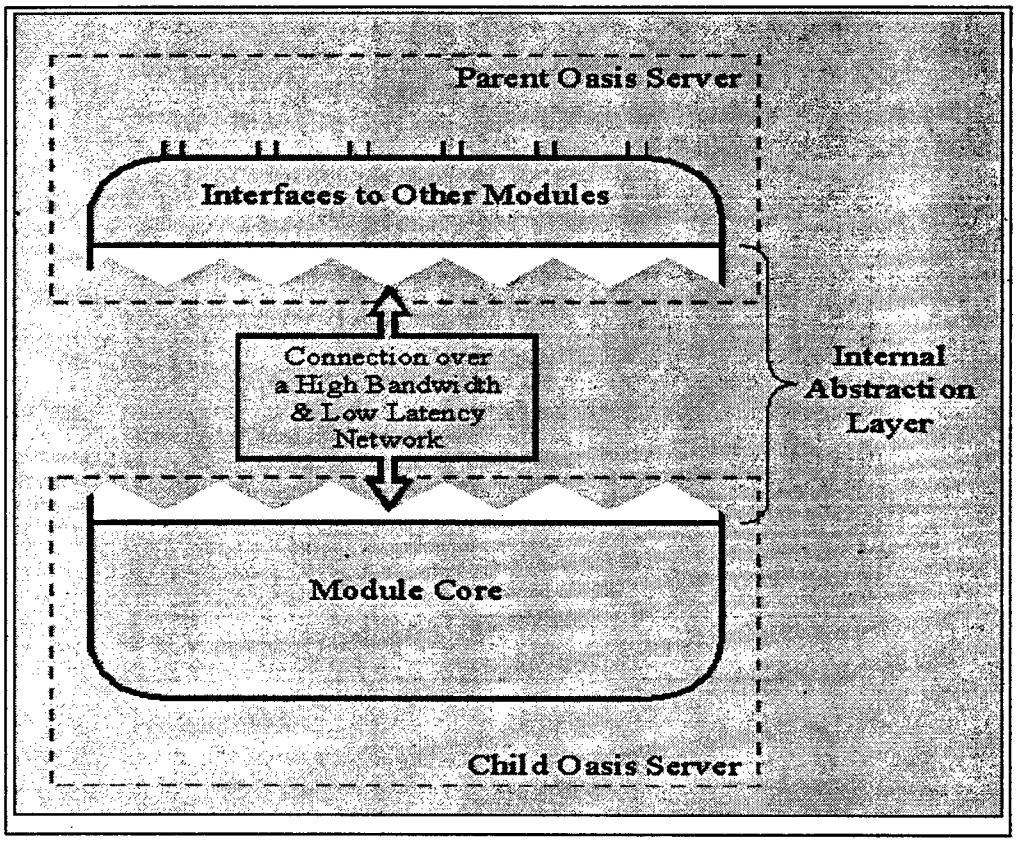


Figure 4.5. Oasis module split to run on different machines

If the Interface and the Core reside on the same machine, then a simple Abstraction Layer Module can be used to simply call functions between the Interface and the Core. If the Interface and the Core reside on different machines, then the Abstraction Layer must use sockets or possibly other protocols to communicate with its remote end. In this case, the Abstraction Layer has to transfer possibly large chunks of data between the two machines. This tells us that some Abstraction Layer Modules that use networks should be used over a very high speed and low latency network; and it reminds us that administrator configuration of the system is as important as the design and implementation of the system.

4.4.1.4 Database Manager

The database manager is responsible for managing the database needs of Oasis. The Database Manager provides a set of interfaces for other modules to use. The Database Manager makes use of the Internal Abstraction Layer, allowing the Core Database Module to reside on another computer. Since the Database Manager provides a standard Interface for other modules of the system and because of the Internal Abstraction Layer, any database, such as Oracle, Access, or Fox Pro, can be easily integrated into the system. Below is a diagram showing the Database Manager.

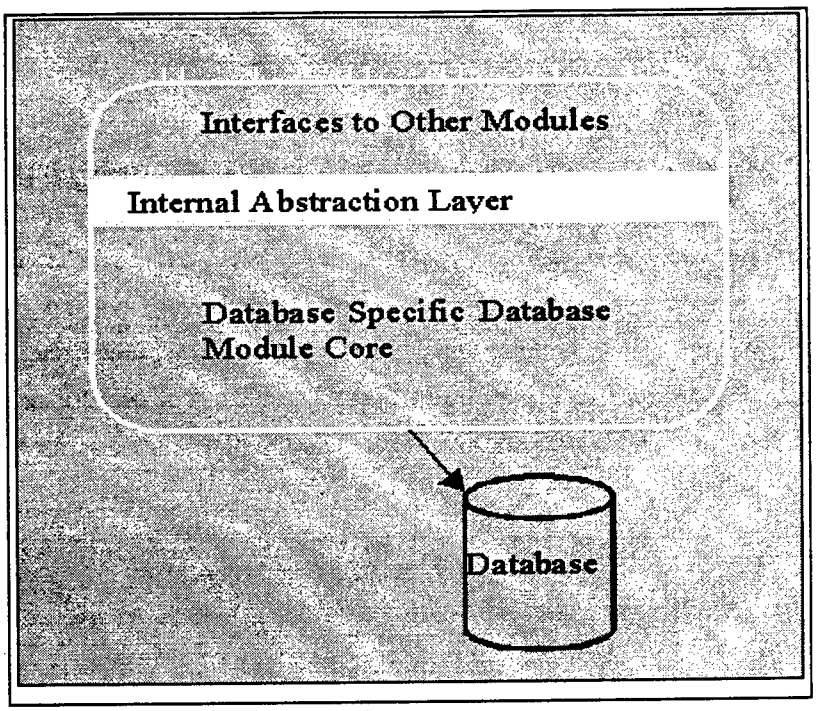


Figure 5.6 Oasis Database Manager

The Database Manager only requires an interface from a Resource Manager residing on the system where its core is located. Please see the next subsection for the reasons why access to the Resource Manager is so important.

4.4.1.5 Resource Manager

For any application (e.g., a. Video Server) to be stable and robust, it must monitor the system it is running on. It must know whether any request made from the system, such as allocation of memory from the heap, can be satisfied and whether this request will result in a less efficient state either for the system of itself.

The task of keeping track of the system's integrity is the task of the Resource Manager, which generates critical events, make reservations for any system resources, and reports resource usage.

It provides a standard interface to all other modules of Oasis, through which resources can be allocated. Other modules do resource allocation, but they are not allowed to make any resource allocation before logically reserving it with the Resource Manager.

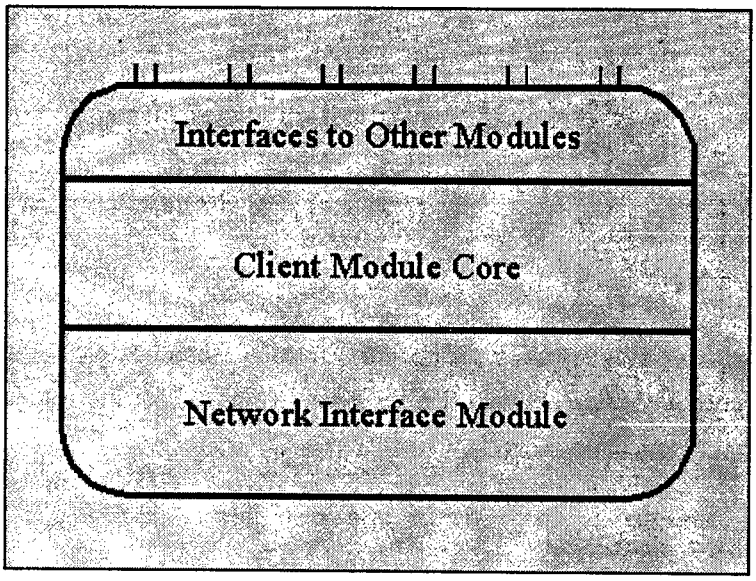
It is possible that at any time another application running on the system can allocate resources such as disk space or memory which might breach the threshold of *minimum free resources warning level* set by the Configuration Manager. At this time, the Resource Manager will generate

a critical event warning all modules. Modules respond by avoiding any further allocation of resources and possibly reducing the amount of resources allocated.

Of course, just as resource reservations are made through the Resource Manager, resource cancellations are also made through the Resource Manager.

4.4.1.6 Client Manager

This is the most important component of Oasis Media Server since it is responsible for accepting connections from clients and servicing them. It allocates resources and tracks the state of each client and the Media Objects requested by the clients. It allows for good scaling by providing service through multiple network interfaces. The Client Manager's architecture is represented in the figure below.

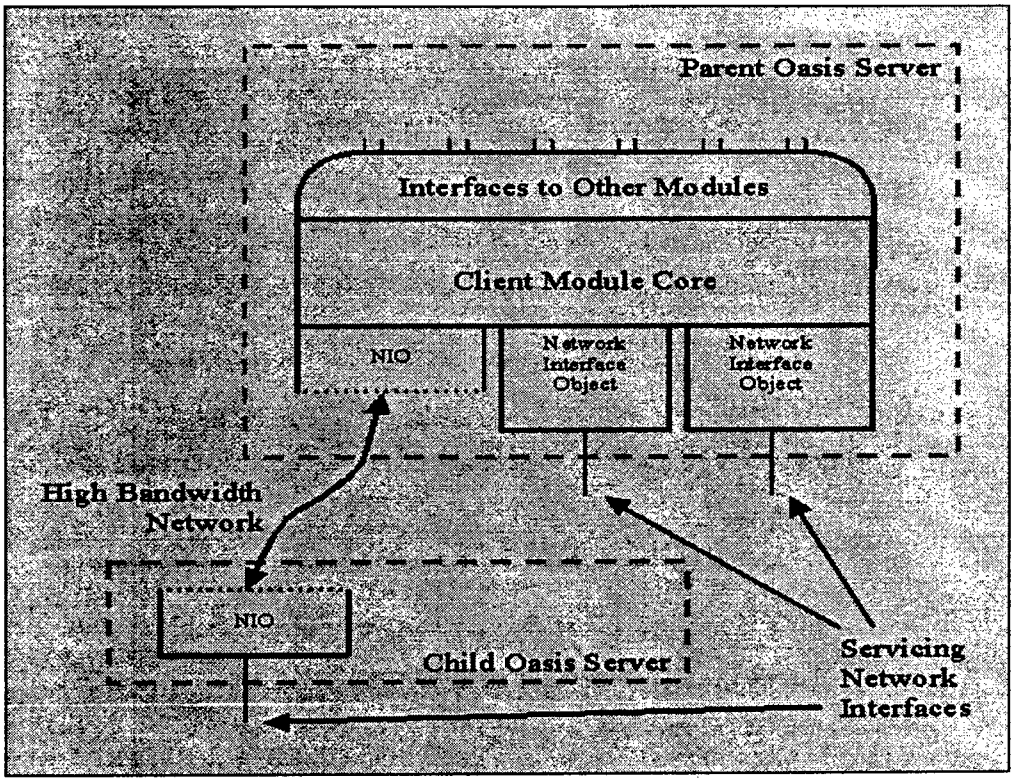


〈 Figure 4.7. Oasis Client Manager module

As we can see from this figure, the Client Manager does not make use of the so-called Internal Abstraction Layer. This is because the Core of the Client Manager Module must reside at the parent Oasis server, whereby using the central server model in dealing with clients local to itself.

To provide service through multiple network interfaces, an Object of the Network Interface Module is instantiated for each of the network interfaces used to service clients. A service network interface does not have to be on the local system. To provide better scaling, the service network interfaces can be distributed among multiple systems. This is done to increase the total amount of network bandwidth for servicing clients and possibly help in servicing a larger geographical areas. Each instance of the Network Interface Module is responsible for network I/O done through the network interface it is assigned to. It is also responsible for any communication between the parent Oasis Server and the remote system that holds the remote network interface. By

dynamically adding and deleting instances of Network Interface Modules for remote network interfaces through special, high bandwidth, low latency interconnects, proxy servicing is achieved.



〈 Figure 4.8. Remote network interface of the Oasis server

4.4.1.7 User Manager

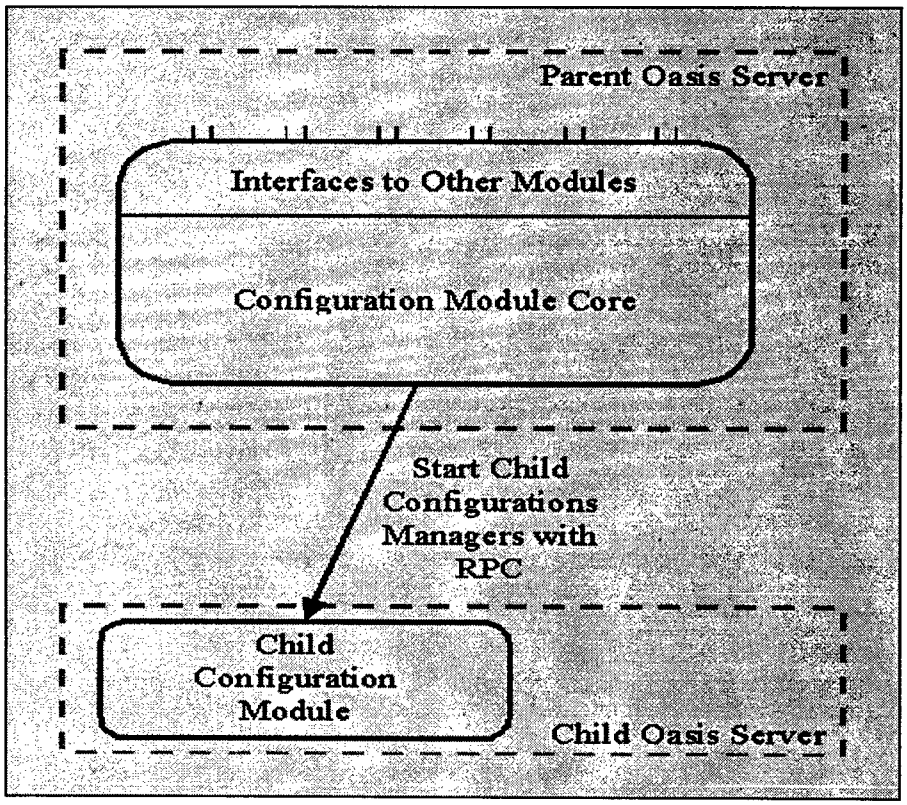
The User Manager keeps an account of the users, their privileges and their activities. It provides an interface to the Client Manager, which queries the User Manager to provide Access Control for the system. Special User accounts are kept for special clients such as guests, auditors, content providers, as well as monitoring and configuration clients such as Shade. Its architecture is rather simple since it does not make use of the Internal Abstraction Layer. It requires an interface from the Database Manager for accessing local or remote databases for adding, deleting and verifying users.

4.4.1.8 Log Manager

It is important for a service provider to keep track of the activities of the system since this is an important basis for improving the system, as well as a way to audit clients. The Log Manager records anything that the other Modules want to keep track of. It keeps the logs through the Database Manager. Its other responsibility is to filter the logs and events at the request of the Configuration Manager. Also, it is the job the Log Manager to time stamp event logs so that more meaningful data can be returned back to special monitoring and auditing clients. This is the

reason why even the Database Manager should log things through the Log Manager, although it could easily do it so itself.

4.4.1.9 Configuration Manager



〈 Figure 4.9. Oasis Configuration manager

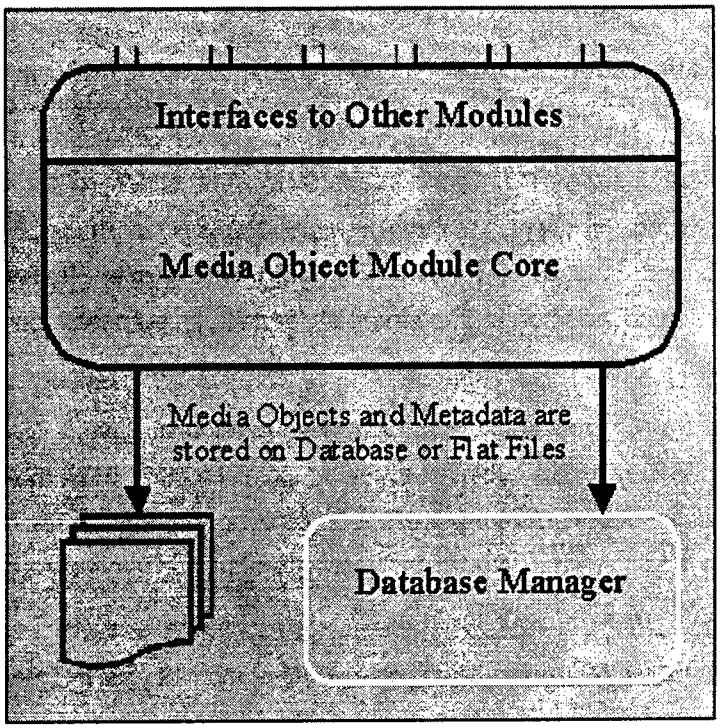
The Configuration Manager is responsible for the configuration of all modules. It instantiates, sets up, and "starts" all other modules. Although every other module provides an interface for the Configuration Manager, the Configuration Manager only provides an interface for the Client Manager because Oasis allows special clients to change the configuration of the system and the Client Manager has the task of passing configuration requests to the Configuration Manager. The figure below shows the internal structure of the Configuration Manager.

The Configuration Manager is the only module that doesn't store its data in the Database Manager. It uses flat files to keep its own configuration information.

The Configuration Manager is also responsible for instantiating, initializing and starting all remote modules on child Oasis Servers. This is done by starting a copy of itself at a remote machine using RPC, which then starts all the modules for the child Oasis Server.

4.4.1.10 Media Object Manager

Media Objects are stored, retrieved, and replicated at remote machines by the Media Object Manager. It can store Media Objects as best suited, whether in flat files, within archiving media (ex. Tape Drive), or within a local database through the Database Manager. It is responsible for I/O required to store and retrieve Media Objects from Local Disk and archiving media, whereas the Database Manager will handle I/O for Media Objects stored at the Database.



〈 Figure 4.10 Oasis server Media Object manager

The Media Object Manager is also responsible for reporting the availability of Media Objects at the local Oasis server. It keeps track of all available Media Objects using the Database Manager. Media Object characteristics such as Bandwidth requirements, file size, and availability are just some of the variables stored in the database for quick metadata retrieval.

Replications of Media Objects at other Oasis Servers are handled by the Media Object Manager by acting as a special client, namely as *a content provider.*

4.4.1.11 Implementation

Our intentions were to build a sound and robust product while making sure that it would not limit our research. It was important to use good Object Oriented Programming style during the implementation of Oasis. We have made full use of modularity to allow easy updates and additions to the system. This allowed us to easily plug in new code to improve sections of the system and add new features. Modular programming also made the system much easier to debug

and test. We wanted to make sure that Oasis did not crash under heavy loads and that it did not alter the integrity of the system on which it ran. This is one of the reasons why we have put a lot of effort in monitoring the system that Oasis was to run on.

Current implementation of Oasis is somewhat limited compared to the Functional Model provided above. The structure described above was put in place using a highly rich set of C++ virtual classes supplemented by the writing of many derived classes to implement a working server. This server is limited in several ways:

- **Configuration Manager:** Not yet implemented to use RPC to start remote copies of itself to enable a child Oasis server.
- **Client Manager:** Network Interface Module needs to be expanded to allow servicing of clients through remote Network interfaces. The Configuration Manager must be fully implemented for this to happen. Passing clients from one Server to another is implemented for testing purposes only following special events generated through the user interface of Oasis.
- **Media Object Manager:** Currently, cannot act as a content provider for replicating Media Objects at other locations; this must be done manually at present. Media Object Metadata is retrieved through the Database Manager; but the addition and deletion of Media Objects and their metadata must be done manually.
- **Resource Manager:** Total network I/O at the Network Interface is not monitored, though the network I/O produced by Oasis is. More resource information might be accessible from Windows NT, but we don't know how to retrieve this information at present.
- **User Manager:** Interface to the Client Manager is not complete. Users cannot be added or deleted from a client; they must be added and deleted manually from a text file. The User Manager uses interfaces provided by the Database Manager(which does the actual file I/O) to verify user information.
- **Log Manager:** The log manager currently does not have any filtering mechanism. It appends a date and time stamp to each log generated by other modules and gives it to the database, which merely writes this data into a flat file. Retrieval of these logs are currently done manually.
- **Database Manager:** The Database Manager currently does not make any use of a real database; instead, it stores everything in flat files with fields separated by an ASCII "tab." It provides all the necessary interfaces to store and retrieve any amount of data.

As development and expansion of Oasis continues, missing parts will be implemented to provide the functionality described above in the Function Model section.

We have discussed the functionality of Oasis by breaking it up into seven modules. These modules are used among many threads, which are used to implement Oasis. The total number of threads are given by the following formula:

$$3 + f + n_c + n_d$$

One thread is used to start all other threads and then to monitor the system resources and active clients. One thread is used to wait for a connection at a particular port. One other thread is used by the user interface of Oasis, which currently just displays the logs generated by the Manager Modules. f threads are used as worker threads for doing file I/O. $n_c + n_d$ threads are used for doing network I/O for command and data channels respectively, where f, n_c, and n_d are either the number of processors the machine has or a concurrency level which is chosen by Windows NT.

Current implementation of Oasis consists of a project with more then two dozen C++ and header files that were written with Visual C++ 4.2. This project generates a single executable file and two DLLs. The executable is the core of Oasis. The two DLLs are used as plug-ins to Oasis. One of them is a generic *Internal Abstraction Layer* that enables Module Interfaces to communicate with Module Cores residing on the same machine. The second DLL is a *Database Abstractor* that implements a simple, flat-file database. The use of these DLLs allows for easy expansion to the system without recompilation.

4.4.2 Oasis Monitoring Tool

4.4.2.1 Overview

During normal execution, Oasis keeps a record of many events ranging from logging client activities to monitoring the amount of available system in order to achieve good Quality of Service and to maintain system integrity.

By making these system statistics and information available to Special Clients, we can make tools for Oasis administrators. These special Monitoring Clients can be used to view and alter system configurations, view system status and system activities, and control active sessions.

We will now look at Shade, a special client used to Monitor NPAC's Oasis Media Servers. First we discuss the subsystems Oasis uses to track system events and resources. Then we describe Shade and all of its capabilities.

Disk I/O Monitor (DM)

The DM is responsible for calculating and reporting activities of the Disk I/O subsystem. DM keeps track of the amount of data written or read from the local disk(s) and reports average values at specified time intervals. Only average file I/O performed for sending and receiving Media Object data are logged. This log produced is sent to a requesting Monitoring program where it can be reviewed and even represented graphically. Not logged are file I/O performed for maintenance, logging, and for other system processes.

Whenever Oasis needs to know if a client request can be handled, the DM calculates the amount of bandwidth the File I/O subsystem can handle. This information along with the information returned from other subsystems is used to decide the acceptability of the client's request.

Although DM does not log or report the disk I/O performed by other processes, it must be aware of the entire file I/O produced by the system in order to determine the availability of file I/O bandwidth.

Network I/O Monitor (NM)

Just as the DM is responsible for reporting and logging information about Disk I/O, the NM is responsible for reporting and logging information about the Network I/O of Oasis.

Oasis also requests information from NM regarding the availability of Network Bandwidth. It is possible for NM to find all the Network I/O that is taking place at the system's Network adapter, but presently NM can not know whether the network between the client and Oasis can handle the bandwidth. The best solution for this would be the use of new protocols, such as RSVP, that reserve bandwidth on the network path between the client and the server.

Resource Monitor (RM)

For every client being served, Oasis reserves buffers for both Disk and Network I/O along with buffers for Client Accounting. One of the main jobs of the Resource Monitor is to keep track of and report the Buffer Availability, i.e., the amount of free Memory the system has. For a system that serves real-time data, it is important to distinguish between the amount of free real memory and the amount of free virtual memory. It is the job of RM to report both of these.

Since Special Oasis Clients can upload data to Oasis, Oasis will also need information about the availability of storage space. If Oasis can not reserve storage space for incoming data, the clients request to upload data will be refused.

Media Object Monitor (MOM)

MOM is responsible for answering queries regarding the Media Objects on the system. Oasis will query MOM for availability and status of Media Objects. MOM is responsible also for logging the statistics about Media Objects, such as their frequency of access.

We should note here that just as MOM and other Monitoring subsystems have responsibilities to Oasis, Oasis has responsibilities to the Monitoring Subsystems. For example, Oasis must notify MOM when a Media Object is loaded.

User Monitor (UM)

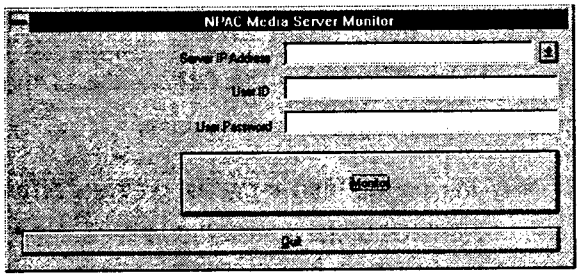
Oasis makes all queries regarding clients to the UM. The UM is responsible for keeping accounting information about the client, querying User Databases for the user's existence and access privileges, preparing data to be retrieved by Accountant Clients, etc. Special Clients called Accountant Clients retrieve accounting Data prepared by UM. Their job is to archive the information logged by UM, allowing Oasis to free up storage resources.

4.4.2.2 Connecting to Oasis as a Monitoring Client

Monitoring Clients such as Shade are one of several types of Special Clients. To have monitoring and controlling access to Oasis, the user must log in with the user name **monitor** or **administrator**. clients who log in as "monitor" only have monitoring privileges; clients who log in as "administrator" have both monitoring and control privileges. These two accounts are local to each Oasis Server.

We use *Monitoring Clients* to refer to clients that can monitor only or clients that can monitor as well as control Oasis. Although clients who log in with "monitor" as their userid can view all information that administrators can, only clients who log in with "administrator" as their userid can make configuration changes to Oasis.

Below we see the login dialog used by Shade. It contains several fields to be filled followed by a button which will use the filled field to connect and log in to an Oasis Media Server.

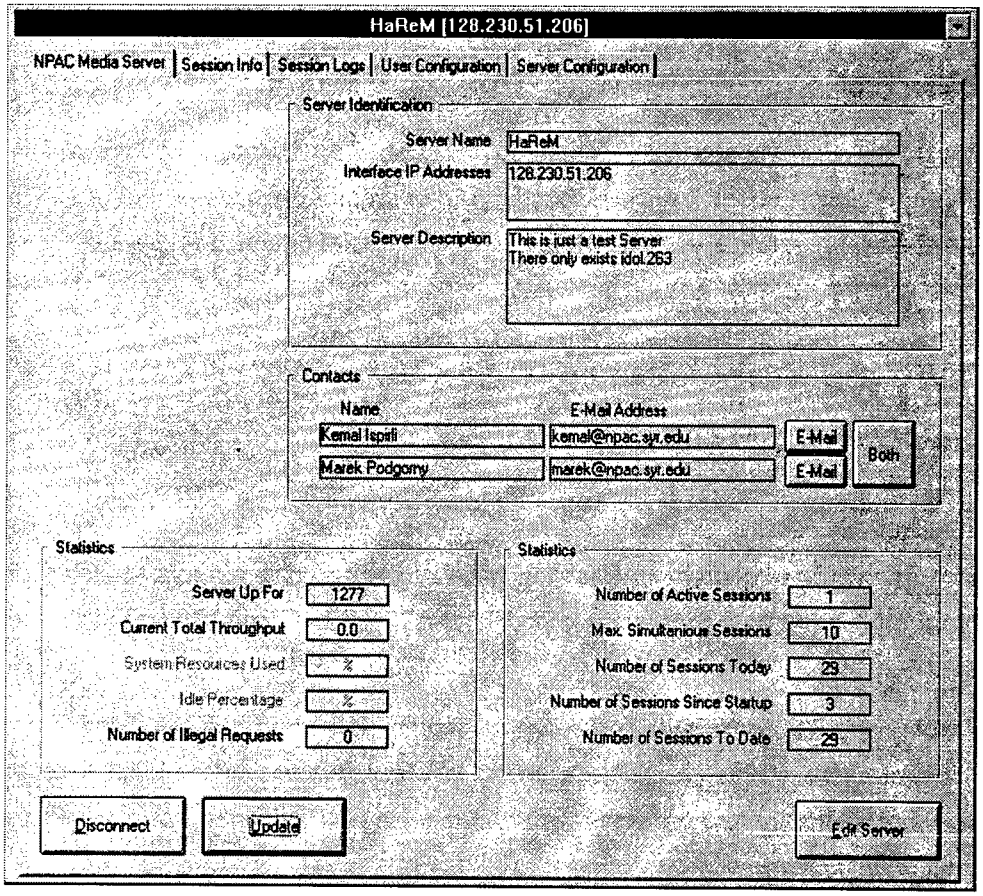


〈 Figure 4-11 Login panel of the Oasis Media Server monitor

One first must specify the IP address and port number of the Oasis connection. Then one must specify a user id and a password, which are setup by Oasis to allow for remote system monitoring.

After proper connection is made to Oasis, Shade presents a tabbed dialog box, as shown below. The user selects tabs to view or edit the server configuration, server statistics, active client

sessions, and to edit special users such as the administrator. Let us now examine these tab dialogs one by one.



〈 Figure 4.12. Main dialog panel of the media server monitoring tool

When the main dialog box appears, the Oasis Info and Statistics page is shown. Within the server identification group, one sees the name of the server, the unique Server Identifier, the IP addresses for all of its Network Interfaces, and the Description of the Server, which contains information regarding the Server's purpose and main contents.

The "Contacts" group contains the names and e-mail addresses of two Server maintainers or administrators. By using one of the three buttons provided, one can send e-mail to one or both of the contacts.

The next group displays a set of general "Statistics" maintained by Oasis. We list these statistics with short descriptions below.

Sets:	Descriptions:
■ **Server Up For:**	This is time in hours:minutes:seconds since Oasis

	was started.
■ **Current Total Throughput:**	This is the sum of the entire average network throughput used for each Media Object that is being served.
■ **System Resources Used:**	This is the percentage of system resources used by Oasis, as reported by Windows NT.
■ **Idle Percentage:**	This is the percentage of time in which Oasis was idle with no Client Connection.
■ **Number of Illegal Requests:**	This is the total number of non-conformance for all clients served since Oasis was started.
■ **Number of Active Sessions:**	This is the current number Sessions that are active. For a definition of a Valid Active Session, refer to Oasis Client Implementation.
■ **Max. Simultaneous Sessions:**	This is the maximum number of simultaneous sessions allowed by Oasis.
■ **Number of Sessions Today:**	This is the number of valid session's established today starting at 12:00 am.
■ **Number of Sessions since Startup:**	This is the number of valid sessions established since Oasis was started.
■ **Number of Sessions to Date:**	This is the number of valid sessions established since Oasis was setup on this system.

For a graphical representation of any of the statistical information, the user can press the button next to the field, which will display a dialog box with a graph as shown below.

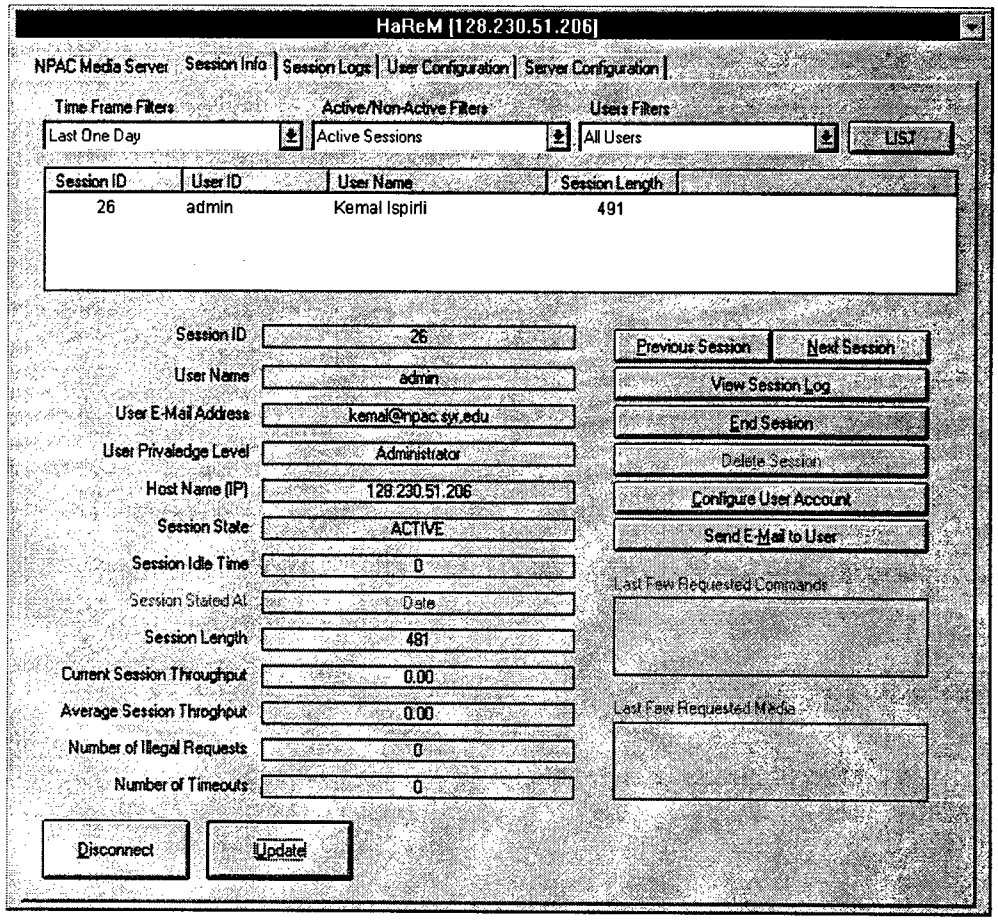
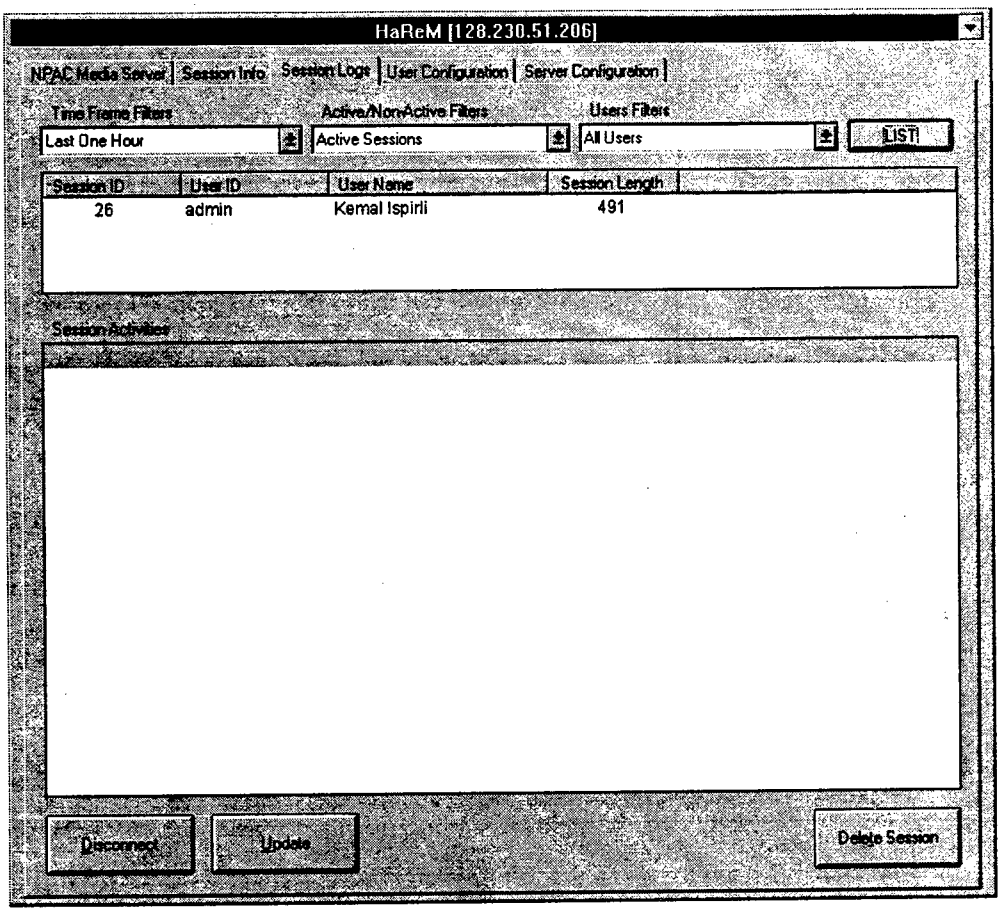


Figure 4.13. Statistics panel of the media server

This tab along with all other tabs can be updated by pressing the **Update** button at the bottom the dialog box. Pressing the **Edit Server** button at the bottom of this page will jump to the Server edit page.

The next tab in Shade's main dialog box will display the "Session Logs" dialog box, which lists three filtering mechanisms on the top of the box. The fist filter allows one to narrow the list of sessions by selecting a time frame, such as This Hour, This Day, This Week, or For all Time. The second filter allows one to view either active, non-active, or all sessions. The third filter allows one select or enter a particular user.

When the **List** button is pressed, the filters are logically combined to produce a list of sessions that appear in the list box. Double clicking on any of the entries in the list box returns information that is displayed in the fields listed below.

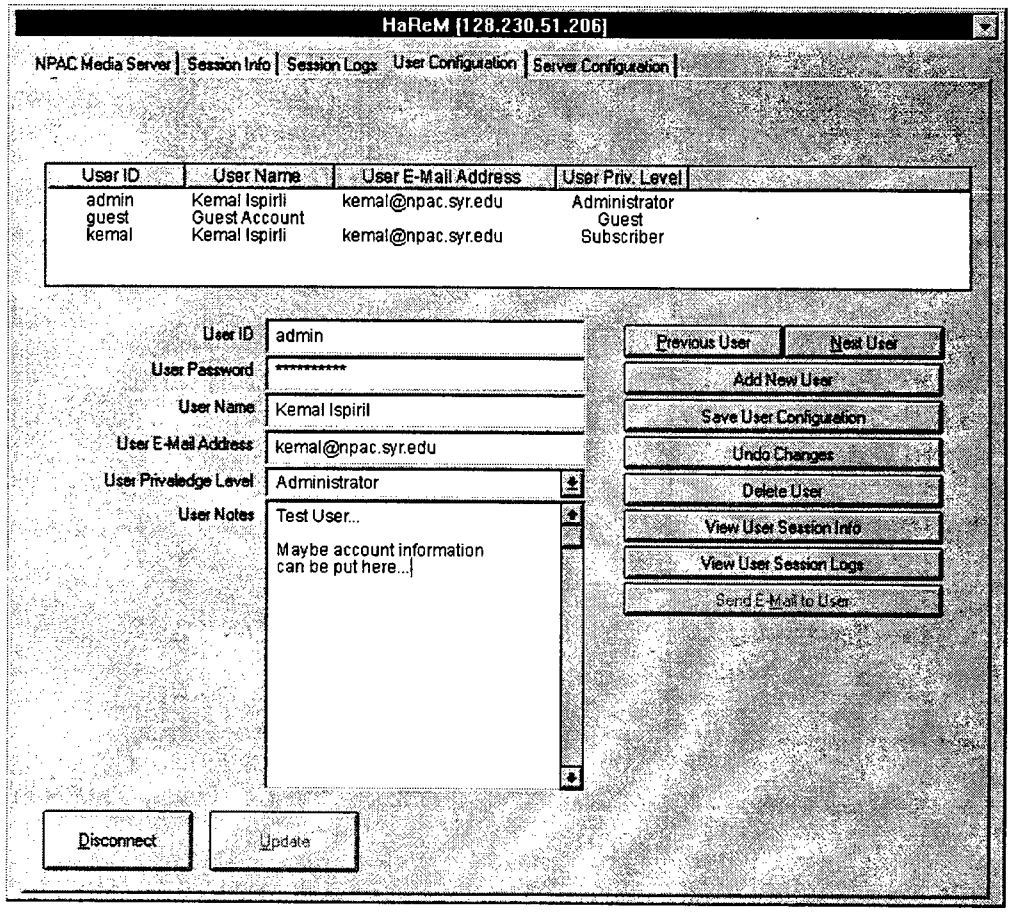


〈 Figure 4.14. Sessions panel of the monitoring tool

Session Attributes	Description
■ **Session ID:**	This is the unique Session Identifier assigned to every valid Session.
■ **User Name:**	The username used to log in to Oasis.
■ **User e-mail Address:**	E-mail address of the user.
■ **User Privileges:**	Privileges of the use as reported by the User Monitor subsystem of Oasis.
■ **Host Name (IP):**	IP address of the machine the client resides.
■ **Session State:**	State of the session, whether the session is active or not.
■ **Session Idle Time:**	The amount of time the Session was idle.

■ **Session Started At:**	Date and time when the Session was started.
■ **Session Length:**	Length of the Session.
■ **Current Session Throughput:**	The amount of network resources used to serve this client.
■ **Average Session Throughput:**	The average network resources used for this client since the start of the session.
■ **Number of Illegal Requests:**	The number of client non-conformances during the session.
■ **Number of Timeouts:**	The number of times the client didn't reply on a timely manner.

Buttons on the right side of dialog box are used to end a session if active, view the sessions log, send e-mail to the Session owner, or browse through the sessions.

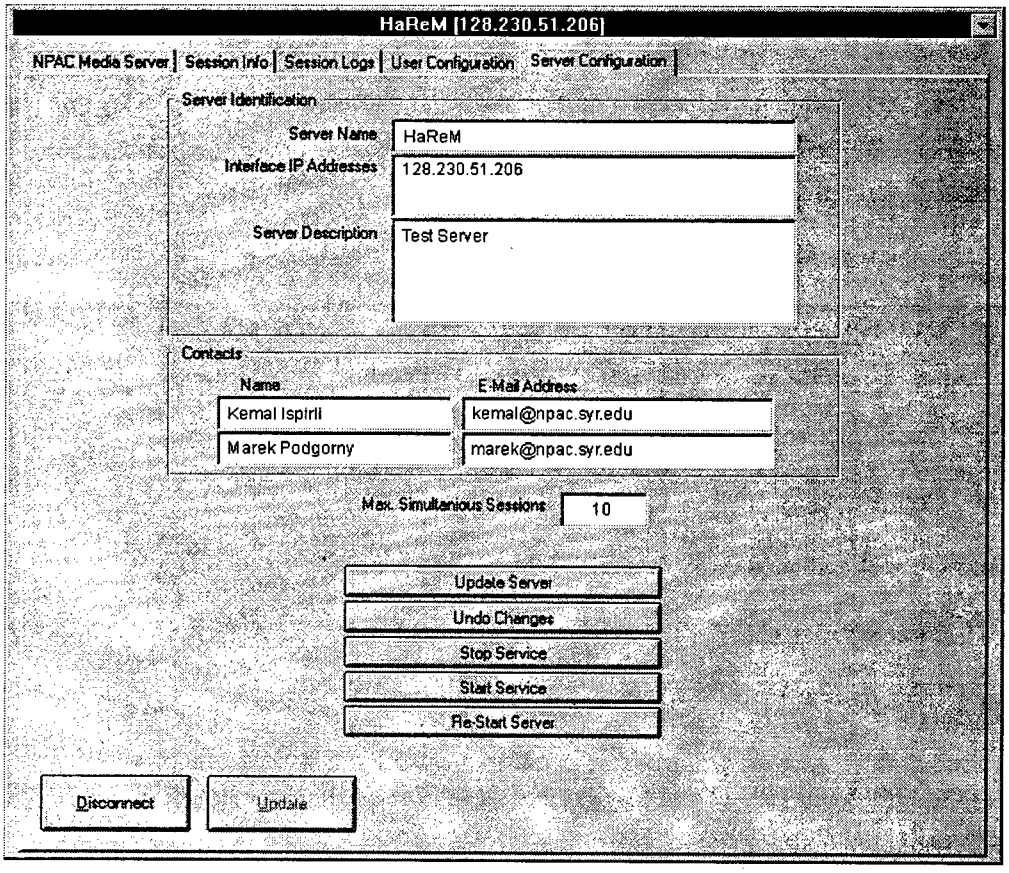


〈 Figure 4.15. User control panel in the monitoring tool

The next tab, "User Configuration," displays a dialog box which enables the editing of users. The dialog box presents a list box containing the names of all local Oasis users and, underneath, a display of the following editable fields:.

User Attributes	Description
■ **User ID:**	Unique user identifier. Identifiers 0 through 1000 are reserved for local users.
■ **User Password:**	Password for the User
■ **User name:**	Name of the User
■ **User e-mail address:**	e-mail address of the user.
■ **User privileges:**	Privileges given to the user.
■ **User Notes:**	Notes about the User.

The buttons on the right hand side of this page are used to browse through the list of users, add new users, save user configurations, delete users, and to send e-mail to the user.



〈 Figure 4.16. Server configuration panel of the media server monitoring tool.

The last tab, "Server Configuration," is currently used only to change simple variables such as the Server Name and e-mail addresses of the Server Administrators.

4.4.3 Client implementation for Oasis Media On Demand Server

This section contains information needed for implementing a client for the Oasis Media On Demand System[9]. The example client implemented in this document is written for the Microsoft Windows NT/95 platforms. We discuss the following aspects of the clients:

- 〈 Network Platform/Protocols
- 〈 Responsibilities of the client
- 〈 Expectations from the server
- 〈 Buffering requirements for the client

[9] After completion of the project, we have learnt about the new protocol designed to transport continuous media. Initially a private submission,. Currently the protocol (RTSP) gained the status of official Internet draft. RTSP is similar to HTTP but supports state. It is our intent to map functionality of the client protocol described below to RTSP.

- Application layer protocol between the Server and the Client and corresponding Data Structures.

This document does not cover the protocol used to query Content Databases.

4.4.3.1 Network Protocol Requirements

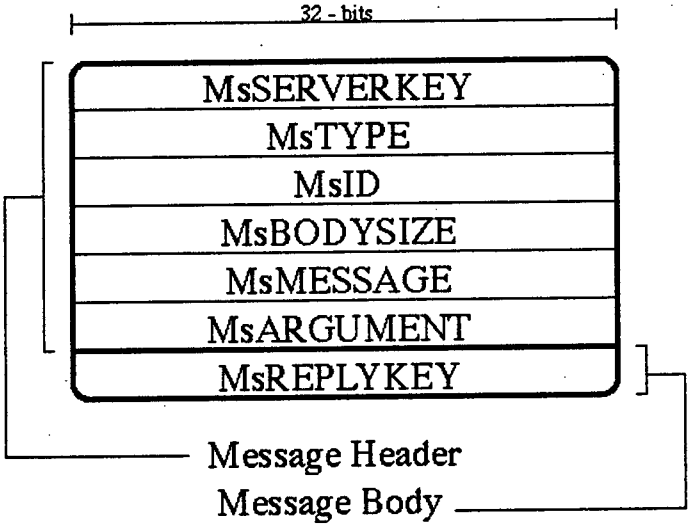
Oasis uses TCP/IP sockets for all its connections to the client. This means that all data and messages sent and received from the client are reliably delivered. Multicast and Unicast UDP-based data connections will be supported in version 2.0.

4.4.3.2 Connection Procedure

To connect and establish a valid session with Oasis, the client must know the IP address and port number at which the Oasis server listens for new clients.

The client should connect to the server with an asynchronous stream socket connection. This will be used mainly for the sending commands to Oasis.

The first command sent to Oasis must be the `MSconnectReply`. This command notifies Oasis that a Reply connection is being requested. The format of `MSconnectReply` is as follows:



For now, let us note that the `MSREPLYKEY` should be a random number used by Oasis to match up a Reply Socket connection with a particular client.

The figure above shows that the `MSconnectReply` message has two parts: message header and message body. Every message (command) sent to or from Oasis will have a Message Header, but it will not always have a Message Body. The parts of this Message Header are described as follows:

`serverKey`: This is a unique number assigned to every server. This "server

ID" is needed with every message. Oasis will sometimes use this to find a valid message within a stream of bytes received from the client.

`type:` This specifies whether the message being sent is a Request or a Reply to a Request.

`id:` This key is used to identify the Message or Message Reply. It should be a unique number generated by the sender of the initial message. That is, if the client wants to send the `MsStop` message, it should generate an internally unique number and set the Message type as being `MsRequest`. Then Oasis will acknowledge back with a Message type `MsReply` with a Message id that is same as the message it just received.

`bodySize:` Some messages will need a Message Body. In such cases, `bodySize` should be set to the exact number of bytes that will follow the Message Header.

message: This is a number signifying a particular message or command.

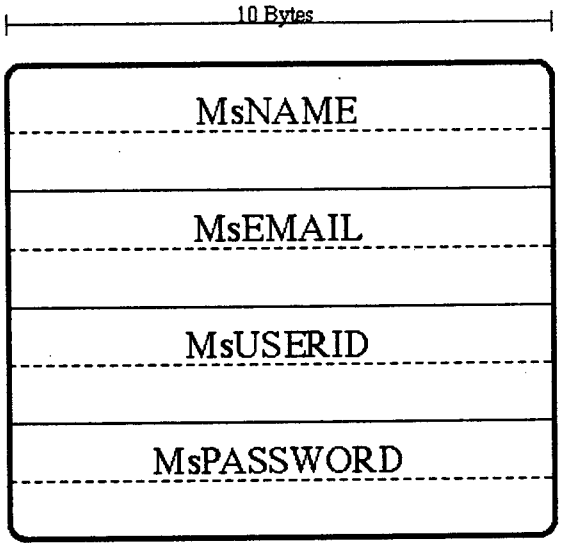
argument: This is a 32-bit number to be used with Messages that do not need to send arguments longer then 32 bits. As an example, if a client wanted to send an `MsSTOP` message, it would also need to identify the Media Channel to stop. The client in this case should place the `MediaConnectionID` in the argument field of the message header. This avoid unnecessary allocation and the freeing of memory for most Messages.

After sending the `MSconnectReply` messages, the client should open a socket connection to Oasis at a port that is one greater (+1) than the port it initially connected to.

After a connection is made at the Reply Port, Oasis will wait for the `MsAssociateReply` Message from the Reply Port. This message should contain the Reply Connection identifier in the argument field of the message header. Oasis will match this identifier with the identifier it had received on the Command Port with the `MsconnectReply` Message. If Oasis finds that there is no client waiting for a Reply Connection with the same RCI, it will drop this connection. If the client fails to connect to the Reply Port with a valid RCI within a predefined time, then Oasis will drop the client completely and free all resources allocated for that client. `MSassociateReply` is the first, last, and the only Message the client should send to Oasis on the Reply Socket.

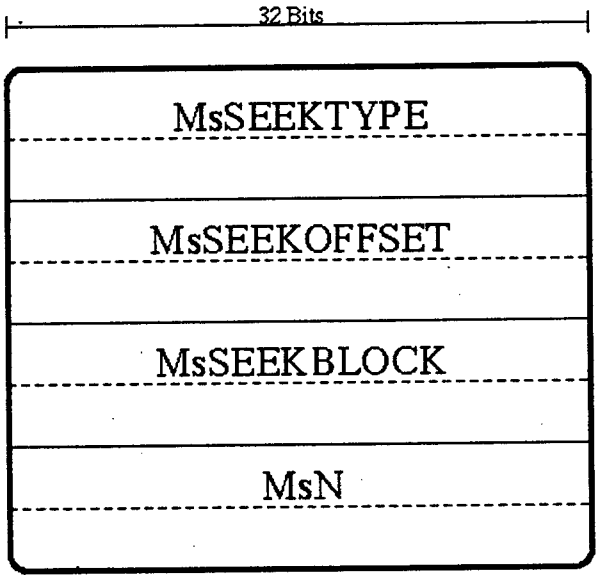
From this point on, the client will send Messages only through the Command Port and will receive messages only from the Reply Port.

To establish a valid Session, the client now must log in. Before logging in, however, the client is permitted to send the following commands:



MSlogin Message Body

The process of logging in is slightly more complicated than sending an `MSconnectReply` message. The client must send a Message Header with the `MSlogin` command, followed by a Message Body with this structure:



MSmcSeek Message Body

The third part of the Message Body is a 20-byte character array containing a NULL terminated string for the client's `Username`. The fourth part, completes the `MSlogin` Message Body with a 20-byte character array containing a NULL terminated string for the client's password.

After the client sends a `MSlogin` message, Oasis will verify the client and acknowledge with a `MSloginReply` containing a Message argument set to either `MSloginBAD` or `MSloginOK` for unverifiable and verifiable login respectively. The Message Body will contain the replica of the Message Body Oasis received with the `MSlogin` message.

If Oasis can not verify the client, it increments the Un-`Conformace` flag of the client. If Un-Conformance reaches a count of five, Oasis will drop the client and free all resources allocated for that client.

A valid session is established when Oasis verifies the client and the `MSloginOK` message is received by the client.

4.4.3.3 Establishing and Closing a Media Channel

Section 4.4.2 of this report explained how to establish a valid session with Oasis. We have described how the opening of two socket connections to Oasis and logging in are the requirements for a valid session. The first socket connection is used to send command and reply messages to Oasis, and the second socket connection is used to receive command and reply messages from Oasis. These two lines of communication are reserved for passing small amounts of control and info data between Oasis and its clients.

To send Media Data, such as video, audio, streamed text, VRML, etc., we use one or more sockets other than the Command and Reply Sockets. We establish a new socket with Oasis for each of the Media Objects we want to access. Currently, Oasis does not support delivery of more than one Media Object at a time on a single Media Channel.

At any time after the client logs in, it can request the status of a particular Media Object. It is desirable for the client to check the status of a Media Object before requesting the loading of it.

Oasis does not allow the client to establish a Media Channel until the client has been verified and has logged in. Only then can the client send Oasis the `MSwantMediaConnection` message. `MSwantMediaConnection` makes use of the Message Body but not the argument field. The Message Body contains the following:

`ip`:	IP address to which the Media Objects should be sent by Oasis.
port:	The port number that the client has bound to and is listening on for a connection from Oasis.

As we can see here, this is an unorthodox way for a client to make a connection to a server; but it is quite versatile since it allows the client to specify any destination for the Media Object, including a Multicast Address. Also, this method allows Oasis to serve the client from anywhere, including from an Oasis Server located on another machine.

When Oasis receives a MSwantMediaConnection, it replies either with a MSmediaConnectionRefused message containing an argument specifying the reason for the refusal or with a MSmediaConnectionAllowed message followed by a Message Body with the following:

ip:	The IP address Oasis will try to establish a connection with.
port:	The port number that Oasis try to establish a connection with.
maxBand:	The maximum bandwidth that can be reserved for this Media Channel at this time.

After sending a MSmediaConnectionAllowed message, Oasis tries to establish a connection with the destination specified by the client. After a new Media Channel is established, Oasis will send a MSmediaConnectionReady message to the client's reply port.

From this point on, the client can start sending Media Control Messages to Oasis through the Command Port. Media Control Messages are discussed in Section Four of this document.

Oasis will free all allocated resources, including Media Channels, at the end of a session; but this should not stop the client from closing Media Channels when they are no longer needed. At some intervals, Oasis will query the client for inactive Media Channels; and if the Media Channel is not claimed by the client as being needed, Oasis then drops the Media Channel and frees all resources allocated for it.

A client wishing to close a Media Channel should send the MScloseMediaConnection message with the Media Channel Identifier in the argument field of the Message Header. From this point on, the client should not send or expect any more data through the specified Media Channel.

Upon receiving a MScloseMediaConnection message, Oasis closes all files associated with the Media Channel, frees all allocated resources such as Network and Disk Buffers, and replies with the MSclosedMediaConnection message to the client. The following sub-subsection discusses Media Control messages used by the client to send or receive streaming or non-streaming data.

4.4.3.4 Media Control Messages

The purpose of this sub-subsection is to describe the messages available to the client for the purposes of controlling data transfer over the Media Channels. The description, here, assumes the existence of a valid session and a live Media Channel, as described in the preceding subsections. The client can use a Media Channel to either send or receive streaming or non-streaming data. In this report, we discuss only the data transfer from Oasis to the client; clients

that upload data to the Server (and thus fall into the category of *Special Clients*) will not be discussed.

We note that, once a live Media Channel is established, it should be used only to transmit data. All messages by the client should be sent and received through the Command and Reply Channels respectively.

Loading of Media Objects

Before any data can be sent over the Media Channel, the client must notify Oasis of the Media Object that is to be associated with the Media Channel. This step is accomplished by the client sending a `MSloadMediaObject` message and waiting until Oasis replies with a `MSmediaObjectLoaded` message.

Oasis will reply to the `MSloadMediaObject` message with one of the following messages:

`MSmediaObjectLoaded`:	Oasis sends this message upon successfully opening the file associated with the Media Object and verifying that the client has the access rights to this Media Object. If Oasis replies with this Message, then a Message body (as discussed below) follows it.
`MSmediaObjectNotFound`:	The specified Media Object was not found. This Message does not have a Message Body.
`MSmediaObjectArchived`:	The specified Media Object is archived. This means that the Media Object is stored in a mass storage system. To access such a Media Object, the client has to make an appointment, at which time the Media Object will be ready for service. This feature of Oasis is not implemented yet.
`MSmediaObjectNoPriv`:	The client does not have privilege to access the specified Media Object.
`MSmediaObjectMissing`:	This message is sent in the case of a discrepancy in the internal content database of Oasis. That is, if Oasis cannot find the file associate with the Media Object even though its internal content database shows that the Media Object exists, Oasis sends this reply.

If Oasis finds that the client has access privileges to this Media Object and if the file associated with the Media Object is successfully opened, then Oasis sends the MSmediaObjectLoaded message with a Message Body that is the same as the MSloadMediaObject message. The difference is that while the client is required only to fill the Media Objects ID, Oasis will fill in the rest of the Message Body if the information it received from the client is different from the information it had; for example, if the client has not filled in the name of the Media Object, then Oasis fills it in when it is replying to the MSloadMediaObject message.

If the client wants to load another Media Object in place of the loaded Media Object, it needs only to call MSloadMediaObject once again with a different Media Object Identifier. Oasis will unload the first Media Object and attempt to load the new Media Object. This means that if Oasis cannot load the new Media Object, the old Media Object will still be unloaded. This is one reason why the client should always use a MSqueryMediaObject message first to check on the status and availability of the Media Object.

Unloading of Media Objects

To unload a Media Object, the client should send MSunloadMediaObject without a Message Body. The client needs only to specify the Media Channel Identifier in the argument field of the Message Header. Oasis will reply with one of the following:

MSmediaObjectUnloaded:	This indicates that the Media Object was successfully unloaded and that all resources, including I/O buffers, have been freed.
MSinvalidMediaPort:	The specified Media Channel was invalid or was not owned by the client.

Oasis will reply back with MSmediaObjectUnloaded regardless of any Media Object being loaded or not, as long the specified Media Channel is valid.

NOTE: If the client only wants to change the current loaded Media Object on a Media Channel, it does not have to call MSunloadMediaObject. See Section 4.1.

Next, we describe the messages that allow for transfer of data over the Media Channel along with messages that affect the transfer of data.

MSmcPlayForward

The main purpose of any server/client architecture is to send data from one node to another. MSmcPlayForward and MSmcPlayBackward are the two messages for which Oasis waits before sending data to the client. For these messages to be successful, the client must have loaded already a Media Object on the specified Media Channel.

These two messages do not have a Message Body. The client needs only to specify the Media Channel that this message will affect in the argument field of the message. If MSmcPlayForward is used, Oasis will increment the file offset by *n* bytes each time it sends *n* byte chuck of data. If MSmcPlayBackward is used, Oasis will decrement the file offset by *n* bytes each time it sends *n* byte chuck of data.

If Oasis can send data, it will reply to a MSmcPlayForward or MSmcPlayBackward with a MSmcPlayingForward and MSmcPlayingBackward respectively and start sending the data at the rate requested. If Oasis cannot send the data, it will reply with one of the following:

MSmcCantPlayEoF:	If the file offset is already at the end of the file when MSmcPlayForward is sent, Oasis returns this Message.
MSmcCantPlayBoF:	If the file offset is already at the beginning of the file when MSmcPlayBackward is sent, Oasis returns this Message.
MSmcCantPlayInvalidMC:	If the argument field of the Message Header is not a valid or an existing Media Channel Identifier, Oasis replies with this message.
MSmcCantPlayNoMO:	If the valid Media Channel does not have a Media Object loaded, Oasis replies with this message.
MSmcCantPlayNoResource:	If for some reason Oasis has a limited resource and cannot deliver the data at the specified rate, then this message is sent. In this case, the client can either adjust the rate or wait until more resources are available.

MSmcStop

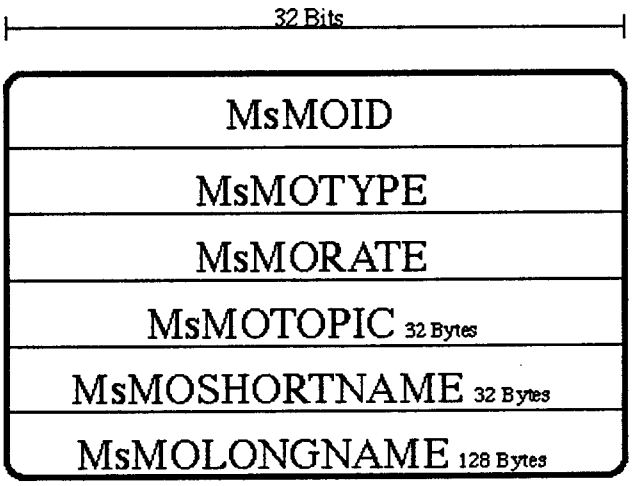
The client can request the data transfer to stop at any time with the MSmcStop Message. Under normal circumstances, Oasis will acknowledge a MSmcStop message with MSmcStoped even if the state of the Media Object was already stopped. In the case of an exception, Oasis might reply with the following:

MSmcCantStopInvalidMC:	If the argument field of the Message Header is not a valid or an existing Media Channel Identifier, Oasis replies with this message.
MSmcCantStopNoMO:	If the valid Media Channel does not have a Media Object loaded, Oasis replies with this message.

Note: The client should be aware of the fact that, if Oasis stops sending data, there still could be data available to read at the specified Media Channel. There could be data remaining at the Network Buffers, which might need to be flushed.

`MSmcSeek`

Oasis allows clients to seek anywhere within the Media Object, allowing random access capability. At any time while a Media Object is loaded, the client can send the `MSmcSeek` Message to Oasis. Just as with all the Media Channel Control messages, the argument field of the `MSmcSeek` message header should contain the Media Channel Identifier. Oasis allows several types of seek; and, since `MSmcSeek` has multiple options, Seek Messages may have a Message Body. The following figure shows all the fields that make up the `MSmcSeek` Message Body.



MSloadMediaObject Message Body

The following describes all fields within the body of the `MSmcSeek` message:

type:	The type of seek to perform. The following are valid seek types:
`MSseekOffset`:	This tells Oasis to seek to the offset specified in the *n* field.
`MSseekBlock`:	This tells Oasis to seek to the *n*th block in the Media Object. Size of the block can be set with the `MSmcSetProperties` Message described in the next sub-section. Block 0 indicates the beginning of the file.
n:	This field is used by Oasis to seek to an offset or a block depending on the seek type that was specified. If *type* is `MSseekOffset`, then *n* is interpreted as a file offset. If *type* is `MSseekBlock` then *n* is interpreted as the block number and

Oasis will seek to file offset *n*`*blocksize`.

`MSmcSetProperties`

The `MsmcSetProperties` command can be sent by either Oasis or the client, allowing both to set preferences regarding the Media Object and the Media Channel, such as the maximum bandwidth supported by the Media Channel. The message body for `MsmcSetProperties` is the same as the message body for `MsloadMediaObject`. The properties will be set to the minimum preference of the client and Oasis. For example, if the client is requesting 1.8 Mb/s, but Oasis can only send at 1.5 Mb/s, then the lesser of the two requests (1.5 Mb/s) is chosen.

4.4.4 Implementation status of the distributed video server

In the time frame of this project, most of the server software was implemented. Limitations of the server implementation are listed in subsection 4.4.1.11. As for global system implementation, no operational video clients have been ported to the server to work with the Oasis server. Also, there was no performance testing.

In the light of recent developments in the practical implementation of the Integrated Services Model for the Internet, we believe that the Oasis server design should be modified to comply with the protocols used in the model. This modification would include use of UDP for media streams, RTP for multiple media stream synchronization, support for multicast, and explicit support for Reservation Protocol.

5. Video Client Architectures

Video client implementation was one of the main thrusts of the entire project. We searched for a software environment that would allow us to write portable video applications. Such a framework simply does not exist today. Video applications depend greatly on specialized video hardware. And, although there is a strong movement towards standardization of many elements, such as video codecs and multimedia protocols, the actual implementation tools for networked video applications have no cross-platform common denominator. For instance, the fact that the structure of the MPEG1 system stream is strictly defined does not imply that there are commonalties in the way the stream is presented to MPEG decoders on different platforms. It is important to understand that compliance with certain standards does not imply much for cross-platform multimedia development. Consider, as an example, a family of standards such as H.320 or T.12X. These protocols ensure application interoperability. The H.32X family declares an application compliant if it can handle an H.261 video stream. Yet such an application, when implemented on a PC and a UNIX platform, will use vastly different programming environments. Another, perhaps more suggestive example, is the situation with hardware MPEG decoding cards. Microsoft defined a programming environment known as Video for Windows (VfW). This environment (described in detail below, 5.2) ensures that the drivers for such third-party cards will respond to a set of standardized commands, known as Media Control Interface (MCI). Unfortunately, this command set is of such a high level that it does not deal with the critical issue of a standardized interface to video encoders and decoders. Since VfW does not deal with network video at all and does not support video playback from a user space buffer, the net result is that even with this environment in place it is necessary to obtain an independent SDK from each and every MPEG decoder card vendor to be able to implement a video client. This situation almost led to a revolt among developers of video game software. We have faced the same difficulty during the entire project.

Historically, when the project started, there were no machines among PCs and desktop class workstations able to play a smooth MPEG1 video stream. Further, for most UNIX brands, there was no meaningful system software support to write network video applications. The notable exception was Silicon Graphics, and we initially. concentrated on this platform. Later, we moved towards PC platforms, mostly because it was economically unfeasible to expect that we would be able to install video laboratories for the Living School Book project or even at Syracuse University, with dozens of video clients using SGI workstations. As the performance of PC platforms was too low initially to support software video playback, we concentrated on the hardware solutions. We used the VfW environment to implement a high-end networked video client using the hardware MPEG1 decoder from OptiBase. The project establishes the feasibility of such a solution, but this

solution was limited to just one hardware board. Lack of the standardized hardware drivers rendered this work a research curiosity. We describe this effort in detail in Subsection 5.2.

During the final stage of this project, Microsoft introduced a new technology, code-named ActiveMovie. We gained early access to the technology, by becoming a beta-tester site for this product. ActiveMovie goes a long way toward providing a uniform interface to video coders. Prior to the end of the project and before ActiveMovie was officially released to the public, we implemented a completely new, software-based video client for our systems. This software can run on any decent PC and is being currently deployed in the Living School Book testbeds. Although we have not achieved portability across PC and UNIX platforms, we believe that we have provided a viable video client solution for inexpensive educational installations, mostly using PC workstation. Details of this work are described in Subsection 5.3

The remaining challenge of "grand unification" will be resolved soon. This will happen by completely switching multimedia development to Java. In section 5.4 we describe our early Java implementation of the H.263 video client. This software is entirely cross-platform. Given the rapid maturation of Java technology, we predict that this language will be used for 90% of all multimedia development two years from now.

5.1 Video client for SGI platforms

We implemented both MPEG1 and H.263 video clients for SGI workstations. Both modules were implemented using SGI Digital Media Library.

5.1.1 Digital Media Library

Most of our UNIX work in the project has been done on SGI platforms. SGI offers a relatively coherent development environment for digital media. This environment is encapsulated in the so-called Digital Media Library (hereafter referred to as DML).

The overall design structure of the Digital Media Library is an interesting case as it represents one of the advanced industrial attempts to provide a comprehensive API to the multimedia functionality of a UNIX workstation. The entire library consists of 4 major parts: audio library, video library, compression library, and movie library. Audio and video libraries provide an API to the proprietary SGI audio and video hardware functions, like device initialization, control of audio and video channels on the dedicated devices, etc. These two parts are, per definition, not portable and can only be used on SGI workstations. The compression library is a collection of algorithms for both audio and video compression and for providing access to the memory structures used to store and forward the audio and video streams. This part of the library is crucial for our work since it enables us to implement a network delivery framework. Finally, the movie library provides mechanisms for audio/video synchronization and for the random access to the audio/video ("movie") material.

On the first glance, DGI's DML appears to be a complete and well-designed framework.. The bast parts of the library are the audio and video modules. Handling of multimedia hardware on SGIs is much more robust, stable, and straightforward compared to the PC platforms. The Audio library provides support for full duplex transmissions and for multiple concurrent streams. The video library allows for rather easy access to the digitized video, while the digitization functionality is almost completely hidden from the programmer. Audio library supports, among other things, the following basic functionality:

- configuration of audio hardware and management of audio-related I/O between the applications and hardware
- support for many popular file formats, such as AIFF/AIFFC, .au, .wav, etc.
- automated creation of the audio library data structures
- support for CD players and DATs as audio source, as well as full support for MIDI

The Video library is a collection of device-independent C calls working on top of the SGI video device drivers. The most relevant functionality includes

- support for video display on the computer screen
- support for video input and digitization devices
- support for blending different video streams.

Note that the Video Library does not deal with video compression or even with different digitized video representations. The basic functionality of the library deals with getting video streams in and playing them out. Anything in between must be handled by the application. To aid developers, SGI provides a compression library and a movie library.

Unfortunately, the good initial impression quickly shatters upon closer inspection. The problem with the current version of the DML is that it was implemented by four separate teams of programmers with little coordination. As a result, the data structures used in different parts are incompatible, and the actual information flow along the audio/video -- compression -- movie path is not really supported. One could expect that the movie library will be able to access the decompressed audio/video streams, synchronize them, and send them to a display device. Conversely, the compression library should be able to receive a request from the movie library to deliver a particular part of the compressed audio/video and serve it. This model is clearly used within the SGI application software but the API and data structures providing this functionality is not published. As a result, there is a discontinuity between the compression and movie library: it is possible to display either audio or video streams directly from the compression library, but there is no support for streams synchronization. The movie library internally supports synchronization but is unable to read from the data structures supported by the compression library. As a result, the

crucial functionality needed to implement network video is not only not supported, but actually carefully hidden.

The movie library requests a file descriptor as input. We have attempted to bridge the gap between the compression and movie libraries by using FIFOs (named pipes). This attempt failed: while the FIFO file descriptor is accepted as input, the movie library does not support the additional parameters necessary to handle dataflow in the pipe.

We were able to confirm our assessment in a conversation with the SGI developers, who admitted conceptual fractures in the design of the DML. According to SGI, a major effort is underway to rewrite the library. We will receive incremental upgrades to the software as the rewrite process progresses. But it was not until the end of the project that we received a version of the DML that would support explicitly network video.

5.1.2 MPEG1 Video Client for SGI workstations

To ensure project progress, we implemented a client-server architecture supporting network delivery of the audio/video streams from the server to a client operating on the level of the compression library. We designed and implemented the framework that moves data from the network buffers to the user buffer space, demultiplexes the audio/video stream, feeds the streams to appropriate decoders, collects the output, and submits it to the audio and video library routines for presentation. This process does not use the high-level movie library at all.

The entire architecture of the SGI video client revolves around the Compression Library (CL). SGI's CL supports the following relevant functionality:

- it supports quite flexible and extensible software interface for compressors and decompressors of the audio, video, and still-image data;
- for video, CL supports MPEG1 and a few of SGI proprietary compression schemes (these schemes are of little relevance as they are neither standard not particularly advanced);

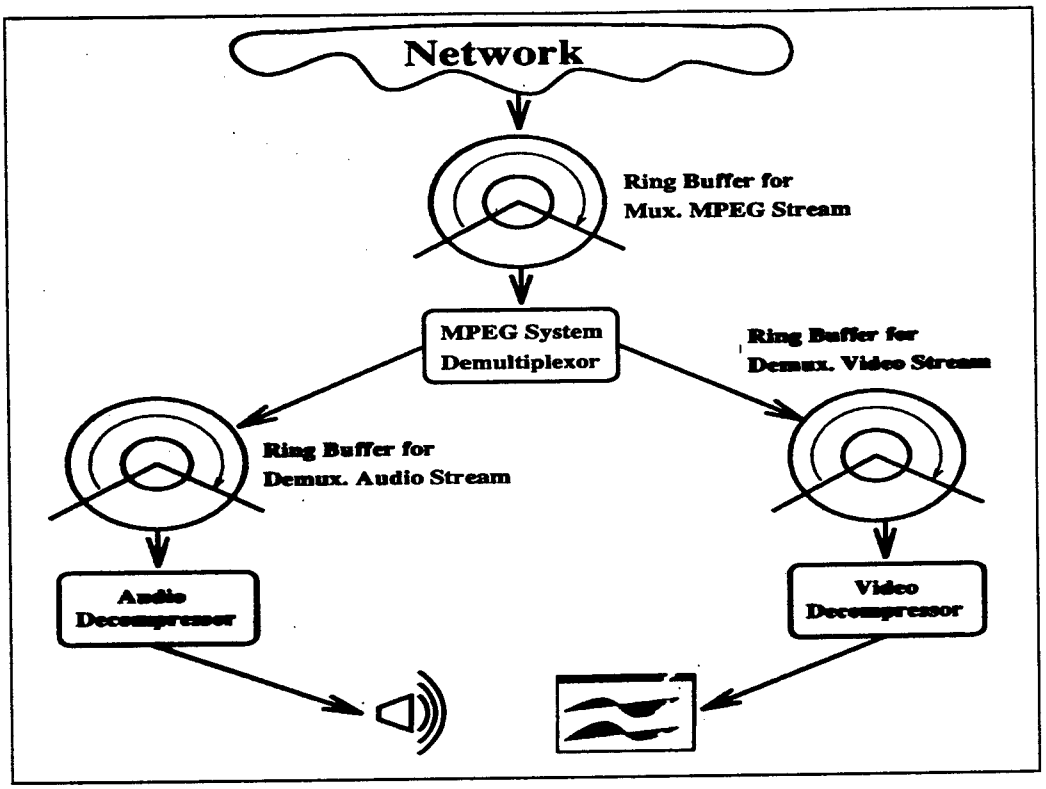


Figure 5.1. Architecture of the Video Client for SGI workstations.

- it allows installation of new compression algorithms;
- it is binary compatible across all SGI platforms.

The figure above illustrates the high-level architecture of the SGI video client

The video client is multi-threaded. Technically, multi-threading is implemented using the `sproc` mechanism described in Subsection 4.1. The four threads of the video client serve the following functions: network I/O handler, Audio and Video decompression handlers, and User Interface handler. The client uses compression, audio, and video libraries from the DML and the IRIS GL library to handle screen display. The client is capable of random access.

The network I/O thread of the client reads blocks of the MPEG1 system stream from the network interface and places them into a ring buffer. Ring buffers are used throughout the client to ensure full asynchronicity. The ring buffer concept is different than double buffering. It is one contiguous segment of memory which is wrapped around. The producer process writes at the buffer tail pointer and moves the pointer to the first free memory location in the buffer. The consumer process reads data starting at the head pointer and updates it to the location of the first unread memory locations. Pointer arithmetic is encapsulated in the ring buffer data structures and is transparent entirely to the producer and consumer processes.

The data from the network ring buffer is demultiplexed into separate audio and video streams (MPEG1 system streams interleave audio and video). The demux process is implemented using CL support. Then both media streams are written in two other ring buffers. These buffers are read asynchronously by the audio and video decompression threads.

The audio and video decompression threads are similar. Both read the circular buffers into respective decompressors provided by CL. The decompressed data is submitted to either an audio port (using Audio Library) or to the GL routines for screen display. If hardware accelerators are available, they will be used automatically.



Figure 5.2. SGI MPEG1 video client

The GUI thread of the video client is built using Motif and SGI widgets. This interface (depicted below) supports VCR-like controls, a scrollbar to control random access to the stream, and a text area to display close captions. In addition, the interface supports so called "video marks" to facilitate marking video segments of interest. The GUI thread communicates with all other video client threads using signals.

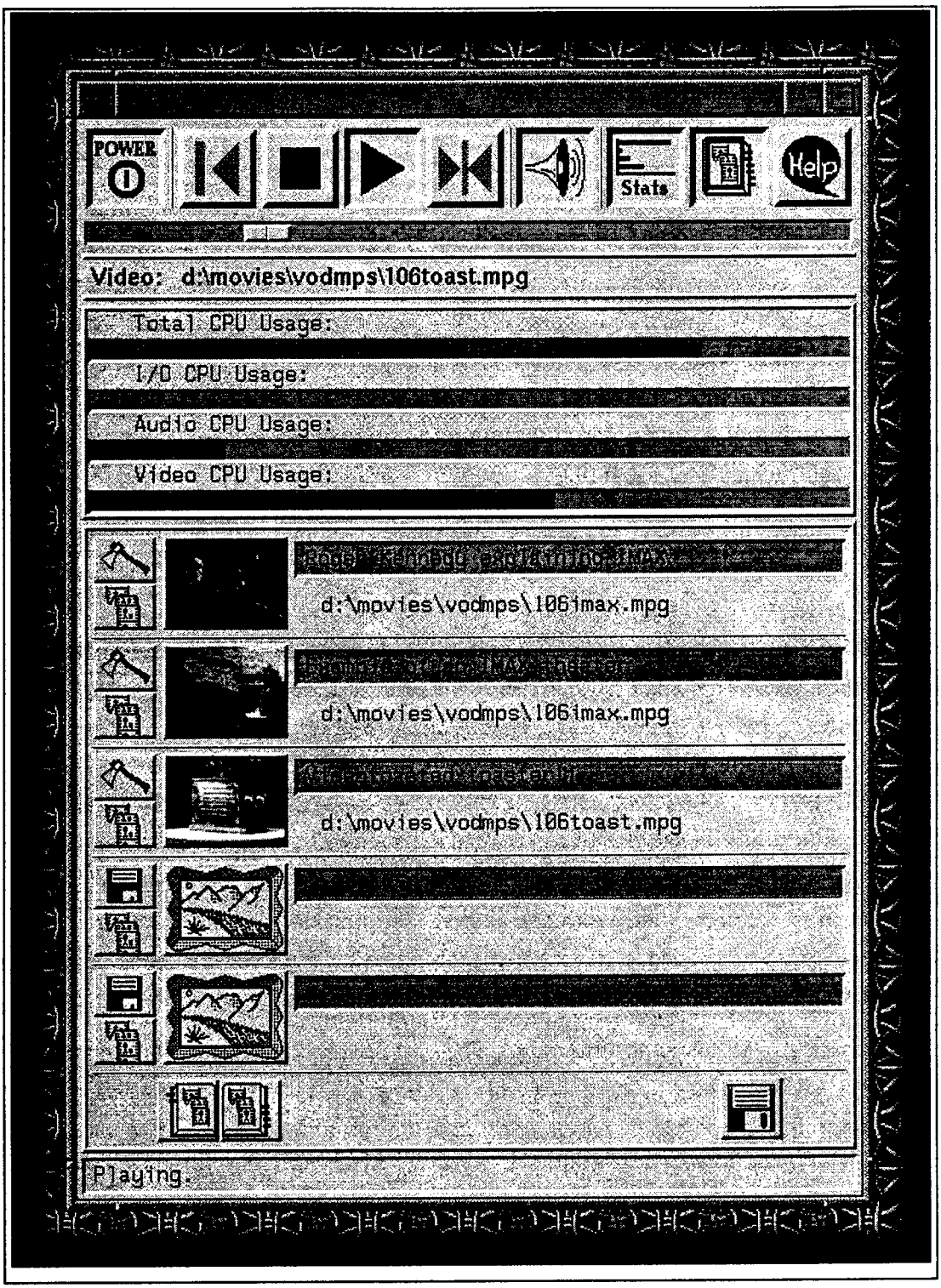


Figure 5.3. SGI video client: performance monitoring panel and video marks panel.

Since we could not use the Movie Library, we needed to resolve the audio/video synchronization problem. For MPEG1. The audio/video stream is interleaved. Thus, in principle, it should not be

difficult to achieve synchronous playback. Two situations arise that require attention: first, there may be insufficient network bandwidth. For non-adaptive applications playing interleaved streams, there nothing can be done to remedy this problem. Insufficient bandwidth starves the decoder, which stops playing. Whether audio and video are synchronized during intermittent playback is of little relevance - the stream is unusable. However, with sufficient bandwidth, we often have a problem of insufficient CPU power to decode the video stream. Two solutions are possible: one is to stop playing audio and play all frames of the video stream in a kind of slow motion, the other is to skip portions of the video stream to keep up with the average bitrate. In systems which push the data at a constant bit rate, as our server, only the second solution is acceptable. Thus, there is no problem of audio/video sync, but rather a problem of gracefully degrading video playback by smoothly skipping video frames as needed when the CPU falls behind. Now, this problem is resolved (to a degree) in the native SGI applications using Movie Library; but it is a difficult to solve using the tools provided in DML.

We determined that the SGI software MPEG decoder performs quite poorly. Clearly, it has not been optimized. As a result, an average SGI workstation (say, a 150-MHz MIPS 4400 Indy) is unable to play MPEG video streams at full bitrate. We discovered that, if only video is being handled, CPU utilization tends to saturate, at which point the DML MEG1 decompressor simply fails (i.e., it displays garbage). Now, to relieve the CPU load, we devised an algorithm that skips B or PB frames when falling behind. Implementation is rather difficult since the CL does not provide any tools to recover presentation time stamps from the MPEG1 system stream. However, the main stumbling block was the fact that the CL routines, when asked to skip frames and go to an I frame ahead in the stream, do not return promptly! As a result, instead of gaining ground on the incoming video stream, the CPU utilization drops to ~50% yet the control does not return to the application - it is like the MPEG decoder, asked to skip a frame, sleeps internally for the time that would otherwise be needed to play the frame. As a result, the decoding process does not use all available CPU time while maintaining poor visual performance.

We were unable to correct this behavior.[10] The best playback quality we managed to achieve is marginal, at best. The playback quality of our player is slightly worse than the quality of the indigenous SGI movie player (which, *nota bene*, is not a networked application). Apparently, access to a non-published API helped SGI developers to write a slightly better implementation, but even this implementation provides mediocre MPEG1 playback.

Another annoying problem is the inability of the Audio Library to support random stream access. Basically, in spite of the manuals stating otherwise, AL dumps the core when presented with an MPEG1 audio stream that is off by as much as 1 bit. We achieved a functional but less than

[10] The most recent version of the DML (coming IRIX 6.3) simply removed the capability to skip frames from the library, i.e., SGI developers made a bad situation worse.

robust implementation by keeping the initial audio header in memory and trying to localize a valid entry point to the audio stream corresponding to the Group of Pictures (GOP) mark.

In conclusion, the networked video client for the SGI workstation was implemented, but its performance and robustness are not satisfactory. As outlined above, a way around the difficulties is to implement a high-performance MPEG1 decoder from scratch. The time frame and level of funding of this project precluded such a solution.

5.1.3 Low bit rate, H.263 video client for SGI workstations

In the case of the low bit-rate video client, we implemented all the decoders from scratch. We still used the DML framework by adding the H.263 decoder to the library, following the compatibility rules defined by SGI. Because we had full control over the decoder source, we did not experience performance problems as those described above for the MPEG1 decoder. The H.263 player handles video streams with bit rates in the range of 28.8 -- 384 kbps. Because H.263 is a video-only standard, we modified the video server to serve two synchronous streams (the problem of multiple synchronicity has been addressed in a more general fashion in the Oasis server). The low bit rate H.263 video client recognizes audio streams coded in either GSM 6.1 or ADPCM format. The client supports QCIF and CIF file formats, and it enables one to play the movie, to stop it, and to randomly access the stream.

For a description of the H.263 codec, please refer to Appendix 9.2.

GSM 06.10 audio codec

GSM is a telephony standard defined by the European Telecommunications Standards Institute (ETSI). The GSM 06.10 compressor models the human-speech system with two digital filters and an initial excitation. The linear-predictive short-term filter, which is the first stage of compression and the last during decompression, assumes the role of the vocal and nasal tract. It is excited by the output of a long-term predictive (LTP) filter that turns its input -- the residual pulse excitation (RPE) -- into a mixture of glottal wave and voiceless noise. The GSM encoder compress 160 16-bit voice samples into 264-bit GSM frame. GSM 06.10 is faster than code-book lookup algorithms such as CELP. It offers 13kbps bandwidth.

Intel/DVI ADPCM

The ADPCM compression algorithm uses the correlation between adjacent audio samples to reduce bit rate. It transmits only the differences between samples and their predicted values, which have less a dynamic range than the samples themselves. Predictor coefficients and reconstruction levels are calculated dynamically using coded signals. This enables a reduction of the bandwidth but make this adaptation technique more susceptible to transmission errors. There

are a few ADPCM standards, e.g. Intel/DVI and G.721. Intel/DVI provides good quality, even for music, even though it is not computationally intensive.

Synchronization

Synchronization is based on the internal SGI Audio Library mechanisms. Procedure sending audio samples to the speaker port blocks until all previously sent samples are processed. It is therefore enough to send audio portion and then decode video frame to achieve the synchronization and keep the frame rate. The only problem is that this mechanism requires single video frame decoding which moves the responsibility of single frame reading from the decoder to the AV client.

To support this sync mechanism, the following changes/enhancements were necessary both at the decoder and client side:

- the original H.263 decoder was transformed into a library that allows the decoder to initialize and then decode single frames passed as a function argument;
- a special function for detecting ends of H.263 frames was implemented and used in the single-frame reading procedure;
- a system of ring buffers was implemented to make independent the process of taking a single H.263 frame from data being read from a socket.

Random access

Since audio portions are always the same size, there is no problem finding the beginning of an audio segment. However, the situation differs in H.263 video compression. First, H.263 frames have different lengths. Second, because H.263 is a predictive encoder, only INTRA frames can be accessed this way. Thus, a way to offset a H.263 stream is needed.

The H.263 system is intended as a preview tool for MPEG movies. Hence, H.263 sequences are created by converting MPEG files. It is rather important for a preview tool to have random stream access. This requires the generation of offset files.

To obtain offset files, it was necessary to add new features to the H.263 encoder:

- the ability to encode an INTRA frame every fixed number of frames (only INTRA frames can be accessed randomly);
- the ability to write an offset file with the number and stream position of every INTRA frame.

All these modifications have been implemented. The offset files are created by the MPEG1- H.263 re-encoding tool.

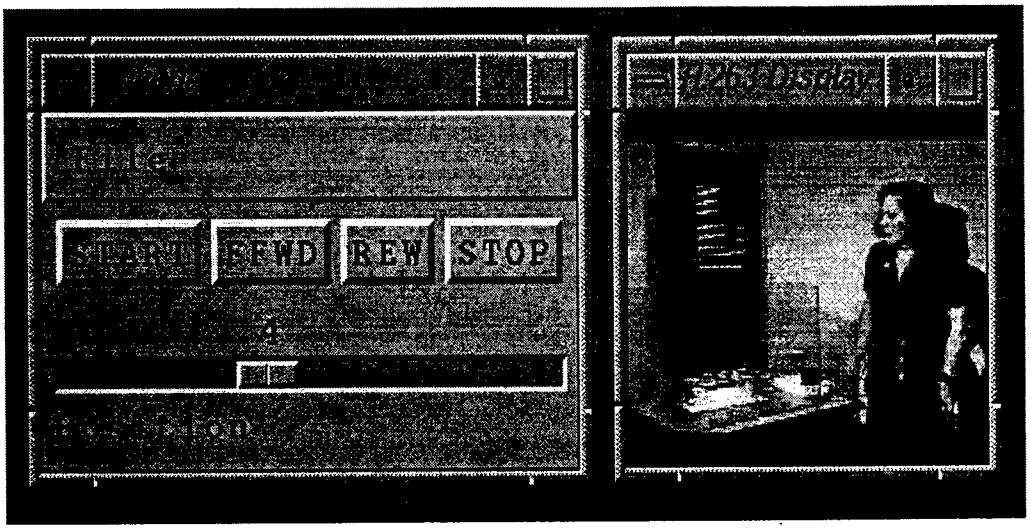


Figure 5.3. SGI H.263 client playing a video clip in the Museum of Science and Technology in Syracuse

5.2 Video for Windows networked video client

This section describes the architecture and implementation of the video client developed for the Windows 3.1 environment. As discussed above, it was not possible to achieve acceptable quality MPEG1 playback using software decoders until very recently. Now, however, we have started implementing video clients for PC machines using hardware MPEG1 decoders. The software development framework was provided by so called Video for Windows (VfW) architecture. The following section briefly describes this framework then details the design and implementation of the video client using the Optibase MPEG1 hardware decoder.

Although the Window 3.1 OS is somewhat obsolete, we continue to use the Optibase video client in NPAC because the playback quality is superior.

5.2.1 Video for Windows architecture

The Microsoft Video for Windows (VfW) architecture was designed to provide developers with services for developing video capture, editing, and playback applications. It contains a set of application programming interfaces (APIs) necessary to create these types of applications, as well as audio and video compression and decompression (codec) drivers and video-capture drivers. The VfW development kit includes C window classes, Visual Basic programming system custom controls, sample code, debugging tools, and the VidCap and VidEdit video capture and editing tools.

The VfW DK is designed for developers who are implementing

- end-user tools for video processing,
- device drivers for audio and video data capture and compression, and

- content applications that incorporate video sequences.

The VfW architecture provides the following components:

- **A data container**. The VfW uses the Audio Video Interleaved(AVI) file format for applications that capture, edit, and play back audio-video sequences. In general, AVI files contain multiple streams of different types of data, though multiple streams of any data type are supported. A simple AVI file might contain an audio voice stream, a MIDI stream, and a video stream. A file containing two audio streams, each in a different language, is an example of a single file that serves a dual purpose.
- **A set of APIs**. Other functionality that might be included in an application is provided by APIs that enable the manipulation AVI files, that create and work with audio and video codecs and capture device drivers, and that communicate with Media Control Interface (MCI) devices.
- **Predefined classes and controls**. The VfW architecture provides C window classes and Visual Basic custom controls for creating capture and playback applications. For example, the functions in the `AVICap` window class provide features that appear in the `VidCap` tool. The `MCIWnd` window class functions provide features that appear in the playback window of the `VidEdit` tool. A subset of the features provided by the `AVICap` and `MCIWnd` classes appears in their related Visual Basic controls, CAPWNDX.VBX and MCIWNDX.VBX.

VfW architecture is designed to be extensible. Developers can expand VfW run-time capabilities by adding their own custom stream and file handlers that incorporate new types of data into the AVI file format. Audio and video codecs can be added to provide additional options for software compression and decompression. New capture device drivers can be incorporated to take advantage of advances in hardware development.

The main drawback of this architecture is its lack of modularity. Although it is possible to add an entire new codec to the framework, the codec itself has to handle both the data source and the actual decoding process. Technically, what is missing in the VfW architecture is the *codec handle*: it is not possible to simply call a decoder on behalf of the data in a user space buffer. Consequently, there is no simple way to expand the VfW architecture to networked video. In our case, two project requirements were (1) to play video over the network, and not from local disk drive, and (2) to use a hardware MPEG1 decoder. Since an installable codec in the VfW framework must handle both data source and the decoder, there was very little we could reuse from the VfW except for the GUI of the popular Media Player. The Optibase MPEG1 decoder card did not have an MCI (i.e. VfW compliant) driver until late 1995; but, even if it had, there would

have been no way to integrate low level data handling and high level MCI (Media Control Interface) within the VfW framework. We have implemented the entire installable driver for the Optibase card using three software ingredients: (1) our own code for the client network interface, MCI commands handlers, and decoder control, (2) Optibase development kit to submit MPEG data to the decoder, and (3) Microsoft Media Player GUI to control the client.

5.2.2 MPEG decoders

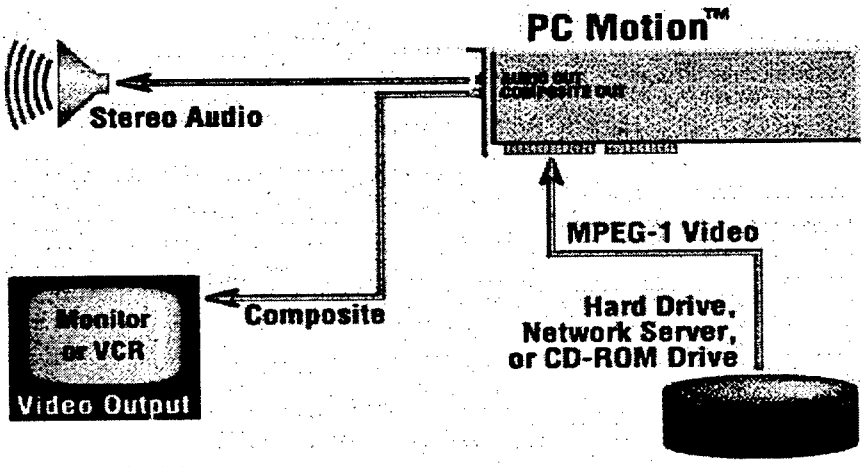
MPEG (Moving Pictures Experts Group) is a group that meets under the leadership of the International Standards Organization to generate standards for digital video (sequences of images in time) and audio compression. The MPEG standard defines a compressed bit stream, which implicitly defines a decompressor. Implementation of the compression algorithm itself is up to the individual manufacturer, and that is where proprietary advantage is achieved within the scope of a publicly available international standard. MPEG itself is a nickname. The official name of the standard is ISO/IEC JTC1 SC29 WG11.

The hardware decoder used for our Video Client is a PCMotion board manufactured by Optibase, Inc.

This decoder was compatible with

- the ISOMPEG1 standard (the world standard for digital compression),
- industry-standard PC/AT computers, and
- the MPEG1 system layer files.

The PCMotion board is an ISA card inserted into the PC workstation. The following illustration shows hardware elements interacting with the board.



The decompressor is able to decode a MPEG1 system layer file in real-time at 30 frames per second. The board could also encode and decode MPEG1 audio layer files. The PCMotion card

produces either composite or Y/C signals that can be connected to a TV monitor. There is no provision to display decoded video on the computer screen. This functionality can be obtained by adding a TV card to the machine and looping back the PCMotion video output to the TC card's input.

The PCMotion card comes with an optional software development kit. The SDK includes DOS libraries (not used), a Windows DLL, and a sample source code. The API to the DLL was delivered in the form of a poorly commented header file. The Optibase product was not designed to decompress files obtained from the network, and its API focuses on local disk files access. Nevertheless, the API supported playback from the user space buffer. We uses this critical capability to implement the networked MCI driver for this card.

We also evaluated another hardware decoder manufactured by OptiVision. This card cooperates with the VGA display by using a so-called VGS feature connector. Specifically, the decoded video is overlaid with the VGA output in a window on the workstation screen. This technology is appealing as it removes need for a TV card. However, the VGA feature connector does not have the bandwidth to handle decompressed video, and the resulting quality is rather poor. Optivision provided us with an SDK that was somewhat more sophisticated than the Optibase product; but we abandoned software development for this card because of the poor quality of MPEG playback, the complicated card installation procedure, and buggy drivers that often crashed the entire system.

5.2.3 Windows networking tools

The Windows 3.1 environment originally did not include TCP/IP protocol suite. For many years, this market niche was exploited by companies providing third-party products. In general, installation and maintenance of a TCP/IP layer on Windows 3.1 machines was a major pain.

For this project, we tested some products, including packages for Microsoft, FTP, Sun (PC-NFS), and Trumpet (public domain shareware). Because we were interested primarily in development, it was important to find a Windows Socket implementation that was truly compatible. We determined that the Trumpet package is the most stable and robust.

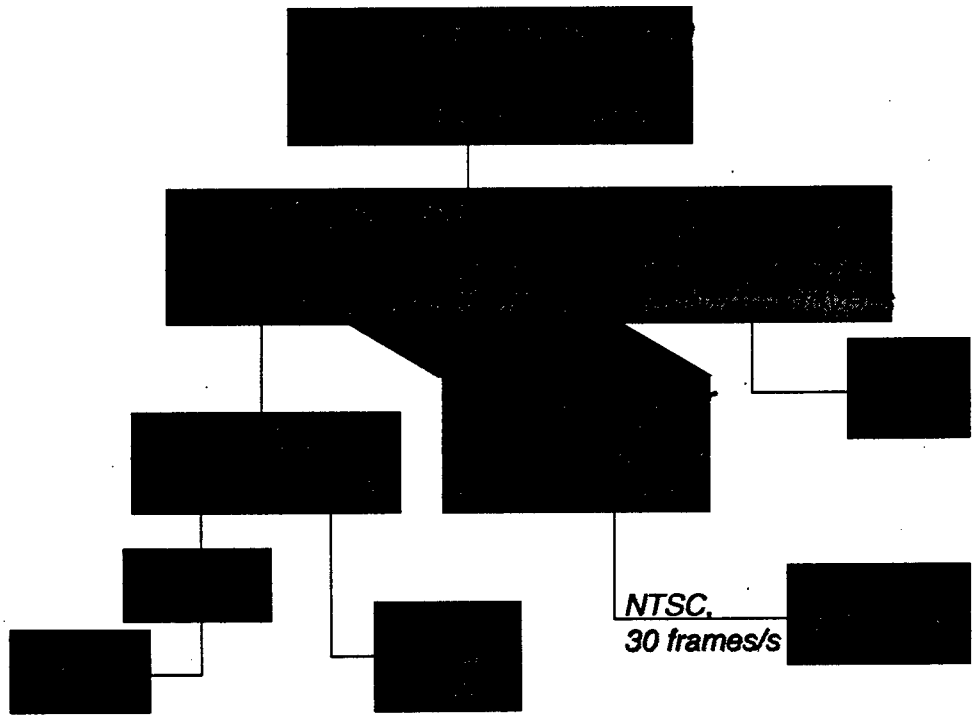
5.2.4 Media Control Interface Optibase MPEG driver

The main objective of creating a driver for the Media Control Interface (MCI) was to enable development of a user interface for a video client working with Optibase MPEG hardware. The general function of all MCI drivers is to provide a high-level interface to control various media devices through generalized commands such as play, pause, and stop. We used specific commands sets of MCI to control different device types in the Microsoft Windows system. However, the basic commands are the same for all device drivers. We decided to build such an

MCI device driver in order to blend our video client into the Windows multimedia environment. A properly implemented MCI driver can be loaded by the Microsoft Media Player, which is included in the standard Windows Accessories program set. Media Player is commonly used to control the playing of video files or audio wave forms from the disk, to control a VCR connected to the computer, as well as CD-Audio or CD-i. To make such control possible, each of these devices has its own MCI driver that handles the messages sent to it by Media Player. These messages are sent by Media Player in response to the user operating elements of the graphical interface. A properly implemented MCI driver must properly handle all MCI commands. The benefit of a standardized interface is that one can build hardware-independent multimedia applications in the Windows environment. Knowing only the general specifications of the MCI command set for this type of device, a developer can easily replace Media Player with a custom application. Typically, such applications are created using Visual Basic.

5.2.5 MCI driver in "Video for Windows" architecture

In the architecture we have described, all responsibility for the hardware-dependent aspects of developing multimedia applications remains with the MCI driver developer. The MCI driver we built dealt with two hardware interfaces: the network interface and the MPEG decoder interface.



The graph above displays the architecture of the MCI driver for the Video Client in VfW framework.

5.2.5.1 MCI driver in Video Client architecture

We designed a three-layer structure of the Video Client architecture. The top level is the end user application, including GUI and interacting directly with the user. (As mentioned, this application could be either the Media Player or another, custom application.) The intermediate layer is just the MCI driver that is loaded by the upper application as a dynamic link library (all MCI drivers are Windows DLLs). This layer implements the MCI interface. Finally, the bottom layer is the Video Client application that was responsible for networking and for dealing with decompressor board.

This architecture separates the MCI driver's DLL from the network and decoder layers. This is important in the case porting the driver to other hardware decoders. Another reason for this partitioning is the synchronization of events generated by the user, Media Player, sockets, and the decoder.

The most difficult part of the implementation process involved the non-preemptive nature of Windows 3.1. The driver architecture is, in fact, multithreaded; but such a model does not fit well with the message-driven Windows 3.1 architecture. Another set of problems involves mutual calls between two different DLLs, which is possible only when all pointers used in the data structures are 32 bit.

5.2.6 Technical details of the driver

We adopted the MCI AVI commands set as the most appropriate solution for digital video playback. The command set that the MCI driver must handle belong to two categories: (1) the general Windows driver set and (2) the MCI-specific driver set. We provided handles for loading, freeing, opening, closing, installing, and de-installing the Windows driver. We also generated an `oemsetup.inf` script that enables an automatic install of the driver, e.g., from a diskette in Windows 3.1 operating system.

The set of MCI specific commands that was handled and tested in the overall VoD architecture includes *open, close, capability, play, play from, stop, pause, resume, status length, status mode, status position, status start position, status time format, seek to, seek to start, set time format , set audio all on/of.*

The *Status* message was generated periodically (1-2 times per second) by Media Player to obtain internal driver information about its current mode (playing, stopped, seeking, ...), the current position of media (to update Media Player random access bar), as well as the length of the media. It took some experimenting with the Media Player to learn its behavior, e.g., timing of generated messages, to properly synchronize it with behavior of Video Client (depending on network and MPEG decoder).

The *Set* message is generally used to change time format of MCI driver (corresponding to Media Player time format option). Media Player requires only two basic time formats (milliseconds and

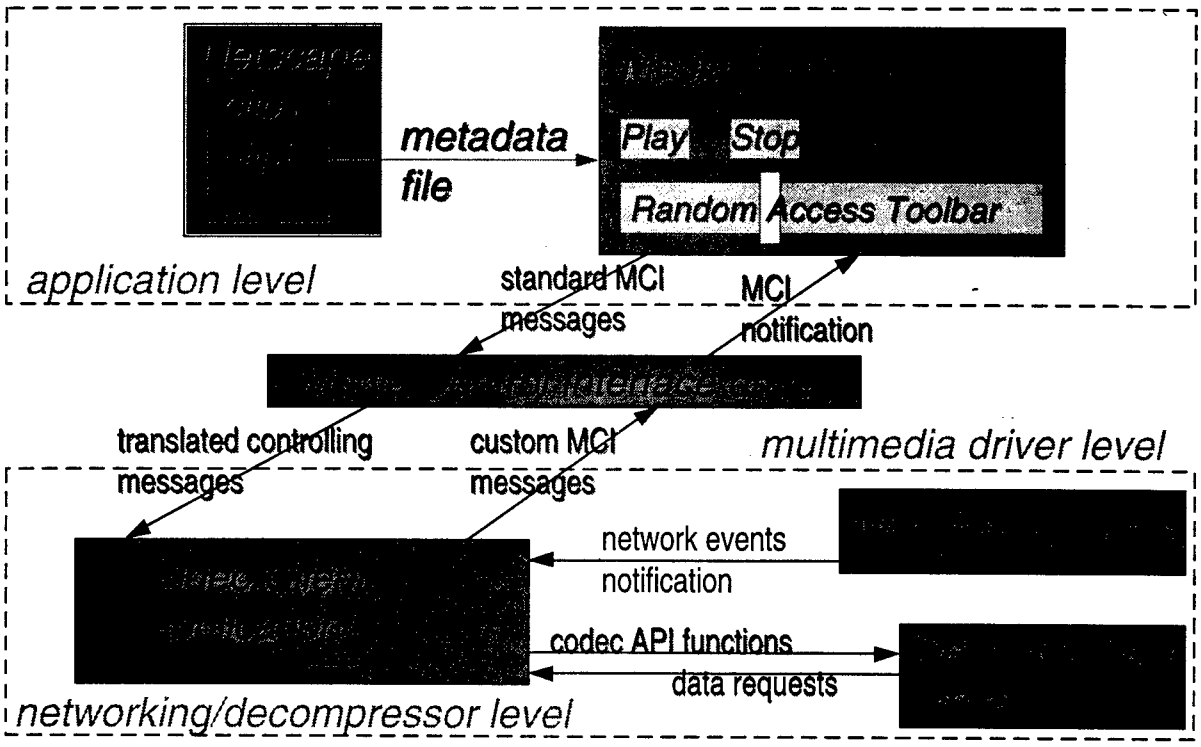
frames) to be provided by MCI driver; but other end user applications, according to specification of MCI, may require other formats. We developed the Optibase MPEG MCI driver to support all possible MCI time formats. We also added audio volume functionality to the MCI driver, and adequately updated the video client application by using the decoder's API.

5.2.7 Execution flow

The video client architecture requires seamless interaction among the following software components:

- Media Player or other top-level media control application,
- the MCI driver's DLL,
- the networked video client application (the code handling the network and controlling the decoder),
- the Optibase MPEG decoder's DLL,
- the Windows Sockets DLL,
- the application that activates TCP/IP stack (Trumpet Winsock), and
- Netscape browser.

Netscape browser executes Media Player after it receives the metafile from the database back-end. This is done automatically, thanks to Netscape's MIME extension mechanism that allows Media Player to be designated as a helper application for the metadata file type. The same mechanism transmits the metadata file from Netscape browser to Media Player. Next, Media Player opens the appropriate MCI driver (Optibase MPEG driver in this case), i.e., it loads its DLL This process is controlled by the information registered in Windows .ini files (win.ini, system.ini, mplayer.ini). While opening the MCI driver, the Video Client application is executed by the driver. Again, the metadata file is presented to the Video Client application. The Video Client application establishes connection with the Video Server daemon according to information provided in the metadata file.



The Video Client application loads the DLLs handling the decoder and Windows Sockets. Windows Sockets use the TCP/IP stack which must be activated before loading the Sockets DLL. However, since the TCP/IP stack also needs to be activated before using Netscape browser, the driver code assumes that the stack DLLs are loaded and active.

Communication between MCI driver and the Video Client application is enabled by Windows 3.1 API (`FindWindow` function). The MCI driver handle is transmitted to the Video Client application in the initialization phase. This allows the application to send driver messages back to the MCI driver.

5.2.8 Parsing metadata file

The metadata file needs to be parsed to retrieve the following information:

- Video Server IP address;
- pointer to the movie clip location in Video Server file system;
- file offset to the requested frame of the clip;
- video clip length in frames;
- indexing information (it is done for MPEG format files). This is a simple lookup table, translating frame numbers to file offsets. This information is needed since,

as mentioned, our server is codec independent. The indexing information is sorted by frame numbers. For MPEG, only indexes for first frames of the GOPs are provided, i.e., the stream access resolution is 15 frames, or one-half second. We found that certain MPEG decoders fail when fed a stream segment other than a GOP header.

5.2.9 Networked Video Client application

The basic functionality of the Video Client application is to load and stream MPEG files to the Optibase hardware decoder. MPEG files are received in real-time from the network and the application uses Windows Sockets to perform this task.

The Optibase decoder's API allows developers to use the user-space buffer to feed the decoder. The remainder of the API was designed to work with local files, but the initial tests that we performed with this API demonstrated that it is possible to use these functions to set the decoder for a networked playback. Unfortunately, decoder behavior in this kind of situation was undocumented, so we were forced to proceed by trial and error.

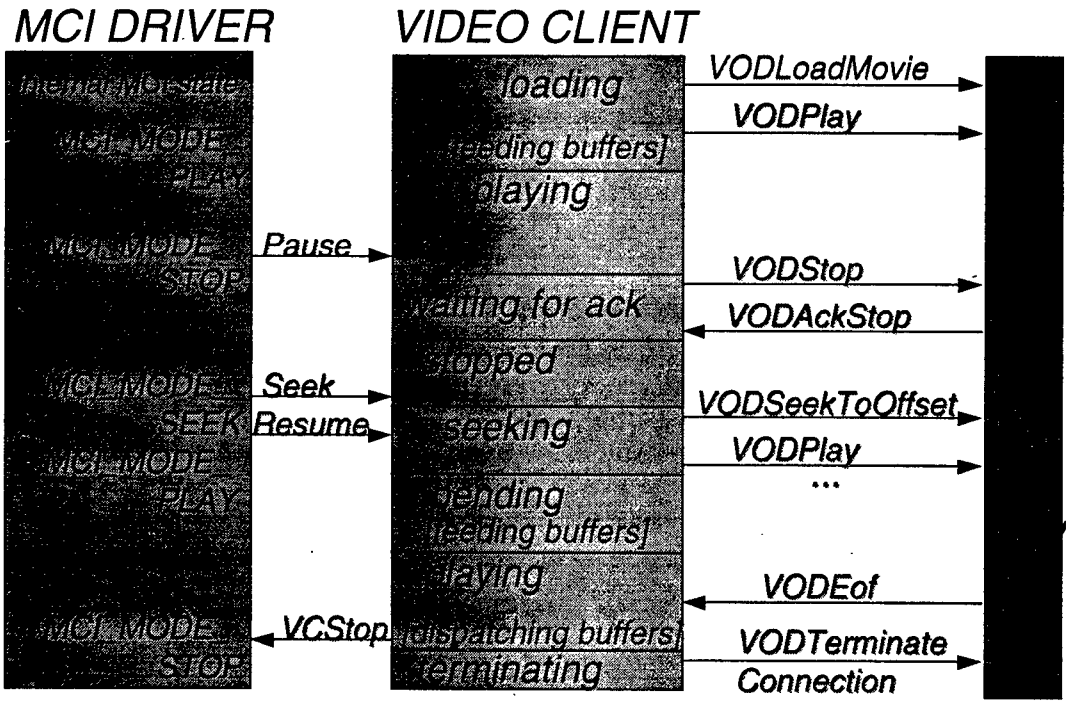
Communication with the decoder is set by implementation of a callback function (`GetSystemBuffer`) belonging to the decoder. This function is called by the decoder's DLL whenever the decoder needs new video data to decompress. It is responsibility of the Video Client application to provide the decoder with the user-space buffer containing valid video data.

The user-space buffer is being written to while the Video Client application is notified of a network event (i.e., new data came to the data socket). The Video Client application uses asynchronous sockets, i.e., every network event causes Windows Sockets DLL to send appropriate message to the application's message queue. Like the majority of Windows applications, the Video Client application dispatches these messages (and other Windows messages) to their respective handlers in a continuous loop.

The Video Client application supports the following operations:

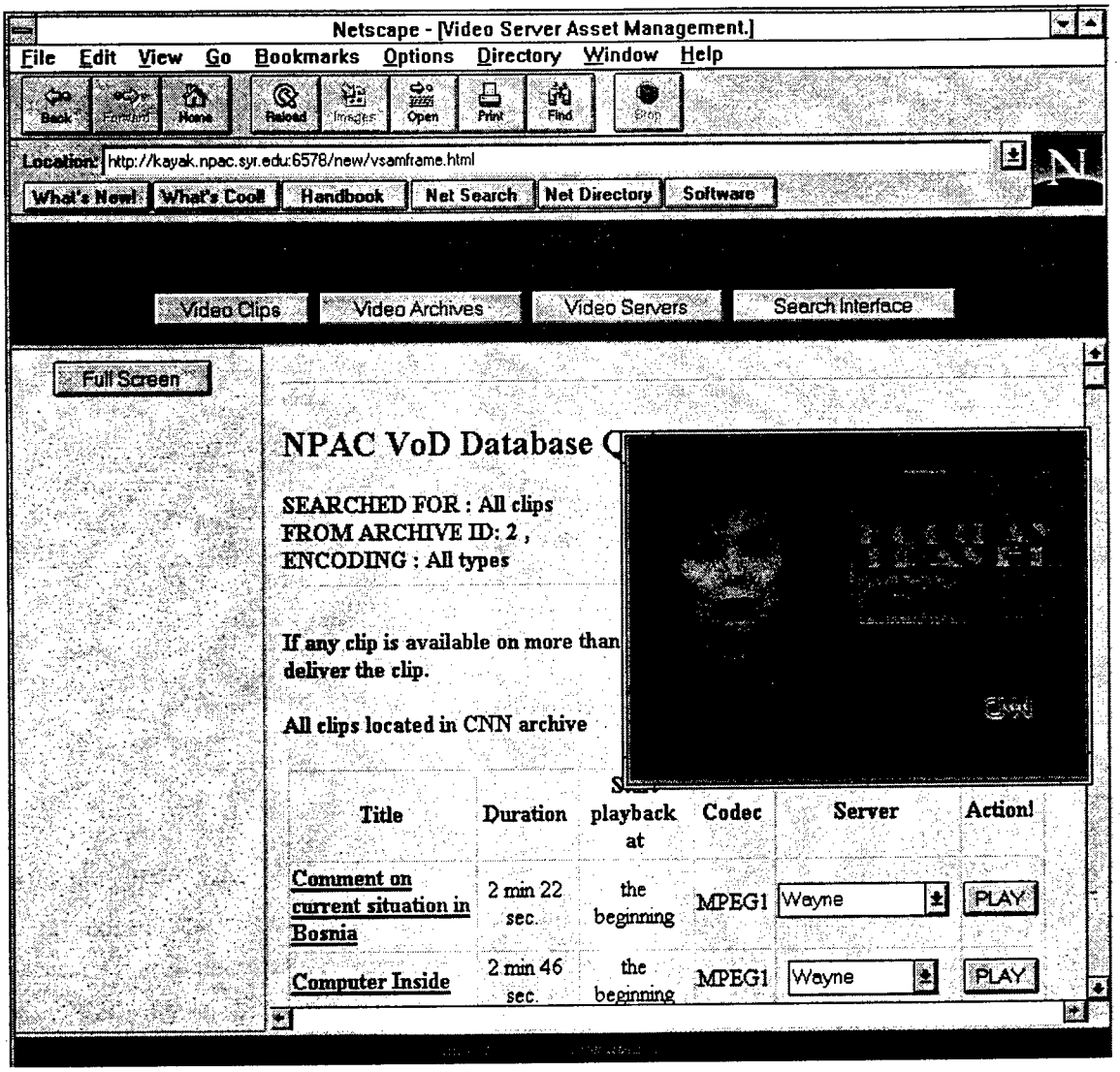
- handling of interactive playback protocol messages (`VoDPlay`, `VoDStop`, `VoDSeekToOffset`, `VoDAckStop`);
- handling the internal state to the Video Client application, and synchronization of its states with modes of the MCI driver; and
- translation of the MCI protocol to the video server protocol.

Details of the data flow in the video client application are illustrated in the diagram below.



5.2.10 Random access

Implementation of the random stream access for networked video is somewhat tricky. We could not use the random access functions of Optibase API since they were implemented strictly for local disk access. Instead, we reinitialized the decoder's DLL every time a random stream access was being performed. For this operation to work, we keep a copy of the MPEG header file in the Video Client buffer and attached it to the beginning of each MPEG stream segment sent to the decoder after a random seek operation. It is also necessary to flush completely all decoder buffers, to avoid the annoying effect of the decoder playing a few seconds of the old sequence before jumping to the new position. The operation of the buffer flush is undocumented and does not always work properly.



〈 Hardware supported MPEG video client. A video window is placed on the workstation screen via an auxiliary TV card displaying analog video produced by the hardware decoder.

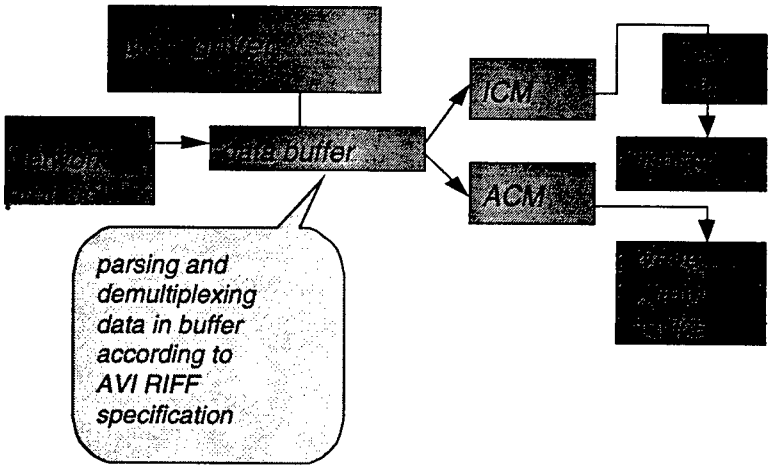
In the current implementation of the video client, there is a 3-to-5 second delay in performing the random access, independent of the bandwidth limitations of the network. The cause of this delay is the size of user-space buffers required by the Optibase decoder. Their total size is 1 MB; and it appears that the Optibase MPEG decoder needs to read a significant portion of these buffers to be able to start the playback. Our tests show that the decoder is not able to ignite with smaller buffers and that this bug is an internal feature of the MPEG decoder.

The random access operations are not completely stable for this version of the video client. Implementation of a more stable version would require better documentation of the decoder API and, possibly, access to the lower level of the decoder mechanics.

5.2.11 AVI Video Client

Due to the modular structure of the MCI video client, it is not very difficult to extend its functionality to AVI files. AVI files can be decoded by software codecs delivered with Windows. This is quite useful since Intel's recently introduced Indeo Video Interactive 4.1 codec has parameters comparable to MPEG decoders.

The graph below represents the Video Client architecture for AVI. The developed MCI driver and networking part of the Video Client application do not require significant changes. The Optibase API has to be replaced with VfW codecs API.



We have partly implemented such an AVI player. This implementation involved writing our own parser of the RIFF file format (AVI is a derivative of RIFF) such that all data structures that need to properly initialize software decoders are correctly retrieved from the video stream. This work was discontinued when the ActiveMovie architecture (see next subsection) made the AVI format obsolete.

The existing Video Client dealing with the Optibase decoder can be ported to 32-bit environment (Windows 95, Windows NT). The 32-bit Optibase decoder API is currently available.

5.3 Codec independent Active Movie video client

This section describes the video client architecture based on Microsoft's ActiveMovie technology. As indicated in Section 2, one of the project goals was to provide a hardware independent video client. The client based on the ActiveMovie technology meets this basic requirement at least for the PC-based workstations.

In the following subsections we summarize the salient features of the ActiveMovie technology and present a detailed discussion of the video client implementation.

5.3.1 ActiveMovie Technology

The following section is based on Microsoft documentation. It contains extensive citations. The quoted material has been selected so as to make it possible for the reader to understand the implementation of the ActiveMovie client for our VoD system.

5.3.1.1 General overview

ActiveMovie is an architecture that controls and processes streams of multimedia data. It is also a runtime that uses this architecture to enable users to play digital movies and sound encoded in various formats, including MPEG1.

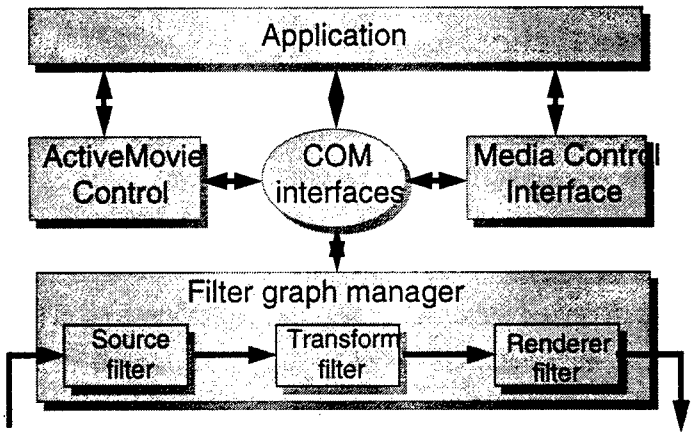
ActiveMovie playback capability makes use of video and audio hardware cards that support Microsoft's DirectX set of application programming interfaces (APIs). ActiveMovie also plays movie files in audio-video interleaved (.avi) or Apple QuickTime (.mov) format.

The ActiveMovie architecture defines how to control and process streams of time-stamped multimedia data by using modular components called "filters" connected in a configuration called a "filter graph." An object called the "filter graph manager" is accessed by applications and controls how the filter graph is assembled and how data is moved through the filter graph.

The filter graph manager provides a set of Component Object Model (COM) interfaces to allow communication between the filter graph and the application. Applications can directly call the filter graph manager interfaces to control the media stream or retrieve filter events, or they can use the ActiveMovie ActiveX (formerly OCX) control for higher-level programming.

ActiveMovie is the architecture to use for most new multimedia applications for Windows 95 or Windows NT. With a few exceptions, it replaces multimedia playback services, APIs, and architectures provided by Microsoft in earlier versions of the Windows Software Development Kit (SDK). The first release of ActiveMovie does not provide a corresponding replacement for each and every solution found in the previous multimedia technology. For example, there is no video capture capability built into the runtime. In these cases, ActiveMovie gives developers an opportunity to use its architecture to provide custom solutions.

ActiveMovie can be accessed through the following interfaces: the COM interface, the ActiveMovie ActiveX control, or Media Control Interface.



Because of the flexible, modular design of ActiveMovie architecture, there are many potential uses and applications for filter graphs. Some examples include filter graphs that implement video capture, control remote devices such as VCRs, sequence animation, and MIDI recording and editing.

5.3.1.2 Component Object Model in ActiveMovie

All components of the ActiveMovie filter graph architecture are implemented as COM objects. This includes the filters through which data is passed and filter components that serve as a connection between filters or allocate memory. Each object implements one or more interfaces, each of which contains a predefined set of functions, called *methods*. An application calls a method, or other component objects, to communicate with the object exposing the interface.

Filter graph architecture uses COM interfaces because they have the following properties:

- COM interfaces are publicly defined. This means that any filter that implements the correct predefined interfaces will work in a filter graph without any knowledge about the other filters since all filters are built with the same interface specifications.
- COM interfaces do not change after definition. A base set of interfaces are guaranteed to work; additional interfaces may be introduced to cover additional services. This definition prevents version problems.
- COM interfaces must have all methods implemented by any object that exposes. This assures that calling a method on the interface of an object will not generate an error.

- COM interfaces are discoverable. All COM objects support a method called `QueryInterface` that enables an external component to discover whether an interface is present and to retrieve a pointer to it.

COM interfaces are implemented by the object that exposes the interface (they do not contain an implementation themselves). The interface is essentially a contract for the functionality. Objects like the filter graph manager, or Microsoft filters, have implemented interfaces that can be accessed. A developer must implement the interfaces on custom-build filters.

Microsoft ActiveMovie provides a framework that simplifies the creation of Component Object Model (COM) objects.

ActiveMovie components are supplied as inprocess servers (that is, servers that run in the same address space as your application). They are packaged in a single dynamic-link library (DLL), quartz.dll. Developers can use the COM framework of ActiveMovie to build their own inprocess COM servers which they can package in their own DLL(s).

Typically, a single C++ class implements a single COM class. The ActiveMovie COM framework requires that C++ classes that implement COM objects conform to a few simple rules, and that the developer provides a class factory template for each such class. The *class factory template* contains information about the class that is vital to the framework.

COM objects are created by their class factories, are reference counted during their lifetimes, and self-destruct when their reference count drops to zero. COM objects may be created in isolation, or may be aggregated with an already existing COM object. In this second case, the existing object (referred to as the outer object) maintains the reference count. The created object (referred to as the inner object) is not reference counted, but will be destroyed by the outer object during the destruction of the outer object. (The application cannot directly manipulate COM objects; an application can only invoke the methods that the object chooses to expose through its interfaces. Typically, COM objects make several interfaces available. All COM objects must support the `IUnknown` interface that is responsible for the reference count.

The concept of a class factory is not specific to ActiveMovie; it is a common design that appears when the underlying type of the object being created is not known to the client that requests its creation. With COM objects, clients request interface pointers but know little about the underlying objects that implement that interface.

5.3.1.3 Filter Graphs

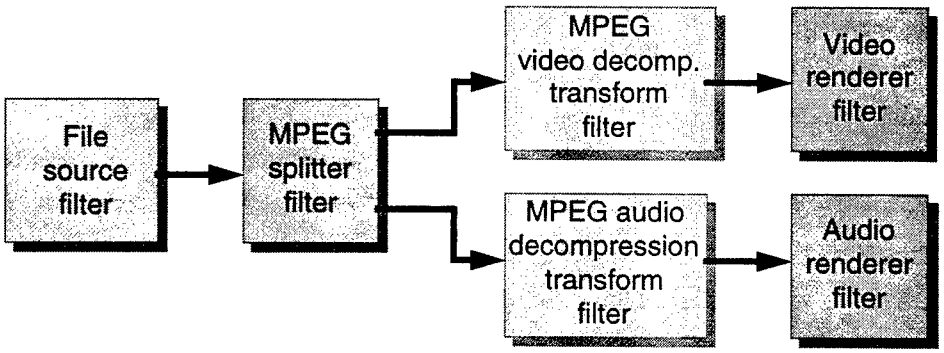
A filter graph is composed of a collection of filters of different types. Most filters can be categorized into one of the following three types:

- a source filter, which takes the data from some source, such as a file on disk, a satellite feed, an Internet server, or a VCR, and introduces it into the filter graph;
- a transform filter, which takes the data, processes it, and then passes it along; or
- a rendering filter, which renders the data; typically this is rendered to a hardware device, but could be rendered to any location that accepts media input (such as memory or a disk file).

For example, a filter graph whose purpose is to play back an MPEG-compressed video from a file would use the following filters:

- a source filter to read the data off the disk,
- an MPEG filter to parse the stream and split the MPEG audio and video data streams,
- a transform filter to decompress the video data.
- a transform filter to decompress the audio data,
- a video renderer filter to display the video data on the screen, and
- an audio renderer filter to send the audio to the sound card.

The following illustration shows such a filter graph:



Filter graphs stream multimedia data through filters. In the media stream, one filter passes the media *downstream* to the next filter. An *upstream filter* describes the filter that passes data to the filter; a *downstream* filter describes the next filter in line for the data. This distinction is important because media flows downstream, but other information can go upstream.

To make a filter graph work, filters must be connected in the proper order, and the data stream must be started and stopped in the proper order. The filter graph manager connects filters and controls the media stream. It also has the ability to search for a configuration of filters that will

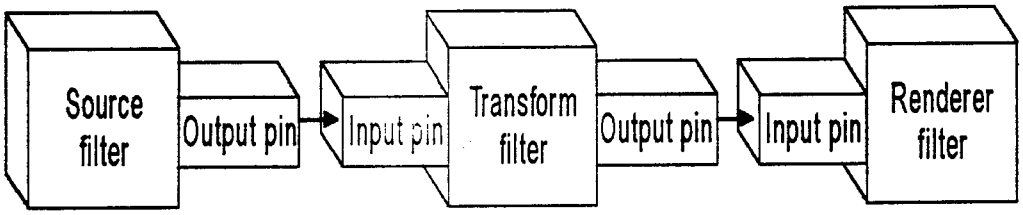
render a particular media type and build that filter graph. Filter graphs can also be pre-configured, in which case the filter graph manager does not need to search for a configuration.

5.3.1.4 Stream architecture

Stream architecture defines objects and interfaces that exchange streams of time-based data. In particular, it defines interfaces for the following requirements:

- connecting filters to other filters,
- negotiating data types,
- transporting data between filters,
- synchronizing presentation of data, and
- graceful degradation of rendering in cases of insufficient resources (that is, *quality-control management*).

The two basic components used in the stream architecture are filters and pins. A *filter* is a COM object that performs a specific task, such as reading data from a disk. For each stream it handles, it exposes at least one pin. A *pin* is a COM object created by the filter that represents a point of connection for a unidirectional data stream on the filter, as shown in the following illustration:



Input pins accept data into the filter, and output pins provide data to other filters. A source filter provides one output pin for each stream of data in the file. A typical transform filter, such as a compression/decompression (codec) filter, provides one input pin and one output pin, while an audio output filter typically exposes only one input pin. More complex arrangements are also possible.

Reference clock synchronization is accomplished by implementing the `IReferenceClock` interface on any filter that has a reference clock. For example, because sound cards are predominantly used for reference clocks, the audio renderer filter implements this interface, which essentially allows any caller to register for the receipt of time notifications.

The ActiveMovie stream architecture provides for graceful adaptation of media rendering to overloaded or underloaded media streams. The `IQualityControl` interface is used to send quality-control notifications from a renderer filter either upstream, eventually to be acted on by

some filter in the graph, or directly to a designated quality control manager. The base classes implement the passing of quality control notifications upstream by providing the `IQualityControl` interface on the output pins of filters. Quality control notification uses a Quality structure, which indicates whether the renderer is overloaded or underloaded. A filter capable of, say, discarding samples to relieve and overloaded condition, can then act on this notification. This is typically done by a source filter but could be done by other filters. For example, the ActiveMovie AVI Decoder filter skips samples until the next key frame when it receives a quality control notification.

The stream architecture allows applications to communicate with the filter graph manager; it also allows the filter graph manager to communicate with individual filters to control the movement of the data through the filter graph. Using the stream architecture, filters can post events that the application can retrieve, so an application can, for example, retrieve status information about a special filter it has installed.

The filter graph manager exposes media control and media positioning interfaces to the application. The media control interface, `IMediaControl`, allows the application to issue commands to run, pause, and stop the stream. The positioning interface, `IMediaPosition`, lets the application specify what section of the stream to play.

Individual filters expose a media control interface so that the filter graph manager can issue the run, pause, and stop control commands. The filter graph manager is responsible for calling these methods in the correct order on all of the filters in the filter graph. (The application should not do this directly.)

The position commands are handled less directly. The filter graph manager gets called by the application, for example, to play a specified length of the media stream starting at some specified stream time. However, unlike the media control interface, only the renderer filter exposes a media position interface. Therefore, the filter graph manager calls only the renderer filter with positioning information. The renderer then passes this position control information upstream through media position interfaces exposed on the pins, which simply pass it on. The positioning of the media stream is actually handled by the output pin on the filter that is able to seek a particular position (for example, a file source filter) because pins are responsible for the data transport mechanism.

Position information is passed serially upstream because there may be filters between the renderer and the source filter that require position information. Consider a transform filter that is written to perform some video or audio modification only during the first 10 seconds of a video clip (for example, increasing the volume or fading in the video). This filter probably requires information about where the stream is starting so it can determine its correct behavior. For example, it should not perform if the start time is after the first 10 seconds, or it should adjust accordingly if the start time is within this duration.

5.3.1.5 COM Interfaces

The ActiveMovie COM interfaces comprise the schematic of an architecture for streaming time-stamped media. The filter graph, through which media flows, is composed of objects, such as filters, pins, media samples, allocators, and enumerators, that work together. COM interfaces are implemented on these objects and are called by other objects with which they interact. The filter is the only filter graph COM object that has a CLSID; all other objects in the filter graph support COM interfaces and are created as needed by the filter. Filters and their supporting object must implement their COM interfaces and a class library is available for help in that task. The filter graph manager, on the other hand, has a CLSID and supports several fully implemented interfaces

The ActiveMovie COM interfaces can be categorized as follows:

- Filter graph manager interfaces, which are fully implemented and used by applications to create, connect, and control filter graphs and by filters within the filter graph to post event notifications and to force reconnections when needed.
- Filter and pin interfaces, which must be implemented by the filter. They comprise the methods exposed by filters for communicating with the filter graph manager, connecting with other filters, passing data downstream (from source filter to renderer filter) and passing quality control and media positioning information upstream (from renderer to source).
- Enumerator and media sample interfaces, which are interfaces on objects created temporarily for passing information.
- Control interfaces, which are exposed by filters and the filter graph manager to enable the starting, stopping, and positioning of media in the stream. The control interfaces on the filters must be implemented when writing a filter, whereas they are already implemented on the filter graph manager.

The filter is the main COM object and has a class ID (CLSID) and name registered in the registry. Filters must provide access to their pins and otherwise communicate with the filter graph. They must also allow the filter graph manager to manage the data flow by accepting state change messages.

At a minimum, a filter exposes the `IFilter` interface. This interface provides methods that allow the enumeration of the pins on the filter and return filter information. It also provides the inherited methods from `IMediaFilter`; these methods allow control of state processing (for example running, pausing, and stopping) and synchronization and are called primarily by the filter graph manager.

Pins do not normally have registered class identifiers and are usually created by the filter object on which they reside. They are exposed externally by the filter, which includes a method (`IFilter::EnumPins`) to hand out pointers to the `IPin` interfaces of its pins, normally to the filter graph manager. The filter graph manager is responsible for connecting pins by calling an `IPin` method on one of the pins with a pointer to the other pin. Once pins are connected, each pin holds a pointer to the pin to which it is connected.

Media sample and enumerator interfaces are temporary objects created to pass information or data between objects. They do not have class identifiers.

The media sample interface is created from the memory allocator, which uses the media sample object as its unit of exchange. It has no class identifier. It is the unit of media data that is passed from one filter to the next via the memory allocator shared by two connected pins.

Enumerators in ActiveMovie are based on the OLE `EnumXXX` interfaces. They include the Next and Prev methods, which tell the enumerator what item or items to return; the Skip method, which skips one or more items; and the Clone method, which makes a copy of the enumerator. Enumerators are used to present lists of items such as filters in a filter graph, pins on a filter, or media types that are preferred by a pin.

Control interfaces allow the filter graph manager to coordinate the activities of the data stream with filters. They are exposed by the filter graph manager, filters and pins.

5.3.1.6 Data flow

Data flow in the filter graph occurs in the following ways:

- Media sample data flows from one filter to the next--originating at a source filter and terminating, eventually, at a renderer filter.
- Control information, such as end-of-stream and flushing notifications, travels with the media data stream from filter to filter.
- Notification events flow from the filters to the filter graph manager and, optionally, to the application.
- Filter graph control data flows from the application to the filter graph manager and finally to the filters themselves.
- Quality control data originates in the renderer and flows upstream through the filters until it finds a filter capable of cutting back or increasing the media data flow. It may also flow directly to a quality control manager if one is registered.

ActiveMovie filters pass media data downstream, that is, from the output pin of one filter to the input pin of the next filter. The flow and control of the data is effected by the interfaces on those

pins and the filters themselves. Data streaming activity is serialized by the filters; all data streaming calls for a given pin are explicitly serialized and usually originate from a single thread.

Data is passed from the output pin of one filter to the input pin of the next. The two connected pins agree upon a common method of exchanging data, called a transport. The most common transport is the local memory transport. This transport is implemented if the input pin supports the `IMemInputPin` interface. The ActiveMovie class library base classes assume this transport.

Filters must follow protocols in order to pass and receive media samples. The connected pins must agree upon the allocator to be used, must have a means of passing the data, and must follow the correct procedure for holding onto a sample or releasing it back to the sender. Media samples are data objects that support the `IMediaSample` interface. They are obtained from an allocator, most likely represented by an object supporting the `IMemAllocator` interface.

An output pin typically exposes the following interfaces:

- IPin methods are called to allow the pin to be queried for pin, connection, and data type information, and to send flush notifications downstream when the filter graph stops.
- `IMediaPosition` allows information about the stream's duration, start time, and stop time to be relayed from the renderer. The renderer passes the media position information upstream to the filter (typically the source filter) responsible for queuing the stream to the appropriate position.
- `IQualityControl` passes quality-control messages upstream from the renderer to the filter that is responsible for increasing or decreasing the media supply.

An input pin typically exposes the following interfaces:

- `IPin` allows the pin to connect to an output pin and provides information about the pin and its internal connections.
- `IMemInputPin` allows the pin to propose its own transport memory allocator, to be notified of the allocator that an output pin is supplying, to receive media samples through the established allocator, and to flush the buffer. This interface can create a shared memory allocator object if the connected pin does not supply a transport memory allocator.

The standard transport interface, `IMemInputPin`, provides data transfer through shared memory buffers, although other transport interfaces can be used. For example, where two components are connected directly in hardware, they may connect to each other by using the `IPin` interface and

then seek a private interface that can manage the transfer of data directly between the two components.

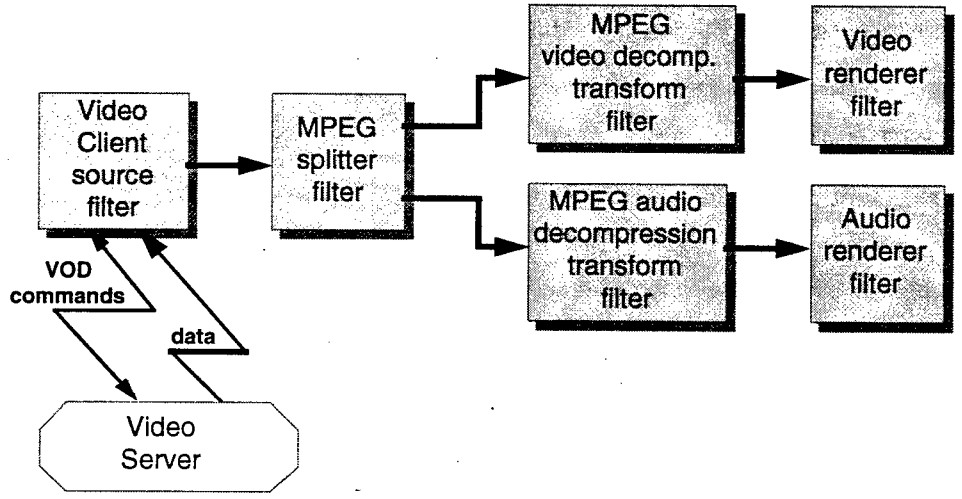
Control data originates at the application and is passed to the filter graph manager. At the COM level, this is handled by filter graph manager interfaces in the control.odl type definition library. Examples of control data are calls to the `IMediaControl` interfaces, such as `IMediaControl::Run`, `IMediaControl::Pause`, and `IMediaControl::Stop`. The `IMediaPosition` and `IMediaSelection` interfaces provide access to moving forward or backward in a media stream.

The most important thing to understand about the flow of control data is that it should always pass through the filter graph manager first. This is because there is usually an order that must be followed in controlling the filters in the filter graph to avoid deadlocks and other problems. The filter graph manager is dedicated to handling these conditions.

5.3.2 Video Client in ActiveMovie

5.3.2.1 Filter graph design

We designed the video client for Windows 95/NT to use extensively the ActiveMovie architecture. The part of the Video Client encompassing most of ActiveMovie's functionality is the filter graph. In our video client, we replaced the functionality of the ActiveMovie file source filter with a new source filter that reads video data from the network. The first filter graph we built using the Video Client source filter worked with the Microsoft software MPEG1 decoder. The figure below presents the network MPEG filter graph.

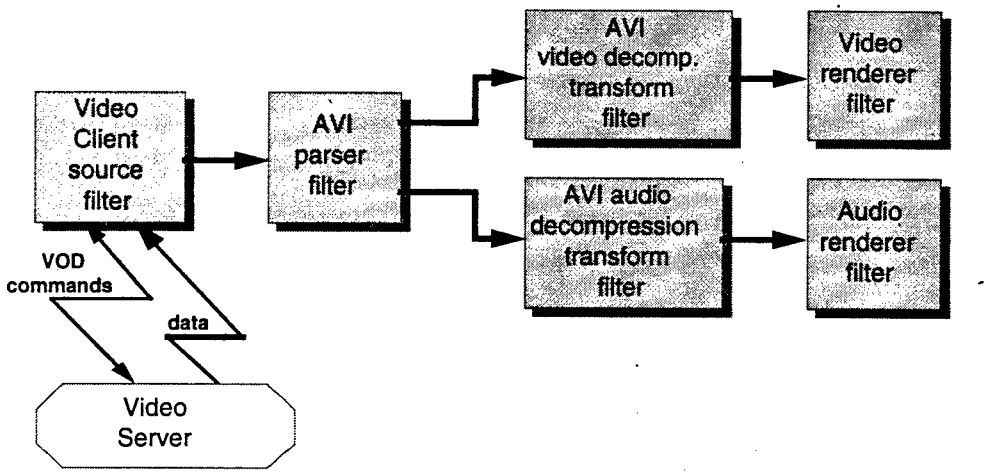


In this filter graph, all filters are provided by ActiveMovie architecture, except for the source filter.

The Video Client source filter connects to Video Server, initiates video data streaming and interacts with the Video Server through a custom VoD protocol. The MPEG1 stream is then copied to video data buffer, in order to eliminate effects of "burstiness" of network streaming in displaying video data (i.e., video or audio breaks). The video data buffer can store a few seconds of video. It is read by the filter to which the Video Client source filter connects. In this filter graph, this is MPEG splitter filter. The MPEG splitter filter is responsible for the demultiplexing of the MPEG1 system stream coming from the Video Client source filter. Later, two separated streams--MPEG1 Video and MPEG1 Audio--are transmitted from the parser filter to their respective decompressor filters. Finally, video and audio renderer filters perform tasks of video and audio playback.

In this model, the Video Client filter connects to two other components in significantly different ways. It connects to the Video Server, which has nothing to do with ActiveMovie architecture - the custom VoD protocol supports control and data flow. To connect to an ActiveMovie parser filter, the Video Client source filter must implement ActiveMovie COM interfaces and must conform to connection and data flow mechanisms of ActiveMovie (the MPEG splitter filter is actually such a parser filter since it parses the MPEG1 system stream to retrieve video and audio streams).

Like every ActiveMovie filter, the Video Client may own pins that will be responsible for communication between neighboring filters. The Video Client has one neighbor filter, a parser filter; therefore, it has to provide one output pin to connect with the input pin of the parser filter. The major COM interface enabling communication between the parser filter and the Video Client source filter is the `IAsyncReader` interface, exposed by the output pin of the Video Client source filter. Parser filters request multimedia streams from a source filter using the `IAsyncReader` interface. The MPEG splitter invokes `IAsyncReader` interface methods on the Video Client source filter. The same interface is invoked on the source filter by AVI parser filter as well as QuickTime parser filter. The last two filters use a subset of `IAsyncReader` methods other than MPEG, as well as another communication model. The Video Client source filter implements the entire `IAsyncReader` interface and implements both communication models. Therefore, the Video Client source filter can be connected to and work with AVI and QuickTime filter subgraphs of ActiveMovie architecture.



The graph above is similar to the previous one. The Video Client source filter uses the same network communication process. The communication of the source filter with the AVI parser filter is slightly different in that it uses other methods of the `IAsyncReader` interface. The MPEG stream processing functionality in downstream filters of the network MPEG filter graph is duplicated by AVI stream processing functionality in the network AVI filter graph.

5.3.2.2 Video Client Source Filter classes

The construction of two objects is necessary for incorporating networking functionality in ActiveMovie filter graphs. The first object is the Video Client source filter; the second object is the Video Client output pin. The Video Client source filter is responsible for initialization tasks, state management, and handling media control messages sent to it by the filter graph manager. The source filter must be registered in the Windows 95/NT system as an inprocess server. The Video Client output pin is not registered in the system because its existence is strictly related to existence of the Video Client source filter. When the filter graph manager constructs the designed network filter graph, it adds all required filters to the graph, including the Video Client source filter. Then, the source filter creates a Video Client output pin object. The Video Client source filter is a COM object and a class implementing it must conform to COM specifications. The Video Client source filter has to handle its reference count and provide interface to other objects to do it. In such a way, the Video Client output pin can exist as an internal part of the source filter, without its own reference count but, instead, using the reference count of the Video Client source filter.

When the Video Client source filter has to provide functionality of the COM object, the Video Client output pin is responsible for data streaming.

An implementation of the Video Client source filter and output pin provides the following classes:

- `CMemReader`, the Video Client source filter class. The object of the class creates a worker thread with the sole responsibility of maintaining the socket window and the socket window procedure. The window procedure handles network event notification messages being sent to the window. Handling of a READ DATA event copies data from the data socket to the video data buffer. The buffer critical section is locked while copying to the video data buffer.
- `CAsyncReader`, base class for `CMemReader`, implements `IFilter` interface.
- `CAsyncOutputPin`, the Video Client output pin class, implementing `IAsyncReader` interface. The pin provides its own allocator if connecting transform filter does not support it.
- `CAsyncIO`, a class responsible for queuing and processing of asynchronous requests for multimedia data from parsers. The object of this class creates a worker thread to enable asynchronous request handling. The class provides Video Client source filter with two lists: requests list and idle list. The first list consists of requests actually performing reading or writing operations. The idle list represents available resources (i.e., not in actual use) of the common, source, and parser allocator (number of available media samples). Lists are manipulated by request and completion events. A communication between threads of the class uses shared memory and events.
- `CAsyncRequest`, a class representing a single parser request for block of data. Used by `CAsyncIO` class.
- `CMemStream`, a class representing the multimedia stream and providing methods to access the stream. The stream is implemented in the class as a circular video data buffer. Virtually it is seen as a whole multimedia stream. The `CMemStream::SetPointer` method is responsible for seeking a position in the stream and for determining whether the Seek command has to be sent to Video Server. If the Seek command is sent to the Video Server, the method does not return until there is enough data in the video data buffer to resume displaying or a time-out occurs. The `CMemStream::Read` method provides a way of transparent reading from the video data buffer. The buffer's critical section is locked for the time of accessing of the buffer.

5.3.2.3 IAsyncReader interface

`IAsyncReader` is the most important interface supported by Video Client ActiveMovie objects. It enables the connection and communication between the asynchronous source filter and parsers.

The Video Client output pin implements this interface while parsers are calling messages of the interface to read multimedia data from it. The `IAsyncReader` interface allows multiple overlapped reads from different positions in the media stream. The `IAsyncReader` interface has to be implemented if a filter reads data of media type `MEDIATYPE_Stream` from some source. The Video Client source filter is such a filter.

`IAsyncReader` supports two modes of data transfer: synchronous and asynchronous. `IAsyncReader::SyncRead` and `IAsyncReader::SyncReadAligned` are methods of synchronous mode. A pair of `IAsyncReader::Request/ IAsyncReader::WaitforNext` messages implements asynchronous mode.

IAsyncReader interface consists of the following methods:

IUnknown methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments the reference count.
Release	Decrements the reference count.

IAsyncReader methods	Description
RequestAllocator	Retrieves the actual allocator to be used.
Request	Queues a request for data.
aitForNext	Blocks until the next sample is completed or the time-out occurs.
SyncReadAligned	Performs an aligned synchronized read.
SyncRead	Performs a synchronized read.
Length	Retrieves the total length of the stream, and the currently available length.
BeginFlush	Causes all outstanding reads to return.
EndFlush	Ends the flushing operation.

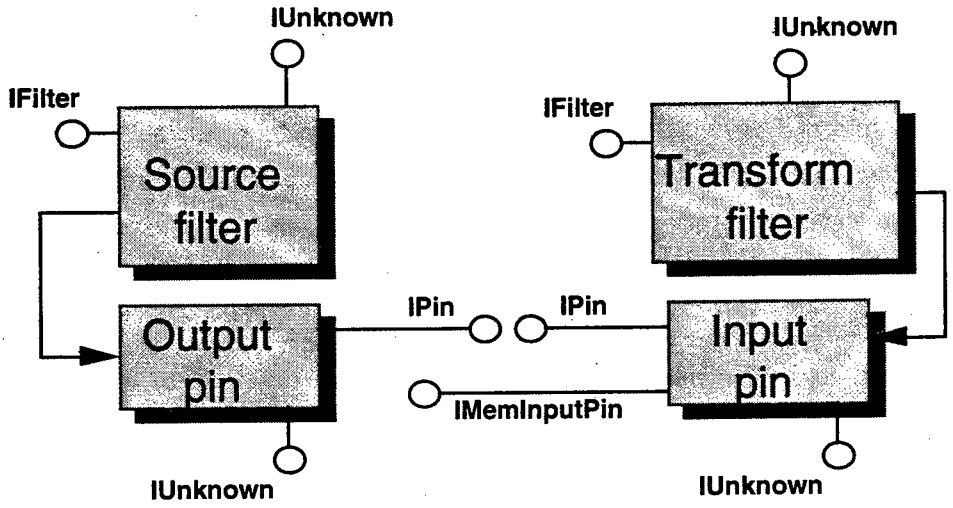
5.3.2.4 Connection process

The Video Client source filter connects to various parser filters. The connection model used by a parser filter at its input is totally different from the standard connection model found in ActiveMovie. This difference accounts for the inability of the Video Client source filter and output pin to inherit from the base classes of ActiveMovie: `CSource` and `CSourceStream`. They may inherit only from the most generic `CBaseXXX` classes.

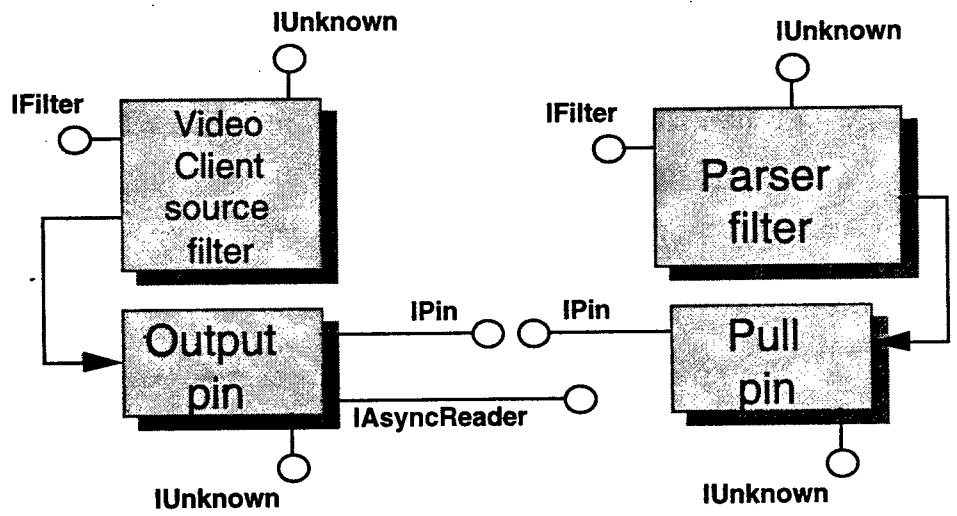
In the standard connection model, the source filter invokes the providing of new data to the filter graph. Because of this, it influences the states of downstream filters. Therefore, the source filter is the best place to create a worker thread that queues and processes control requests from the graph manager. The source filter is "pumping" data to a downstream filter. The source filter calls

`IMemInputPin` interface methods of the downstream filter to deliver data. This model is similar to the "server-push" model in CGI programming.

The following illustration represents the filters, pins, and interfaces (not all of them) taking part in the standard source filter connection with a downstream filter.



Contrary to the standard connection model, the parser filter invokes the providing of new data to the filter graph. Therefore, the parser filter is the best place to create a worker thread which queues and processes control requests from the graph manager. More precisely, the worker thread is created on the input pin of the parser filter. Also, the input pin of a parser is completely different from the standard input pin implementation. The Video Client output pin implements an `IAsyncReader` interface which is used by the parser input pin. The parser input pin may request data anytime it wants, and from different positions. This is why the parser input pin is called a Pull Pin. This model is similar to the "client-pull" model in CGI programming.



5.3.2.5 *Video-On-Demand protocol*

The custom Video-on-Demand protocol was implemented to provide an end user with interactive Video-on-Demand. The protocol handles data and control flow over the network. Data flow is unidirectional, i.e., the multimedia data streaming goes only from Video Server to Video Client. Data flow can have high network bandwidth requirements. In case of MPEG1 streams sent over the network, the required bandwidth is 1.5 Mbit/s of MPEG1 stream data rate plus the overhead of network protocol. Research has shown that the TCP/IP protocol requires 15-20% overhead when transporting multimedia data. Contrary to the data flow, control flow in the VoD protocol is bi-directional and does not have high bandwidth requirements. In case of control flow, its reliability is crucial--the control commands sent from and to the Video Client cannot be lost or reordered.

Current implementation of data and control flow uses sockets. Because both Video Server and Video Client are implemented on Windows platforms, they make use of Windows sockets. There are two sockets created: one to handle the data flow, one to handle the control flow. There is a video data buffer in Video Client supporting the data flow. In the current implementation, the control flow commands are

- from the Video Client: Load Movie, Play, Stop, Seek to Offset, Terminate Connection;
- from the Video Server: Stop Acknowledgments, End of File notifications;

As can be seen, more control capabilities reside on Video Client side. These controls enable the end user to interact with movies being sent from Video Servers. The video protocol on Video Client side is implemented using asynchronous mode of Windows Sockets. There is a callback window created that is responsible for handling of all network (socket) events. The handle to the window is passed to Windows Sockets DLL, together with set of message values that the window would like to obtain as a result of some network events. Generally, there are three types of events that concern the Video Client:

- a portion of multimedia data that has come on the data socket and can be read;
- Windows Sockets are able to send some command to the Video Server, control socket can be written;
- an acknowledgment or notification has come on the control socket and can be read.

While initializing sockets, the Video Client informs Windows Sockets what events it is interested in and what messages Windows Sockets should send to the callback window. In this way, I/O

operations on data and control sockets can be performed asynchronously. The callback window procedure will handle network events when they occur, but the Video Client does not have to poll for them.

5.3.2.6 Video data buffer

The video data buffer was built to minimize video and audio blocking effects when network delivery service is bursty. A size of the buffer may differ depending on the bitrate of video stream, i.e., the video data buffer should be smaller for H.263 video streaming than MPEG video streaming. For an MPEG1 system stream, the size of video data buffer is 1-2 Mbytes, corresponding to 5-10 seconds of an MPEG1 compressed movie. The buffer of the same size could store up to 10 minutes of H.263 compressed video clip. There is no need for such a large buffer in case of H.263. Moreover, for video clips less than 10-minutes long, the Video Client could download all the movie before actually displaying it.

The video data buffer is a circular buffer, with read and write position pointers. The video data buffer has few thresholds, indicating how much the buffer is full or empty. There are three important thresholds:

- video pending threshold: this threshold represents the number of bytes that have to be read from the network and written to the video data buffer before video displaying is activated. The higher the threshold, the longer an end user has to wait for video display started. The lower the threshold, the more likely the Video Client will have to stop displaying video due to lack of new data (remember bursty character of traditional network traffic). Therefore, the threshold must be a tradeoff between these two conditions.
- network inactivating threshold: if the number of bytes in the video data buffer exceeds the threshold, the Video Client should stop reading video data from Video Server. The situation could happen if network connection was faster than required. This threshold eliminates the danger of overloading the video data buffer. The threshold has to be bigger than the video pending threshold, otherwise the video would never start playing.
- network reactivating threshold: this threshold should be little less than the network inactivating threshold. The difference should not be too small, otherwise it will result in frequent network activating and inactivating procedures, which could impact the CPU (utilize CPU time for initialization and un-initialization functions).
- video pausing threshold: this threshold is needed when video data is not received on the data socket for a long time, and the video data buffer is almost empty. The threshold should cause the Video Client to stop video displaying before video

data buffer is empty. The Video Client should still try to read from sockets for some time; and if video pending threshold is reached before a time-out occurs, the Video Client should resume displaying the video material. This threshold is the smallest of all video data buffer thresholds.

The thresholds can be adjusted according to network or CPU conditions. They could be changed dynamically, based on a history showing utilization of the video data buffer. However, in current implementations of Video Clients for the PC, they are fixed.

There is one important difference between the video data buffer implementations for Video Client source filter in ActiveMovie and for Video Client with PCMotion MPEG hardware decoder: the ActiveMovie architecture often requires small readjustments of video position, especially while pausing video streaming. It is not very efficient to send Seek message to the Video Server each time the architecture wants to move few video frames back or forward, especially when this could be improved by redesigning the video data buffer. The Video Client for Windows 3.1 has the network inactivating threshold at 95% of buffer size. The Video Client in ActiveMovie has increased the buffer size but lowered the threshold to 60%. By this operation, the circular video data buffer has 40% of "memory," i.e., the video data that was already read by decompressing and rendering filters is still being stored in the buffer. Now, when ActiveMovie architecture moves several frames back (for 1.2 Mbytes buffer it can move back up to 2.5 seconds), the Video Client is able to restore the old state of video data buffer and does not need to perform the lengthy seek operation.

5.3.2.7 Video Client flow of execution

The user interaction with the NPAC VoD system starts with the search interface provided in an HTML page display by Netscape browser. Once the search results are generated, the user can choose which video material to display. By choosing a result, an end user sends an URL to a CGI script of the Database Server, which makes the Database Server generate the metadata file about a video clip. This metadata file is sent via HTTP from the Database Server to Netscape browser on user's computer. The custom MIME type of the metadata file (vod/nvs) makes Netscape launch the Video Client application. To choose the application, Netscape checks its configuration database; thus, the relation between MIME type and Video Client application must be pre-configured in Netscape.

The Video Client application is written in Visual Basic and has following goals:

- to display the ActiveMovie OLE Control with media control buttons and position slider enabled;
- to pass the metadata file to the ActiveMovie OLE Control; and

- to start displaying from a particular position in the movie, based on the *Offset* field of metadata file.

Thus, Visual Basic makes initial parsing of metadata file.

When the ActiveMovie OLE control is created, the flow of control conforms to ActiveMovie architecture. First, the ActiveMovie OLE Control activates the filter graph manager object. The filter graph manager starts creating a filter graph by choosing a source filter able to handle the input file. The input file for ActiveMovie OLE Control was the metadata file. The metadata file has a header 'NVSID' which is registered in ActiveMovie registry tree. This header is associated there with Video Client source filter. In this way, the filter graph manager knows that it has to add the Video Client source filter. After the source filter is added , the filter graph manager adds Video and Audio Renderer filters to the filter graph and tries to connect them intelligently with the source filter, using different combinations of transform filters between them.

When the Video Client source filter is added to the filter graph, the filter graph manager asks it for the `IFileSourceFilter` interface. The Video Client source filter has to support this interface if it wants to make use of the metadata file. Using this interface, the filter graph manager passes the metadata file to the Video Client source filter.

Now the Video Client source filter can parse the metadata file and retrieve the Video Server IP address as well as the path to the video file on the remote file system. From the extension of the remote file name, the Video Client source filter decides what type of media it is supporting. This media type is presented to transform filters, which the filter graph manager tries to add to the filter graph. The connection is negotiated between the Video Client source filter and transform filters until a filter supporting the remote file media type is found. In this way, remote MPEG1 files will be streamed from the Video Client source filter to the MPEG splitter and further, AVI files will be streamed to AVI parser, and so on.

There is another COM interface that the Video Client source filter could implement to have more influence on building custom filter graphs, IStreamBuilder. When the interface is implemented, the filter graph uses it to build the filter graph. It could be useful when the filter graph manager itself creates a filter graph slowly because of lack of knowledge about custom filter graph that a developer possesses. However, the interface did not result in a performance increase. Finally, the interface was removed in order to preserve more flexibility.

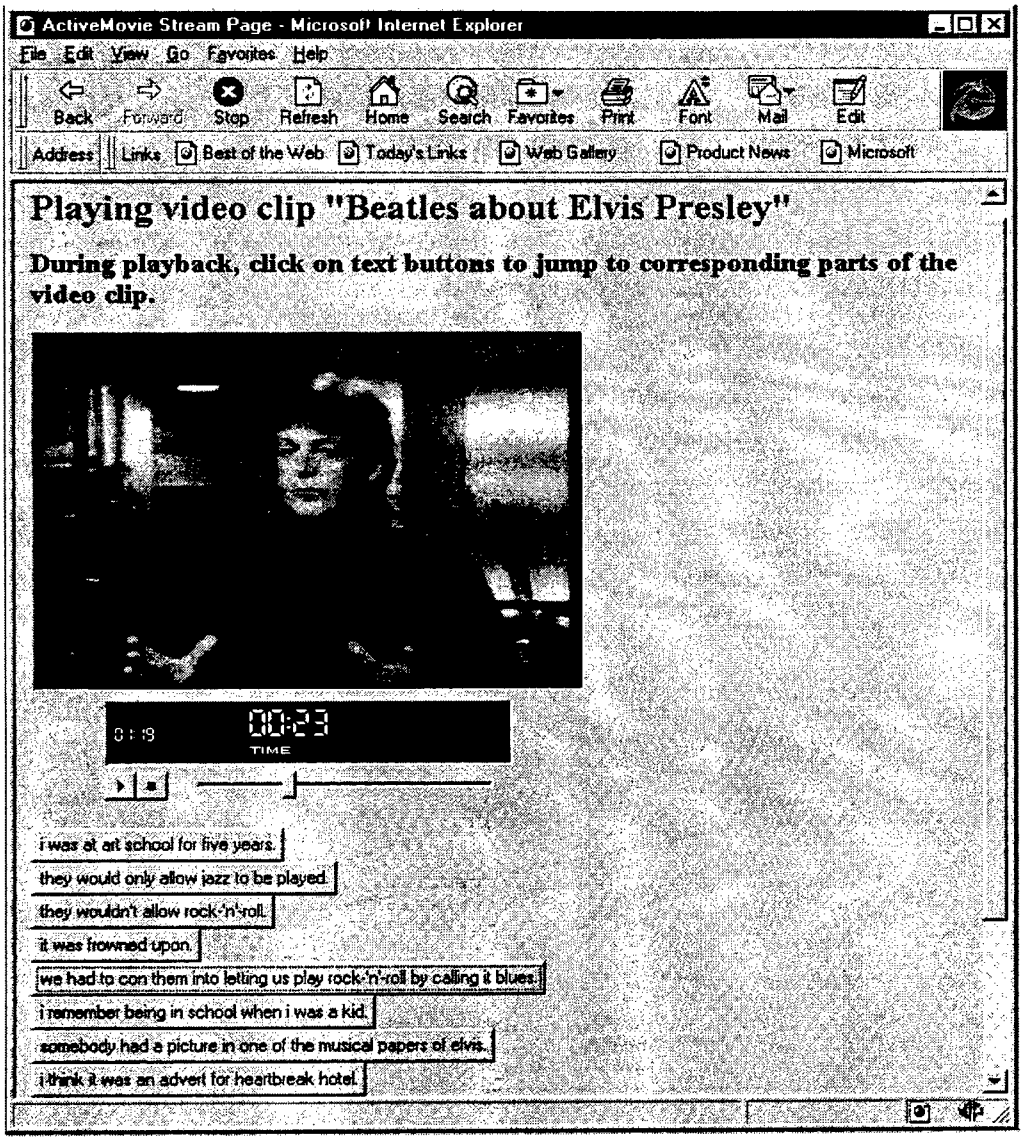
When the filter graph is created, the metadata file is parsed, and the connection to a Video Server established, the Visual Basic Video Client application performs an initial seek to the requested starting position. The control moves from the top-level application to the Video Client source filter which sends the Seek command to the Video Server. Beginning from the particular offset, the video data is streamed and the display starts when the video data buffer is full enough.

The figures illustrating the functionality described above can be found in Subsection 3.2.3, Figures 3..4 - 3.6.

5.3.2.8 Interaction with web browsers

The current Video Client cooperates with Netscape browser. A sequence of HTML pages generated by CGI scripts provides an end user with a search interface for the metadata database. This is a basic WWW functionality and could be represented by any WWW server and browser. After the uses selects a video clip, a CGI script generates a multipart HTML document, consisting of two sub-documents of two different MIME types: vod/nvs and text/html. The first document is the metadata file, the second document is a HTML page to be displayed in the browser with close-caption text. The close-caption text page provides a useful interface for searching video data material. This functionality enables the end user to move to a particular position in the video clip just by clicking on the close-caption text. The close-caption text is time-stamped and this is why the synchronization between the close-caption text and the video is possible.

The seeking functionality of the close-caption text page in Netscape is again implemented through a CGI script and a helper application. The CGI script generates a frame offset related to a close-caption text sentence. The offset is put into the file sent to the Netscape browser, with a vod/ofs MIME type. The seeking application associated with the MIME type uses a Windows API to perform the actual seeking of the required position in the movie. The seeking application simulates movement of the slider on the ActiveMovie OLE Control. But the presented solution is not very elegant. To enable direct communication between the Netscape's user interface and the Video Client application, a Netscape plug-in must be built.



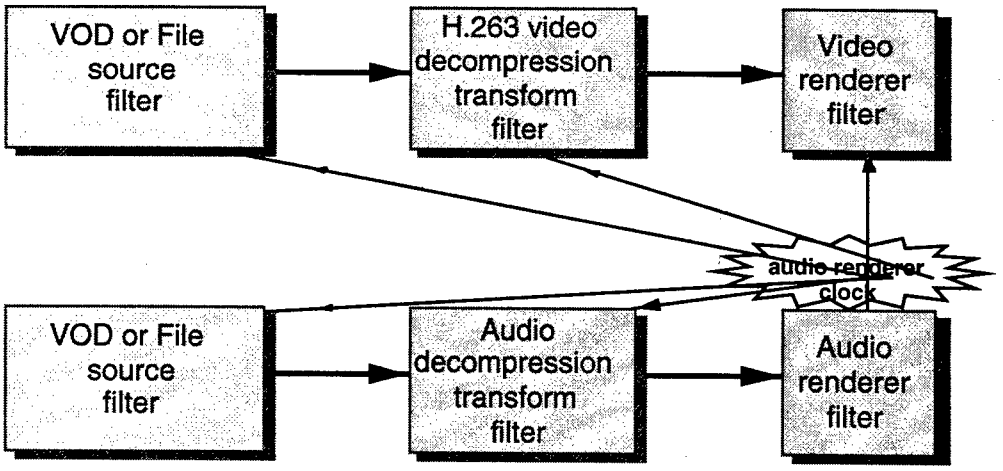
A simpler solution is possible using MicroSoft's Internet Explorer browser. The ActiveMovie OLE Control is an ActiveX object and can be incorporated within an HTML page. ActiveX uses the standard OBJECT tag to insert ActiveX objects into the HTML page, and Internet Explorer is able to activate and display the Control because it was designed as ActiveX container. There is no need for additional plug-in software, and all user interfaces can be included in one HTML page. Moreover, they can talk to each other using Visual Basic Script or Java Script, as illustrated in the preceding figure.

5.3.2.9 Designed Extensions to Video Client

The Video Client in the ActiveMovie architecture is easily extensible. It can make use of new hardware and software codecs incorporated in the ActiveMovie architecture as transform filters. There is a new MPEG software decoder available at NPAC, which will be ported to ActiveMovie;

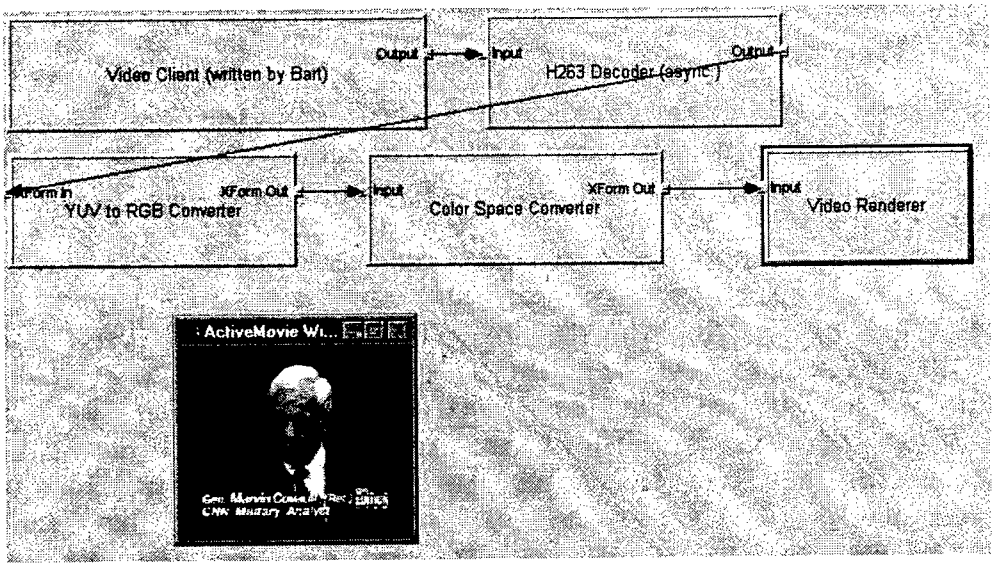
and the new network MPEG filter graph will include the transform filter of this decoder. Actually, the new MPEG solution could be inserted into ActiveMovie by replacing one, two, or three filters of the filter graph. The filter that will be replaced is the MPEG video decoder filter. The MPEG audio decoder filter and MPEG parser filter could remain unchanged since replacing them cannot improve the quality of decompression or shorten decoding time significantly. But each of those two filters could be also replaced with new ones. The only condition is that they should always conform to standard COM interfaces specified by ActiveMovie.

Another, more interesting extension of Video Client in ActiveMovie is building a H.263 video decompression transform filter and constructing such a filter graph that will perform synchronization tasks between H.263 video and associated audio. We already have the H.263 video decompression transform filter working with Video Client source filter. The next step is building an audio decompression transform filter that will handle the appropriate audio compression format. The best solution would use the audio codecs already installed in the Windows system.



The synchronization of H263 video with audio is possible due to the `IReferenceClock` interface exposed by the standard audio renderer filter. The video renderer filter synchronizes to the audio renderer filter simply by calling methods of the `IReferenceClock` interface on the audio renderer filter. The audio renderer is generating timer events which are distributed to all filters in the filter graph that asked for the `IReferenceClock` interface.

We designed the Video Client to enable multiple copies of itself running simultaneously in ActiveMovie. It can build the network H.263 filter graph with synchronized audio using two Video Client source filters. One Video Client source filter is streaming video data, the other is streaming audio data. They may, but do not have to, stream video and audio from two different Video Servers.



The figure above illustrates the ActiveMovie filter graph playing an H.263 video stream from the NPAC video server.

5.4 Web-specific video client architectures

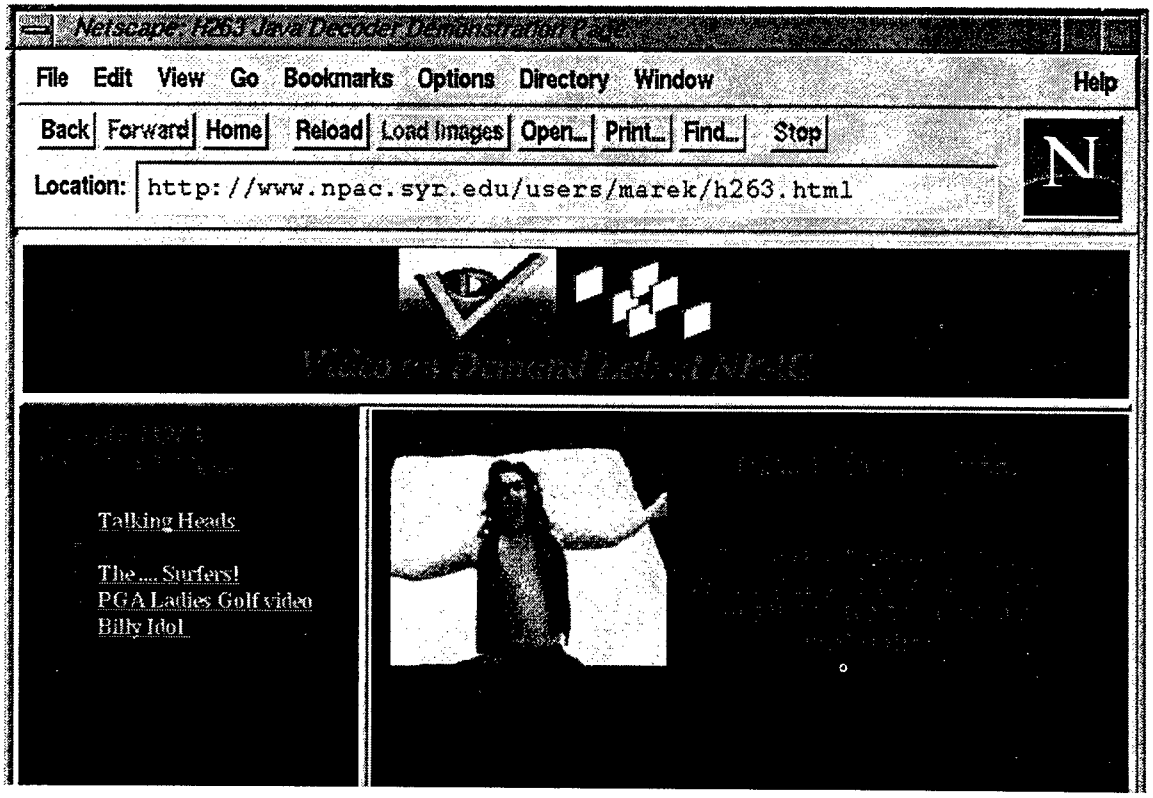
The video client implementations described above follow the model of the Web browser "helper applications." This is not the only model one can implement. Video playback applications can also be implemented as both browser plug-ins and even as Java applets.

Video client implementation as a Netscape plug-in is perfectly possible. We have implemented the H.263 decoder in this form. Two implemented versions of the plug-in have played video from the NPAC video server and from the web server streaming video over an HTTP link. For low bit rate codecs and in situations where interactivity is not essential, video streaming from an HTTP server to a viewer implemented as a plug-in is perhaps the simplest solution. We have observed that it is rather difficult to implement tightly controlled real-time playback in plug-in video clients. Plug-in code executes within Netscape browser, and the browser controls scheduling. It is therefore very difficult to effectively implement critical sections. We would not recommend this mode for CPU-intensive decoders.

Another rather interesting variation is a video client implemented in Java. The main challenge for this solution is performance: Java code is significantly slower than C code. However, with advent of Just-in-Time (JIT) compilers, the performance gap is closing, especially on PC platforms (e.g., Java run-time support on Microsoft Internet Explorer is almost as fast as native C code). This development opens new possibilities for multimedia programming. For a video client, Java is tremendously important since its virtual machine guarantees platform independence. At present, a full Java implementation of a video client playing synchronized audio and video is quite challenging, since the Java support for audio and video is limited. In particular, the Java VM only

knows how to play audio from .au files; user space buffer playback is possible only through an unsupported package from Sun. However, these difficulties are being remedied very quickly. Java 1.1 SDK already supports media interface with much more advanced capabilities. We fully expect that within a year it will be possible to re-implement the entire system we have described in this report entirely in Java, including video server and video clients, as well as the database access.

To demonstrate feasibility of this approach we have implemented an H.263 video client in Java:



The Netscape browser above plays an H.263 stream using a Java applet. The applet communicates with the NPAC server in this case, but it can also receive the H.263 stream from the web server over HTTP. The applet is able to play up to 4 frames per second within Netscape and up to 10 fps within Internet Explorer (because of IE's support of the JIT Java compiler, which Netscape 3 does not support). 10fps is the full frame rate for the streams we have used in experiment, so it is clear that Java can play low bit rate streams such as H.263 at full speed already.

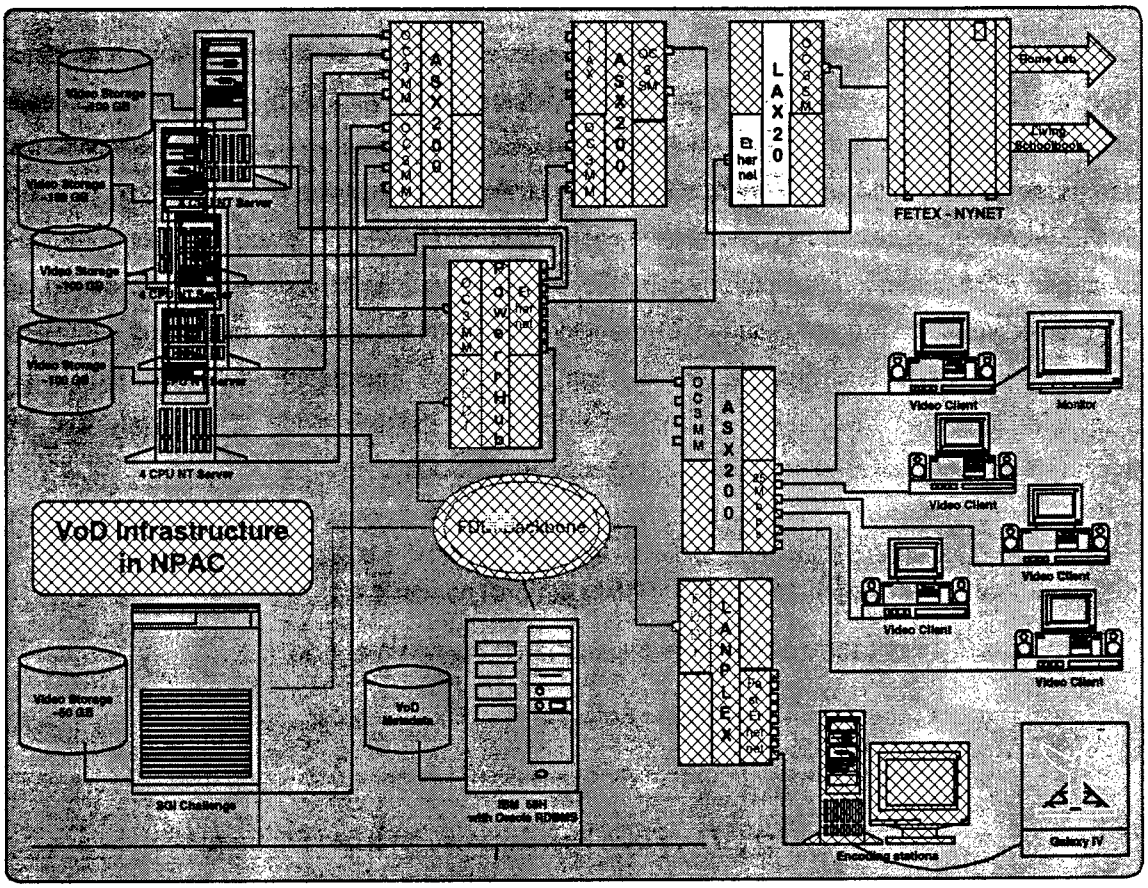
5.5 VoD versus LAN video broadcast: hierarchical multicasting video client

Another very important technology that continues to gain support and popularity is multicast. Our current release of the video server uses TCP/IP and hence does not support multicast. In general, multicast is more useful for broadcast type applications than for on-demand architectures. However, educational applications of the video server could benefit from multicast. Imagine an

application in which teacher wishes to present a video clip in the virtual class. The teacher wishes to retain interactive control over the stream but would like students to see the same video material in real time. Conventionally, on-demand and LAN TV servers are different and support different protocols. However, we believe this is am unnecessary complication. The architecture we have implemented consists of an enhanced ActiveMovie video client which, in addition to its fully interactive playback, also acts as a retransmission agent. The video stream from the server to the enhanced client is delivered in unicast mode. The client re-packetizes the video stream and multicasts it. The simplified video clients on student machines tune themselves to the session multicast address and play the incoming video stream. When the teacher pauses the stream or selects parts through random access, the students can follow. The entire ensemble is a part of a larger collaborative environment being developed in the CIV project. This functionality is completely implemented. We would like to note that the multicasting video client closely follows the RTP model for a translator entity.

6. VoD Testbed Infrastructure

6.1 Video Server assets



〈 Figure 6.1. NPAC video server assets

The main assets of the NPAC video server consist of the following items:

- ALR servers, 4 CPU each, Pentium Pro 200, 128 MV RAM, 3 SCSI channels, ~100 GB HD each machine, connected via OC3 ATM link to the ATM LAN
- 1 Micron 2 CPU Pentium 133 MHz, 64 MV RAM, 1 SCSI channel, ~20GB HD, connected via OC3 ATM link the ATM LAN
- 10 PC Pentium 166 MHz to Pentium Pro 200 MHz workstations, attached via either 25 Mpbs or 155 Mpbs ATM links to ATM LAN

6.2 NPAC ATM LAN

On October 22, 1996, FORE Systems officially announced implementation and support of LAN Emulation (LANE) version 1.0 in its products. LANE enables existing LAN protocols to operate over an ATM network.

LANE enabled us to create virtual LANs--a logical association of clients sharing a common broadcast domain. An intelligent Broadcast and Unknown Server (BUS) reduces unnecessary broadcast traffic and implements multicast. LAN Emulation Server (LES) maps native LAN addresses to ATM addresses. With the release of ForeThought v4.0 internetworking software, FORE implemented all the baseline features of ATM Forum LANE 1.0 giving us an opportunity to create an ATM-based infrastructure required for video- server/clients interaction. To proceed with the project, we purchased all required ATM equipment (switches, adapters, hubs) and seamlessly integrated our local ATM network with existing LANs. This equipment offered immediate and significant improvements in performance, flexibility and maintainability. It is a pity that this improvement was delivered only after the project was concluded.

The current ATM infrastructure at NPAC consists of

- ATM switch ASX200/U Software ForeThought 4.1.1
- ATM switch ASX200 Software ForeThought 4.0.1_1.20
- ATM switch ASX200WG Software ForeThought 4.0.2
- ATM switch ASX200WG Software ForeThought 4.0.2
- PowerHub 7000 (ATM, FDDI, Fast Ethernet, switched Ethernet).

The LAX20, which is still used to support Internet access for the Living Schoolbook Project, will be replaced by an ESX3810 installed in Syracuse University's School of Education.

Currently, NPAC's ATM LAN includes 27 ATM client workstations:

- 8x UltraSparc2 , OC3c/mm fiber
- 3x SGI Indy, GIA200 OC3c/mm fiber
- 6x G6-166/200, PCI-LE ATM25 (25.6 Mbps, UTP5)
- SGI Onyx, VMA200 OC3c/mm fiber
- SGI Indigo2, ESA200 OC3c/mm fiber
- SGI Challenge, VMA200 OC3c/mm fiber
- 5x P5-166, PCA200 OC3c/mm fiber
- 2x 4 CPU ALR, PCA200 OC3c/mm fiber

On top of this physical infrastructure, we implemented three instances of Emulated LANs with LES/BUS servers installed on one of ASX200 switches. We also implemented three instances of Classical IP over ATM (RFC 1577) in NPAC, between NPAC and the Museum of Science and Technology, and between NPAC and the Maxwell School. For the NYNET connections we use RFC 1483 encapsulation of bridging over PVCs, as the CO Fujitsu switch does not support LANE.

NPAC delivers ATM connection for other University Departments including the CASE center, the Maxwell School, and the School of Education. The integration capability is supported by the efficient networking of geographically remote computers by interfacing NPAC's internal LAN to the NYNET high-speed network. In near future we plan to obtain connectivity to vBNS, the very high speed Backbone Network Service.

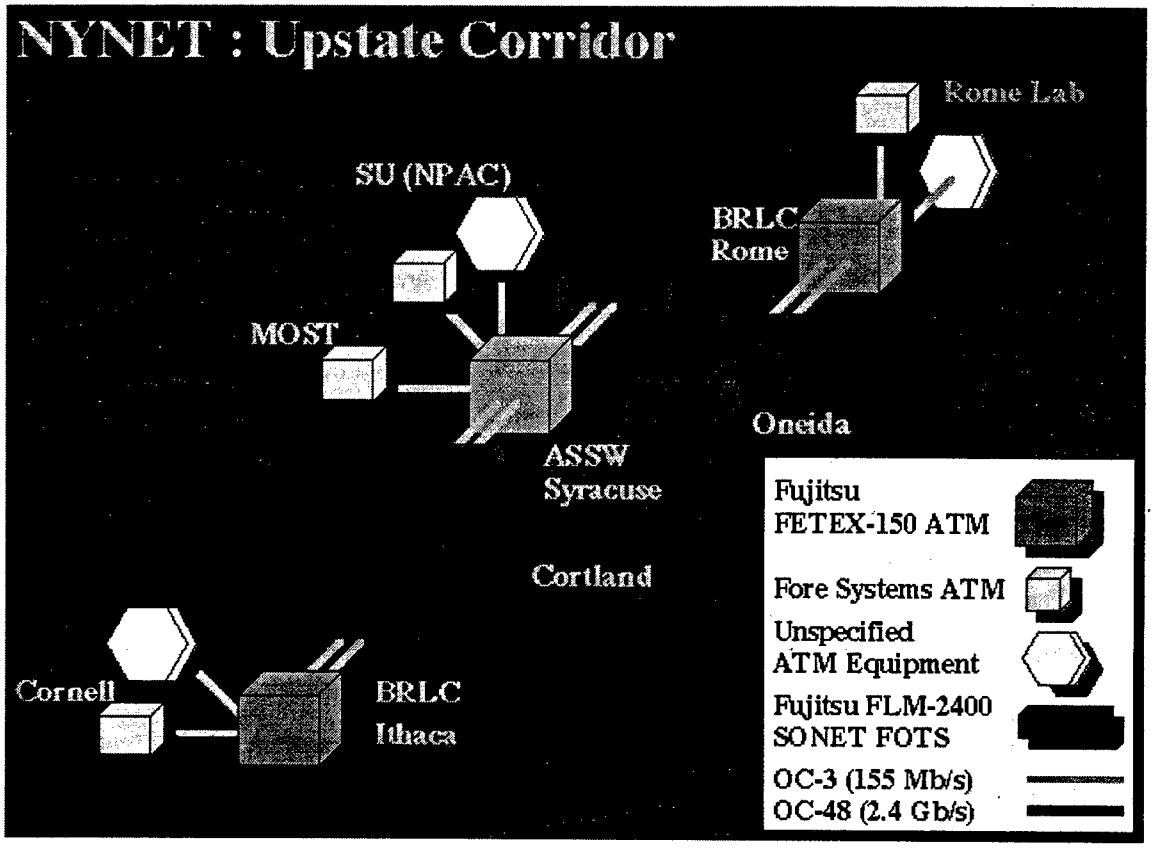
6.2.1 Living Schoolbook Project

The Living Schoolbook Project (LSB), a prototype of the Education Infrastructure of the Future, has been operational in the 24x7 mode for almost 2 years. It is an ATM-based network interconnecting Fowler High School, Rome Free Academy and Whitesboro Middle School with NPAC and the School of Education. The current network operates in the bridge mode on a few LAX20s interconnected via a complex mesh of PVCs. The network shows rather low performance, not acceptable for time-sensitive applications. A new LSB infrastructure is being built in collaboration with SUED and Rome Laboratory, and is design as state-of-the art information and collaboratory network to deliver video, images, and text to the classroom.

Currently we are in the process of upgrading the infrastructure, which will enable us to run an Emulated LAN network. In order to change the infrastructure, NPAC implemented 'lsb' ELAN on our equipment, and the School of Education purchased a FORE ESX 3810 Ethernet workgroup switch with 24 RJ45 Ethernet ports and an ATM uplink.

The same type of equipment will be installed at the schools. NYNEX plans to replace their Fujitsu switch with an ASX200. After these changes are implemented, the LSB network will operate in IP over an ATM environment using the LANE 1.0 emulation standard.

6.2.2 Rome Laboratory



〈 Figure 6.2. Architecture of NYNET for the NPAC - Rome Laboratory connection

Connection from NPAC to Rome Laboratory is provided via two FORE ASX switches. The connection uses one OC3 SM fiber, going over Fujitsu switch at NYNEX central office. Logically, PVC connectivity is used. PVCs between Rome Laboratory and NPAC are terminated not on the switches but on the routing workstations. These workstations, as routers and as gateways, provide packet translation between ATM and broadcast LANs. This solution will be upgraded to full LANE as soon as all elements are in place on both sites.

We provided demonstrations of the VoD service between NPAC and Rome Laboratory. The testbed used in these demonstrations is illustrated in Fig. 6.3.

During the project time frame, we conducted the following demonstrations of our VoD system over WAN ATM links:

1) Supercomputing '95, December '95, San Diego, running over NYNET and the vBNS backbone

2) HPDC-4, August '95, Pentagon City, running over NYNET, WilTel, and Bell Atlantic links

3) Interop, March '95, Atlanta, running over NYNET and WilTel links

4) HPDC-5, August '96, Syracuse, running over NYNET

5) Rome Laboratory, April '96, running over NYNET

6) Rome Laboratory, November '96, running over NYNET

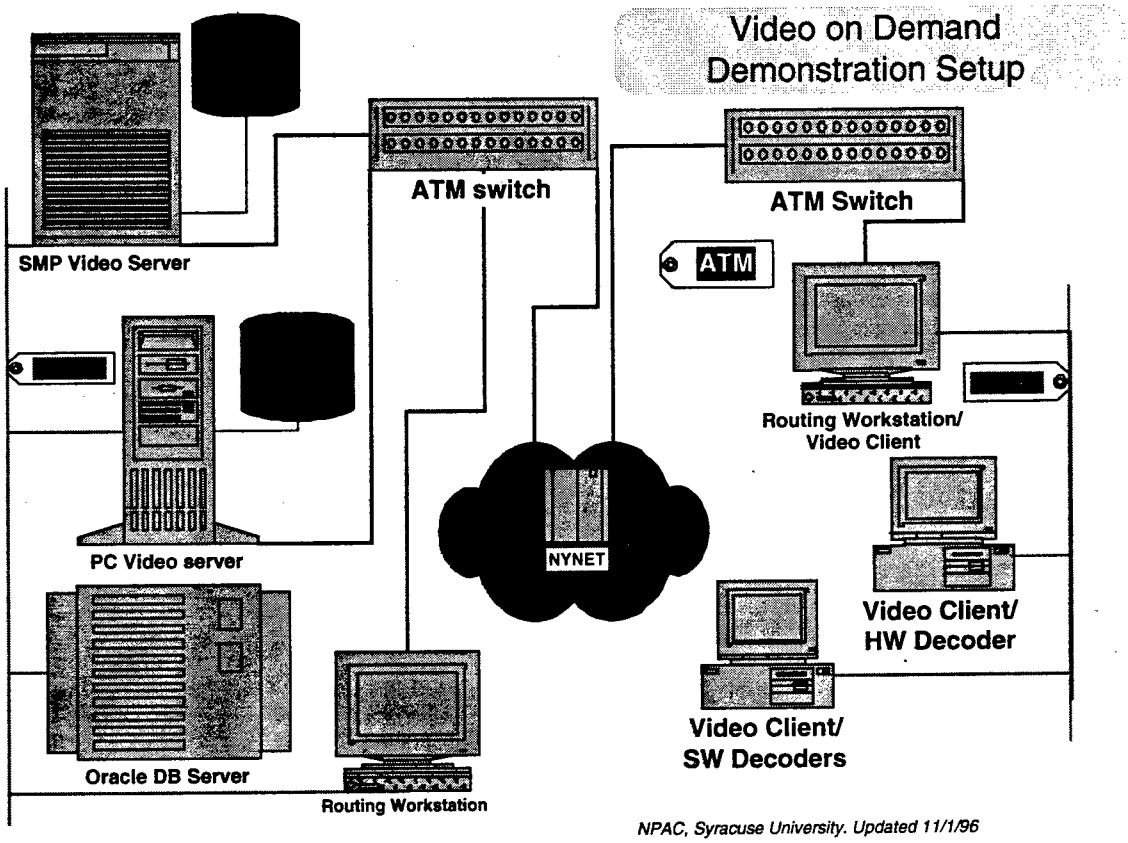


Figure 6.3. Video on demand demonstration setup between NPAC and Rome Laboratory.

All these demonstrations were quite successful. Full system functionality has been demonstrated in all cases, proving viability of the VoD over ATM WAN concept.

7. Selected VoD Research Issues

This Section of the report presents the results of our research into network video transmission, scaleable video coders, and video server architectures supporting adaptive applications. These results are largely theoretical, although some of the described algorithms have been implemented. None of the software was actually used in the implementation of VoD service. The work described in Subsections 7.1 and 7.3 is important for large VoD installations serving hundreds or perhaps thousands of video streams concurrently. The work on hybrid video coders is a proof that implementation of scaleable coders is possible even if today's CPUs are still too slow to decode such streams in real time.

Technically, the results described in Sec. 7.1 are a deliverable for Item 2 (Network Video Services) of the SoW. Results described in Sections 7.2 and 7.3 are deliverables for Item 1 (Video server architectures).

7.1 CBR transmission of VBR encoded video streams

In evaluating network performance, a consideration of the expected network traffic characteristics is of paramount importance. Much research work has been devoted to the study of video transmission using ATM networks {Ott:92},{Heeke:91}, {Heyman:91}, {Ramamurthy:90}, {Morrison:90}, {Shino:90}, {Sen:89}, {Nomura:89}, {Maglaris:88}, {Verbiest:88}.* The primary issue addressed in the present project concerns the performance benefits and trade-offs of using variable bit rate (VBR) encoding schemes versus constant bit rate (CBR) encoding schemes.

In CBR-encoded video sources, picture quality parameters are adjusted to maintain a constant requirement for delivery. As a result, although the video traffic can be transmitted via a fixed, reserved bandwidth, degradation of picture quality often occurs during encoding. In VBR encoding, the video source is encoded with a constant picture quality. This results in a variable number of bits from frame to frame.

To maximize the efficient use of network bandwidth, it is necessary to multiplex several video sources onto the same transmission channel. ATM is an attractive choice as a variable bit rate transport mechanism because of its ability to provide variable bandwidth dynamically (through statistical multiplexing). Obviously, multiplexing several VBR sources by simply reserving the maximum bandwidth required by each source results in a very inefficient use of the network bandwidth. However, problems start to arise when VBR sources are multiplexed without peak bandwidth reservations for each. Under these conditions, the source bandwidth requirements and available network bandwidth fluctuate independently of each other over time. Such fluctuations result in congestion at the network switches which, in turn, cause cells to be queued, then delayed, and sometimes dropped. Furthermore, because the fluctuations (and thus the degree of

* These references can be found on pages 277, 294-296, and 325-326.

congestion) occur randomly over time, the switch delay, aside from causing overall end-to-end delay, also results in increased jitter (variations in end-to-end delay) in the transmitted video stream. Although cell loss is undesirable because it leads directly to picture degradation, it can be tolerated to some extent. However, real-time video sources are extremely delay-sensitive and have very strict bounds on delay tolerance. Jitter leads to choppiness in the output as well as possible loss of synchronization for phase sensitive streams.

A measure of the effectiveness of VBR transmission schemes has been formulated by Heeke {Heeke:91}* and is called the *statistical multiplexing gain*; it is defined as the ratio of the number of multiplexed VBR sources to the number of multiplexed CBR sources while maintaining an equivalent subjective picture quality. Finding a solution to the problem of multiplexing VBR signals over a single transmission channel remains an active area of research. Several solution approaches have been taken, including multilayer bandwidth allocation {Hui:88}, switch-level error control and multiplexing {Dempsey:94}, "smoothing" of the video source {Lam:94},{Ott:92}, {Dagnino:91},{Knightly:94}, and stream rate adjustment {PanchaInfocom:92},{PanchaGlobecom:92}, {PanchaTR:92}.*

Delivering VBR encoded video streams at other than variable bit rates requires some measure of buffering at either the server or client processors. In a recent experiment {Hsieh:94},* a large VoD storage server was designed to contain up to 768 MBytes of main memory, and RAID storage devices. Although the storage system could support delivery of up to 86% of the maximum theoretical number of concurrent streams, the reported number of concurrent streams supported reached only 30% of the theoretical maximum due primarily to server memory space limitations.

An alternative approach {delRosario:94b}* employs buffering at the client process to achieve CBR transmission of VBR data. This strategy increases the maximum number of streams that can be multiplexed by a factor of 4.6 to 9.9 times. However, most of the buffering is placed at the client process. In a set of sample video streams, it was shown that from 3.7 to 14.6 Megabytes of buffer memory may be required. For multimedia PCs, currently being marketed with 64, and 512 Megabytes of RAM, the required buffer sizes amount to between 5.7% to 22.9%, and 0.65% to 2.9%, respectively, of the available memory. Although these buffer sizes may seem reasonable for larger capacity display devices such as workstations or PCs, they are still expected to greatly exceed the storage resources likely to be available in the first generation of set-top boxes.

Our preceding discussion makes clear that any strategy for video delivery in video on-demand systems must incorporate expressions for both transmission rate and buffer size components. Motivated by these issues, we propose a framework for providing end-to-end delivery of variable bit rate encoded continuous media in VoD servers.

The detailed discussion of the results of this research effort are presented in three articles. All three papers are integral part of this report and they have been placed in Appendix 1. Two of the

* These references can be found on pages 277, 294-296, and 325-326.

papers deal with VBR-to-CBR conversion and client-side buffer optimization. The third paper presents a more advanced scheme, m-frame granular transport. M-frame transport assumes variable quality of service and additional buffering on the server side.

7.2 Hybrid wavelet - H.263 video compression

In the Integrated Services Model, applications are capable of constantly monitoring network conditions and adapting its own behavior in relation to these conditions. Future adaptive applications will demand a new family of video codecs. Multi-resolution embedded codecs are particularly important since they capable of providing for graceful degradation by enabling the application to respond to deteriorating network conditions by sending video stream of a lower quality but with significantly lower bandwidth. In Subsection 7.3 we discuss the architecture of a file system supporting such a codec. This section is concerned with a feasibility study of implementing a scaleable codec using hybrid wavelet - H263 technology. Wavelet compression is inherently scaleable and multi-resolution, but is a poor choice for motion compensation. The hybrid scheme takes the best features from the two different compression methodologies and combines them. We have designed and implemented such a codec and have demonstrated that the quality of the resulting video stream is superior to the H.263 quality at the same bit rate. This finding is valuable *per se* since H.263 is one of the best available video codecs. We have also analyzed the performance of the codec and found it slower than the H.263. However, expected increases in CPU speed will make this codec a viable alternative in very near future.

The detailed report describing results of this research effort is enclosed as Appendix 2.

7.3 A continuous media file system for multi-resolution data

Currently, video server design is regarded as a system integration issue where diverse components are haphazardly plugged together. In reality, video servers are complex entities that must address the requirements of individual video streams as well as system-wide requirements. The storage subsystem constitutes a major component of any media server. Yet the file system support for continuous media is, at present, quite limited.

The traditional approach of Quality of Service issues is network-centric. Adaptive applications (i.e., applications that change their behavior in response to changing network conditions) are being proposed. We observe, however, that without architectural changes in the media, the potential benefit of adaptivity is limited. Consider, for instance, a video server capable of adjusting bitrate of outgoing media streams by sending only a part of the video stream encoded by a multi-resolution coder. Using the current file system architectures, the server will have to read the entire stream from the disk before discarding the information that is not being sent. In this section we present an

idea of a video server architecture that provides end-to-end support for such an adaptive application. The proposed Continuous File System (CFS) also provides explicit support for isochronous data.

A full text of this report is attached in Appendix 3.

8. Multimedia Networking: Integrated Services Model

In the preceding sections, we have mentioned on several occasions the Integrated Services Model for packet networks, which provides a blueprint for delivery of multimedia information over packet networks. This model introduces the notions of Quality of Service, resource reservation, stream synchronization, and multicast. We studied these issues very thoroughly during the project time.

The elements of the Integrated Services Model were not mature enough to be incorporated in actual implementation of the VoD infrastructure. However, when designing elements of the system, we have had in mind the future architecture of the Internet for multimedia transport. We believe that our entire system can easily be enhanced to support protocols such as RSVP and RTP, as well as to use multicast. As a matter of fact, the multicasting agent described in an earlier section uses multicast and is, in itself, an example of the RTP translator.

Our study and evaluation of the multimedia networking issues has been documented in a series of lectures that have been delivered in NPAC in form of internal seminars, as well as during the quarterly progress meetings with the Rome laboratory personnel. These lectures form a self-contained unit that presents and discusses the most important features of the integrated services model. The following section contains the slides of all the relevant presentations developed during the project.

The following modules are included in this section:

1) Introductory multimedia networking module (elementary level)
2) Multimedia networks and Integrated Services Model
3) Multicast and switched network basics
4) Reservation Protocol, parts I and II
5) Real Time Protocol.

Multimedia Networking: Elementary Introduction

Basics of the Data Networking

Marek Podgorny

marek@npac.syr.edu, (315) 443-4879 http://trurl.npac.syr.edu

Data Networking

- *Data networking* term refers to exchange of digital information between remote systems
  - Data can be exchange between any kind of devices
- *Computer networking* is a special case of *data networking*
  - While communication between computer components over a bus is clearly a case of digital data transmission, this is not considered data networking example
- Differentiating factors: media, parallel vs. serial, distance, protocol standardization

Physical Data Transmission

- Terrestrial media: metallic cables and optical fibers
  - metallic cables: coaxial cables and twisted pair cables
    - carrier: electrical pulses
    - problems: signal attenuation, noise, bandwidth and length limitations, price
  - optical fiber
    - carrier: modulated light
    - problems: expensive end optic
    - advantages: high bandwidth, sleuth-resistant, long distance

Physical Data Transmission

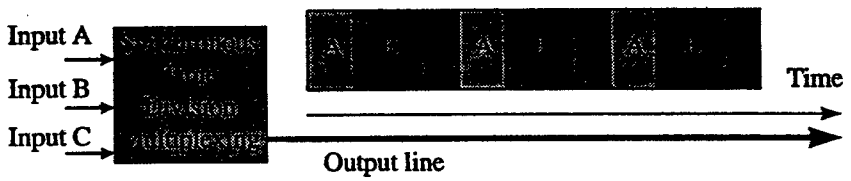
- Aerial systems: surface transmission and satellite transmission
  - both less reliable (higher error rates) as compared to cable/fiber
  - high latency (~540 msec) for satellite transmission lines
  - limited frequency domain and, hence, global transmission capacity
  - significant chunk of global capacity assigned to analog transmission

Baseband vs. Broadband

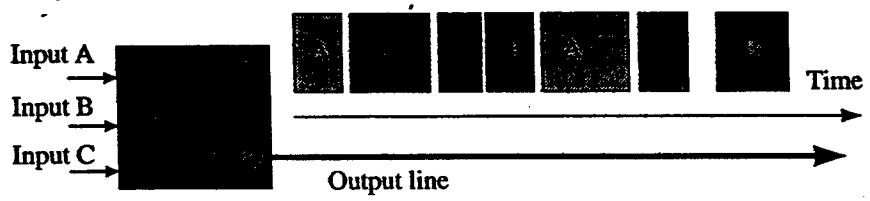
- Baseband transmission
  - Electrical signal applied directly between the wires
    - example: Ethernet technology
  - Basic limitation: only one bit can be transmitted at any given time
  - Overall efficiency increased by Time Division Multiplexing (TDM)
    - Why? Idle time costs money....
    - TDM flavors: synchronous and asynchronous
  - Signal attenuation is a big problem for baseband networks

Time Division Multiplexing

- Synchronous TDM: moderately efficient



- Asynchronous TDM: more flexible and efficient



Broadband Transmission

- Signal is sent by modulating a *carrier*
- Applicable over both cable and fiber
- Amplitude and frequency modulation
- In data transmission world, Frequency Division Multiplexing (FDM) term is used
- Using narrow band filters, receivers are able to separate multiple signals
- Using FDM, multiple transmissions may be concurrently sent over a single cable/fiber
- Example: a *modem*

Circuit vs. Packet Switching

- Circuit Switching Networks
  - Originally, a connection was set up by physically connecting segments of physical wire to form a continuous electrical circuit.
  - At present, the circuit is formed using TDM in the synchronous mode
  - Circuit switched network guarantees the bit rate and, usually, constant, predictable latency.
  - Downside: cost of the connection depends on the reserved bandwidth, not on actual amount of data transferred over the circuit. No statistical multiplexing.
  - Examples: phone lines, ISDN lines

Circuit vs. Packet Switching

- Packet switching networks
  - Multiplex several data streams more efficiently over single wire/fiber/switch
  - Information chopped into small units (packets) sent one a time
  - Many sources of packets can coexist attached to one transmission medium
  - Multiplexing is *statistical* and network behavior is not entirely predictable
  - Packets carry their destination addresses in headers
  - Examples: Ethernet, FDDI, Frame Relay, IP

Are the examples right?!!!

- Ethernet and IP both listed as packet switching networks - is this right?
- YES - the are both packet switching albeit in different layers!
- This leads us to the concept of the layering
- Networks run on top of other networks; protocols run on top of other protocols.
- Layering is used to separate network functionality into logical entities.

Layering - an example

- Secretary-assisted phone conversation
  - Need phone set with right connectors and internal electronics. This is layer 1 - physical/electrical
  - Need microphone (voice to electrical signal) and forwarding mechanism to pass voice generated electrical signal end to end. This is layer 2 - link
  - Need keypad/rotary and a way to setup connection. This is layer 3 - network

Layering - an example

- Need secretary to get another secretary, to agree on the language for their conversation and to request your interlocutor. This is layer 4 - transport
- Need a common language for your own conversation. This is layer 6 - presentation
- Now you can talk business - this is layer 7 - application
- Each layer depends/rides on top of other layers.

Connection vs. Connectionless

- Packet vs. circuit switching determines how information is routed once inside the network
- Connection mode determines under what condition data can be accepted
- Connection-oriented networks are aware about two systems communicating. The network must admit a communication stream before data exchange starts
- Connectionless network has no idea that two systems communicate. Connection is never refused

Connection vs. Connectionless

- Connection-oriented network
  - Implementation: virtual circuits, signaling protocols
  - Advantages: more predictable traffic
  - Disadvantages: need call set-up which introduces delay; wastes network resources
- Connectionless networks
  - Implementations: addresses in packet headers
  - Advantages: no call set-up delay, no busy signal
  - Disadvantages: difficult to reserve resources

Confusing?

- Very much so. The features listed above can be combined in every possible way, leading to an extremely complex topologies and interdependencies
- It is possible to have connectionless and circuit switched network, as well as connection oriented packet switching network
- It is possible to have asynchronous services running over synchronous carriers (ATM)
- It is possible to combine multiple contradictory features over the same physical medium

Popular Network Topologies

- Star/tree/mesh topologies.
  - Used mostly for circuit switched networks (then...)
  - Also used to interconnect broadcast networks (now...)
  - Support packet switching layers....
- Bus networks - shared media
  - All connected workstations see all information
- Ring networks - shared media
  - Use broadcast but assign time slots differently
- Internetworks connect all of the above

Case study: Ethernet

- Bus topology, connectionless, baseband, packet switching
- CSMA-CD (*Carrier Sense Multiple Access - Collision Detection*) principle - contention protocol
  - Only one station can send at any given time
  - Automatic signal detect (remember baseband?)
  - Attached stations send at randomized intervals
- Fixed-length frames (1516 bytes)
- Very unpredictable. CSMA-CD leads to exponential saturation.

Case study: Token Ring

- Ring, packet switching, connectionless, baseband
- Unidirectional transmission, token circulates at full speed
- Token may be either free (no data) or busy (data attached)
- Station wishing to transmit waits for a free token. It sets it to busy and attaches data with header
- The addressee copies the data and sets "data copied" bit
- Originating station sets token to free and removes data

Case study: FDDI

- Ring, connectionless, packet switching.
- FDDI tokens are "captured" and "reissued"
- Multiple frames can be in transit concurrently
- "Token holding time" may support priorities
- "Token rotation time" limits ring latency
- FDDI uses larger frames (4kB) and supports bandwidth of 100 Mbps (in practice, ~60 Mbps)
- Token ring protocols are "non-contention" and do not suffer from exponential saturation.

Network Architectures for Multimedia Delivery

Networked Multimedia Applications and their Impact on the Network Services

March 1996

Marek Podgorny
NPAC
Syracuse University
111 College Place
Syracuse
New York 13244-4100

Abstract of Network Architectures for Multimedia Delivery

- Review of the networked multimedia presentations
- Requirements for the networking infrastructure supporting multimedia applications
  - Application categorization
  - Relevant network performance parameters
- Network technologies supporting multimedia delivery
  - Multicast
  - Switching technologies
  - Quality of service guarantees (integrated services)
  - ATM networks and multimedia

Networked multimedia applications

- **Electronic documents with attached voice and video annotations**
  - (multimedia electronic mail, workgroup applications).
- **Desktop conferencing:**
  - two or more participants connected through the (inter)network,
  - using audio/video, whiteboard, shared applications...
- **Distance-learning courses offered as live broadcasts**
  - or as video-on-demand, supported by
  - On-line textual material, on-line assignments, shared project areas etc.

Networked multimedia applications (Continued)

- **Television broadcasts over local-area networks (LANs).**
- **Interactive kiosks with multimedia product information and demonstrations.**
- **Multimedia databases that store traditional text as well as images, audio, and video clips.**
  - Fully searchable digital libraries with instant delivery.
- **3D navigation in real and abstract landscapes.**
- **Simulation on demand with direct graphical output and interactive input.**

Multimedia Application Industry -- I

- **Hierarchical structure of the MM industry:**
  - **Multimedia content providers: news industry, television industry, entertainment industry, multimedia CD-ROMs manufacturers.**
    - All these companies seek ways to deliver their products over the net.
  - **Multimedia application developers: distance learning, desktop videoconferencing, workgroup collaboration, multimedia kiosks, entertainment, imaging, video on demand.**
    - All these applications need network support.

Multimedia Application Industry (Continued)

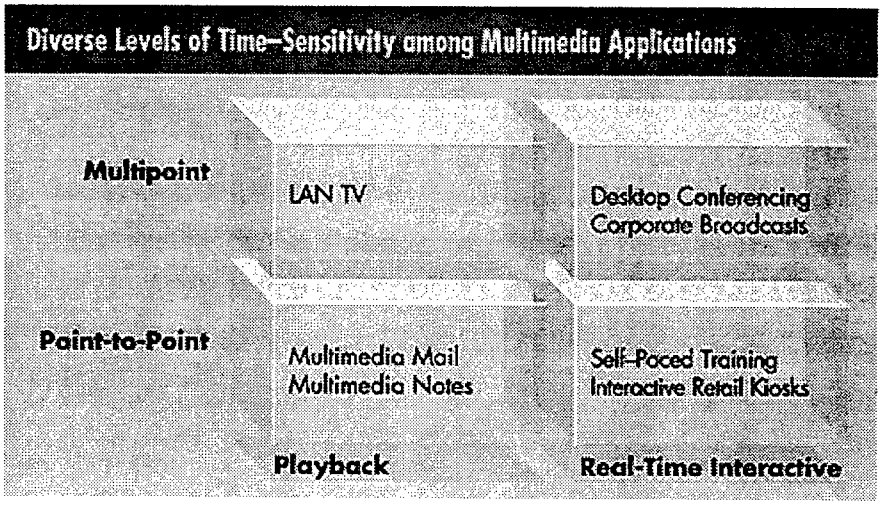
- **Multimedia platform builders: Of the computers sold in 1995, 80% was multimedia capable!**
- **Multimedia capable network infrastructure providers:**
  - **Networking of multimedia is expected to initially occur in businesses over the private networking infrastructure (IntraNets).**
  - **In 1996, a large demand is expected for multimedia applications across public networks such as the Internet for home, education, business, and entertainment.**
- **Our conclusion: Nearly all multimedia industry gears up to switch to networked applications.**

Categories of Net MM Apps -- I

- As shown on next foil, for delivery over an internetwork, multimedia applications can be grouped into categories according to
  - Their sensitivity to delay and the number of simultaneously connected network nodes.
- Applications that use stored data streams are not delay-sensitive.
- Applications that involve real-time interactivity are sensitive to network delay.
- Point-to-point applications involve communication between a single pair of network nodes.
- Multipoint applications typically involve a broadcast to many network nodes.

Categories of Net MM Apps - II

- Pictorial Representation of Time Sensitivity in Applications



Multimedia Kiosks -- I

- **Multimedia kiosk is an important business application!**
  - The financial industry wants to use multimedia kiosks at banks to provide detailed information about financial services.
  - The retail industry wants to use kiosks in stores to help customers locate merchandise and find out additional information about merchandise.
- **Stand-alone kiosk are expensive to maintain. Networking is an operational necessity.**

Multimedia Kiosks - II

- **The entertainment industry wants to use kiosks as**
  - points of sale and to provide
  - advertising for scheduled entertainment events such as the theatre, concerts, plays, etc.
- **The entertainment industry is also interested in multimedia kiosks that allow multi-person video games to occur over networks.**
- **Highway authorities plan kiosks with tourist information**

Multimedia Network Requirements - I - Overview

- **To provide a decent multimedia delivery infrastructure the network must provide three operational characteristics at an acceptable level:**
  - Bandwidth
  - Consistent quality of service
  - Efficient multipoint packet routing and delivery
- **Two latter requirements are specific to multimedia network traffic. In the past, MM traffic was very limited.**
  - With MM revolution just around the corner, the network providers and corporate IS departments face the task of general system reengineering.
- **Fortunately, the technological solutions are available.**

Multimedia Network Requirements - II - Bandwidth

- **Bandwidth requirements**
  - **BW requirements differ widely from application to application.**
  - **Type of applications and the requested quality are main differentiating factors.**
  - **Typical examples are shown on following foil**
- **Typical workstation on a corporate LAN has today available BW of 50 - 100 Kbps**
- **This bandwidth is only sufficient for simple MM applications**
- **Solution: switched broadcast networks - will be discussed later.**

Multimedia Network Requirements - III - Bandwidth (Continued)

- Some Examples of Needed Network Bandwidth

Phone quality audio	8 Kbps
HiFi quality audio	64 Kbps
Application sharing	~100 Kbps
Videoconferencing	128 Kbps - 1 Mbps
Low bit rate video	28 Kbps (I) - 128 Kbps
VHS (MPEG1) video	1.2 - 3 Mbps
HDTV (MPEG2) video	4 - 6 Mbps
Imaging	8 - 100 Mbps
3D rendering, VR	100 Mbps+

Multimedia Network Requirements - IV - Bandwidth Scenarios

- Typical scenarios for deficient bandwidth:
  - I: Desktop machines running MM applications
    - In this environment the most effective way of alleviating the bandwidth problem is network segmentation.
    - However, if there is significant server-related traffic, segmentation is ineffective and a higher BW carrier is needed

Multimedia Network Requirements - V
- Bandwidth Scenarios (Continued)

- **Typical scenarios for deficient bandwidth--Contd:**
- **II: Campus backbone**
  - For the campus backbone the only solution is a high-speed carrier. Current leading technology is FDDI but its position is being challenged by ATM. Taking this path is a very risky decision, though!
- **III: Wide area network**
  - Very expensive (recurring cost!). Eventual solution: bandwidth on demand (ATM, but also inverse multiplexing of more traditional channels).

Multimedia Network Requirements - VI - Quality of Service

- **Three major components in Quality of Service requirements:**
  - Guaranteed bandwidth (note this is different from available bandwidth)
  - End-to-end latency
  - Jitter (i.e. deviation from the average packet arrival time)
- **None of these factors was a design parameter of today's packet networks.**
- **Different applications running concurrently over the networks may have different service requirements: this imples need for an integrated service network able to handle data and stream applications.**
- **QoS support is sometimes referred to as integrated services.**

Multimedia Network Requirements - VII- Quality of Service (Continued)

- **Guaranteed bandwidth requirement: how good are packet-switched networks for different types of applications?**
  - Constant bit rate applications
    - **Difficult on packet networks. With deficient BW they break, surplus BW not useful. Large buffers on client side about the only remedy if QoS absent.**
  - Variable bit rate applications
    - **Via statistical sharing VBR applications are more efficient on packet networks than on circuit switched networks.**
    - **QoS requirements only help with sophisticated application-level network resource scheduling.**
  - Available bit rate applications
    - **Packet networks are ideal transport for ABR apps. ABR apps adapt to the momentarily available bandwidth.**

Multimedia Network Requirements - VIII -- Latency

- **For MM applications network latency requirements are in general less stringent than for compute intensive applications.**
  - Compute Intensive often requires short round trip times
  - The latency is measured on the time scale of human perception.
  - Latency is of little relevance for one-way transmissions (up to a certain limit...) but is a potential nuisance for teleconferencing and for shared applications.
  - For these applications the latency should be kept below 0.5 sec.

Multimedia Network Requirements - IX - Latency (Continued)

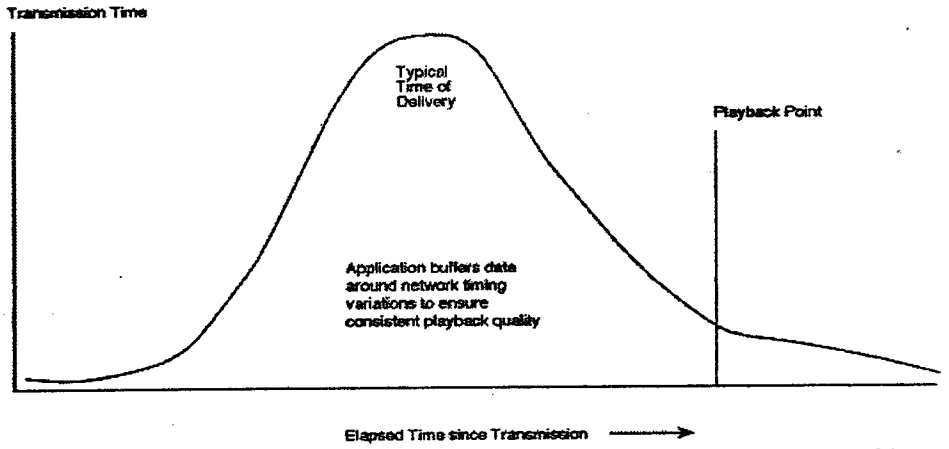
- **Latency issue is almost entirely under control (or lack thereof) of the networking gear vendors.**
  - Almost every operation on a packet contributes to the total latency.
  - Technologies such as cut-through (instead of store and forward) forwarding attempt to decrease the latency.
- **More heterogenous networks usually induce higher latency due to packets encapsulation/header changes or segmentation and reassembly.**

Multimedia Network Requirements - X - Jitter

- **Jitter - audio quality killer**
  - **Due to statistical factors, packets do not arrive in evenly spaced intervals. Instead, arrival time displays Guassian variation.**
  - **Jitter is a problem for both audio and video streams.**
  - **For audio, jitter may cause "hiccup" which is extremely annoying and adversely affects comprehension.**
- **For video, packets that arrive too late require complex logic in the decoder.**
- **Dropping the late packets may cause header loss and decoder confusion.**
- **Processing the late packets compounds the synchronization problems.**
- **Remedy: network buffers, Bandwidth reservation, packet priority handling, etc.**

Multimedia Network Requirements - X I - Jitter (Continued)

- **Typical Jitter Distribution**



Multimedia Network Requirements - XII - Multipoint Packet Delivery

- **One-to-many and many-to-many applications are extremely expensive in terms of both network bandwidth and processing power on both workstations and on the routers. Typically, unicast and broadcast mechanisms are used:**
  - Unicast: the sending computer can send out a different copy of the data for each recipient.
    - **Very wasteful of bandwidth, terrible scaling properties.**
  - Broadcast: the sending computer can send out a broadcast packet. The networking devices need to forward these packets to all portions of the network in order to ensure that they reach their intended recipients.
    - **Also very wasteful of bandwidth.**

Multimedia Network Requirements - XIII - Multipoint Packet Delivery (Continued) -- Multicast

- **Multicast: the sending computer can send out a multicast packet that is addressed to all the intended recipients. The network will replicate the packet when necessary.**
  - It is a very efficient process.
  - In order for multicast to work, the networking devices need to know which computers need to receive multicast traffic, and they need to be able to dynamically build efficient paths to all destinations.
  - Collective communication algorithms on parallel computers use this approach (if they are any good!)

Multimedia QoS Support -- Introduction

- **Global solution of the multimedia QoS problem requires broad consensus of the network gear vendors on how to implement necessary protocols and measures.**
- **All networking devices carrying the multimedia traffic must support the same set of mechanisms.**

Multimedia QoS Support -- General Mechanisms

- Among the mechanisms discussed for QoS Support, the most elementary are:
- Priority queuing: today's packet networks are truly democratic and egalitarian -
  - FIFO is about the only packet forwarding strategy.
  - Priority queuing could alleviate jitter problems.
- Custom queuing: this is a bandwidth reservation mechanism.
  - It is less drastic than priority queuing as it schedules MM packets based on the specified jitter upper bound, not on the absolute priority.

QoS -- ATM and Integrated Services Network

- Two possible solutions:
- ATM networking technology - QoS is a build-in feature for this type of transport. This is probably a solution for tomorrow's GII backbone. Will it work in near future?
  - 100% ATM network will never happen.
  - ATM far more expensive than broadcast networks; new fiber/UTP cable plant needed,
  - Serious management and maintenance problems unsolved.
  - ATM QoS does not provide global QoS support.

ATM versus Integrated Services Network

- **ATM Technology -- problems!**
  - ATM standards still in flux, implementations more so, LANE not universally available. Consequently, QoS mostly works on the paper.
  - There is no API to the ATM QoS mechanisms. Hence, applications do not have any way to request such services.
- **Solution -- Integrated services over the packet switched networks**
  - A new architecture on top of the current IP layer as opposed to ATM low level approach

Integrated Services for the Internet

- **In the following, we will discuss the emerging architecture implementing so called integrated services running on top of the existing Internet infrastructure.**
- **The integrated services infrastructure complements two other enabling technologies:**
  - **Client multimedia hardware:** many modern workstations now come equipped with built-in multimedia hardware, including audio codecs and video frame-grabbers, and the necessary video gear is now inexpensive.
  - **IP multicasting:** while not yet commonly available, it makes its way into Internet infrastructure. Integrated services architecture assumes multicast as a *sine qua non* component of future multimedia Internet

Integrated Services: Additional Rationale -- Bandwidth Allocation - I

- **Real-time QoS is not the only issue for a next generation of traffic management in the Internet: network operators are requesting the ability to control the sharing of bandwidth on a particular link among different traffic classes.**
  - They want to be able to divide traffic into a few administrative classes and assign to each a minimum percentage of the link bandwidth under conditions of overload, while allowing "unused" bandwidth to be available at other times.

Integrated Services: Additional Rationale -- Bandwidth Allocation II

- **These network administration classes may represent different user groups or different protocol families, for example.**
  - Such a management facility is commonly called controlled link-sharing.
- **Therefore, in the following, the term integrated services (IS) for an Internet service model includes best-effort service, real-time service, and controlled link sharing.**

Integrated services: Architecture Elements - I

- **The fundamental service model of the Internet, as embodied in the best-effort delivery service of IP, has been unchanged since the beginning of the Internet research project.**
- **Integrated services postulate a major change.**
  - From an academic viewpoint, changing the service model of the Internet is a major undertaking;
  - However, its impact is mitigated by the fact that we wish only to extend the original architecture.
- **The new components and mechanisms to be added will supplement but not replace the basic IP service.**

Integrated services: Architecture Elements - II

- **The proposed architectural extension is comprised of two elements:**
  - An **extended service model,** called the IS model,
  - A **reference implementation framework**, which provides a set of vocabulary and a generic program organization to realize the IS model.
- **It is important to separate the service model, which defines the externally visible behavior, from the discussion of the implementation, which will evolve during the life of the service model.**

Integrated Services Model - I

- **The IS model includes two sorts of service targeted towards real-time traffic: guaranteed and predictive service.**
  - It integrates these services with controlled link-sharing,
  - and it is designed to work well with multicast as well as unicast.
- **We now discuss several basic assumptions of the model:**
- **Resources (e.g., bandwidth) must be explicitly managed in order to meet application requirements.**
  - This implies that "resource reservation" and "admission control" are key building blocks of the service.
  - An alternative approach is to attempt to support real-time traffic without any explicit changes to the Internet service model.

Integrated Services Model - II -- Basic assumptions of the model

- **It is desirable to use the Internet as a common infrastructure to support both non-real-time and real-time communication.**
  - One could alternatively build an entirely new, parallel infrastructure for real-time services, leaving the Internet unchanged.
  - With this approach one would lose the significant advantages of statistical sharing between real-time and non-real-time traffic, and it would be much more complex to build and administer than a common infrastructure.

Integrated Services Model - III -- Basic assumptions of the model (Continued)

- **There is a unified protocol stack, employing a single Internet-layer protocol for both real-time and non-real-time service.**
  - The IS model proposes to use the existing Internet-layer protocol for real-time data.
  - Another approach would be to add a new real-time protocol in the Internet layer.
  - Unified stack approach provides economy of mechanism, and it allows one to fold in controlled link-sharing easily.
  - It also handles the problem of partial coverage, i.e., allowing interoperation between IS-capable Internet systems and systems that have not been extended, without the complexity of tunneling.

Integrated Services Model - IV -- Basic assumptions of the model (Continued)

- **There should be a single service model for the Internet.**
  - If there were different service models in different parts of the Internet, it is very difficult to see how any end- to-end service quality statements could be made.
  - Single service model does not necessarily imply a single implementation for packet scheduling or admission control. It is possible to implement different mechanisms that will also satisfy the service model.

Integrated Services Model: Unnecessary? - I

- **Arguments have been raised against an integrated service model based on the assumptions we outlined.**
- **In particular, the following arguments have been presented against the necessity of a complex reservation model:**
  - **"Bandwidth will be infinite."**
    - **This will be impossible in the short term and unlikely in the medium term.**
    - **While raw bandwidth may seem inexpensive, bandwidth provided as a network service is not likely to become so cheap that wasting it will be the most cost-effective design principle.**

Integrated Services Model: Unnecessary? - II

- **"Bandwidth will be infinite." (Continued)**
  - Even if low-cost bandwidth does eventually become commonly available, it is unlikely that it will be available "everywhere" in the Internet.
  - Hence, unless the network management provides for the possibility of dealing with congested links, then real-time services will simply be precluded in those cases.
  - In military applications, assume need for "Theater Extensions" with low bandwidth in major areas of activity linked to high bandwidth Global Information Infrastructure

Integrated Services Model: Unnecessary? - III

- **Arguments against integrated services (continued):**
- **"Simple priority is sufficient."**
  - While it is true that simply giving higher priority to real-time traffic would lead to adequate real-time service at some times and under some conditions, priority is an implementation mechanism, not a service model.
  - If we define the service by means of a specific mechanism, we may not get the exact features we want.
  - In the case of simple priority the issue is that as soon as there are too many real-time streams competing for the higher priority, every stream is degraded.
  - Restricting the service to this single failure mode is unacceptable.
  - In some cases, users will demand that some streams succeed while some new requests receive a "busy signal".

Integrated Services Model: Unnecessary? - IV

- **People argue that: "Applications can adapt." However ...**
  - The development of adaptive real-time applications does not eliminate the need to put an upper bound on packet delivery time.
  - Human requirements for interaction and intelligibility limit the possible range of adaptation to network delays.
  - It can be shown in real experiments that, while an application can adapt to network delays of many seconds, the users find that interaction is impossible in these cases.

Integrated Services Model: Reservations - I

- There is an inescapable requirement for routers to be able to reserve resources, in order to provide special QoS for specific user packet streams, or "flows".
  - This in turn requires flow-specific state in the routers, which represents a fundamental change to the Internet model: the Internet architecture has been founded on the concept that all flow-related state should be in the end systems.
  - Designing the TCP/IP protocol suite on this concept of end system flow control led to a robustness that is one of the keys to its success.
  - Adding flow state to the routers threatens Internet robustness!

Integrated Services Model: Reservations - II

- Therefore, the flow state added to the routers for resource reservation in the integrated services model is designed to be "soft", i.e., it is maintained by periodic "refresh".
- Since reservation implies that some users are getting privileged service, resource reservation needs enforcement of policy and administrative controls.
- This in turn calls for authentication of both users and packets.
  - Incidentally, these issues are not unique to "IS": commercialization and commercial security are leading to the same requirements.

Reference Implementation Framework Overview I

- **The implementation framework includes four components:**
  - The packet scheduler,
  - The admission control routine,
  - The classifier,
  - And the reservation setup protocol.
- **The first three items will be discussed in the following foils.**
- **The reservation protocol will be discussed is a separate presentation.**

Reference Implementation Framework - Overview II

- **It is important to understand that the implementation framework discussed below is essentially an example of how the IS model can be realized.**
- **The framework is not a part of any proposed standard!**
- **This notwithstanding, the framework reveals the actual working model of the future Internet....**

Reference Implementation Framework -- flow

- The "flow" is a distinguishable stream of related datagrams that results from a single user activity and requires the same QoS.
  - A flow might consist of one transport connection or one video stream between a given host pair.
  - A flow is the finest granularity of packet stream distinguishable by the IS.
  - A flow is assumed to be simplex, i.e., to have a single source but N destinations. Thus, an N-way teleconference will generally require N flows, one originating at each site.

Reference Implementation Framework -- Router Function

- For integrated services, a router must implement an appropriate QoS for each flow, in accordance with the service model.
  - The **router function** that creates different qualities of service is called "traffic control".
  - **Traffic control** in turn is implemented by three components:
    - The packet scheduler,
    - The classifier,
    - And admission control.

Reference Implementation Framework -- Packet Scheduler I

- **The packet scheduler manages the forwarding of different packet streams using a set of queues and perhaps other mechanisms like timers.**
- **The packet scheduler must be implemented at the point where packets are queued; this is the output driver level of a typical operating system, and corresponds to the link layer protocol.**
- **The details of the scheduling algorithm may be specific to the particular output medium.**
  - For example, the output driver will need to invoke the appropriate link-layer controls when interfacing to a network technology that has an internal bandwidth allocation mechanism.

Reference Implementation Framework -- Packet Scheduler II

- **There is another component that could be considered part of the packet scheduler or as a separate piece: the estimator.**
  - This algorithm is used to measure properties of the outgoing traffic stream
  - To develop statistics that control packet scheduling and admission control.
  - This presentation will consider the estimator to be a part of the packet scheduler.

Reference Implementation Framework -- Classifier I

- **For the purpose of traffic control (and accounting), each incoming packet must be mapped into some class; all packets in the same class get the same treatment from the packet scheduler.**
- **This mapping is performed by the classifier. Choice of a class may be based upon the contents of the existing packet header(s) and/or some additional classification number added to each packet.**

Reference Implementation Framework -- Classifier II

- **A class might correspond to a broad category of flows, e.g., all video flows or all flows attributable to a particular organization.**
  - A class might also hold only a single flow.
- **A class is an abstraction that may be local to a particular router; the same packet may be classified differently by different routers along the path.**
  - For example, backbone routers may choose to map many flows into a few aggregated classes, while routers nearer the periphery, where there is much less aggregation, may use a separate class for each flow.

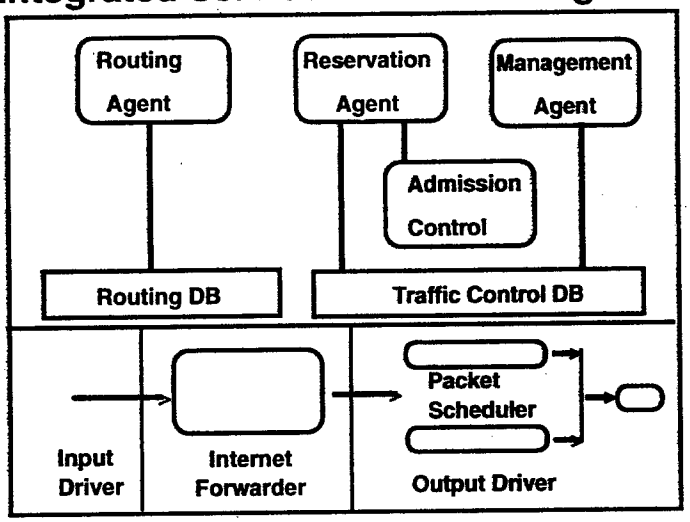
Reference Implementation Framework -- Admission Control I

- **Admission control implements the decision algorithm that a router or host uses to determine whether a new flow can be granted the requested QoS without impacting earlier guarantees.**
- **Admission control is invoked at each node to make a local accept/reject decision, at the time a host requests a real-time service along some path through the Internet. The admission control algorithm must be consistent with the service model, and it is logically part of traffic control.**

Reference Implementation Framework -- Admission Control II

- **Admission control is sometimes confused with policing or enforcement, which is a packet-by-packet function at the "edge" of the network to ensure that a host does not violate its promised traffic characteristics.**
  - Packet level policing is considered to be one of the functions of the packet scheduler.
- **In addition to ensuring that QoS guarantees are met, admission control will be concerned with enforcing administrative policies on resource reservations.**
  - Some policies will demand authentication of those requesting reservations.
- **Finally, admission control will play an important role in accounting and administrative reporting.**

Integrated Services Router -- Diagram



- **The router has two broad functional divisions: the forwarding path below the double horizontal line, and the background code above the line.**

Integrated Services Router -- Forwarding Path

- **The forwarding path of the router is executed for every packet and usually involves a hardware assist.**
- **The path is divided into three sections: input driver, Internet forwarder, and output driver.**
  - The Internet forwarder interprets the internet working protocol header appropriate to the protocol suite.
  - For each packet, an **Internet forwarder** executes a suite-dependent classifier and then passes the packet and its class to the appropriate output driver.
  - The **output driver** implements the packet scheduler.
  - It now has two distinct sections: the packet scheduler that is largely independent of the detailed mechanics of the interface, and the actual I/O driver that is only concerned with the nitty gritty details of the hardware.
  - The **estimator** lives somewhere in between.

Integrated Services Router – Background Routines

- **The background routines create data structures that control the forwarding path.**
  - The **routing agent** implements a particular routing protocol and builds a routing database.
  - The **reservation setup agent** implements the protocol used to set up resource reservations.
    - **If admission control gives the "OK" for a new request, the appropriate changes are made to the classifier and packet scheduler database to implement the desired QoS.**
  - The **network management agent** must be able to modify the classifier and packet scheduler databases to set up controlled link-sharing and to set admission control policies.

IS : Host Model and Routing Changes - I

- **The implementation framework for a host is generally similar to that for a router, with the addition of applications.**
  - **Rather than being forwarded, host data originates and terminates in an application.**
  - **An application needing a real-time QoS for a flow must somehow invoke a local reservation setup agent.**
  - **The best way to interface to applications is not determined yet.**
  - **For example, there might be an explicit API for network resource setup, or the setup might be invoked implicitly as part of the operating system scheduling function.**
  - **The IP output routine of a host may need no classifier, since the class assignment for a packet can be specified in the local I/O control structure corresponding to the flow.**

IS : Host Model and Routing Changes -- II

- **In routers, integrated service will require changes to both the forwarding path and the background functions.**
  - The forwarding path, which may depend upon hardware acceleration for performance, will be the more difficult and costly to change.
  - It will be vital to choose a set of traffic control mechanisms that is general and adaptable to a wide variety of policy requirements and future circumstances, and that can be implemented efficiently.

Integrated Service Model: Core Services - I

- **A service model is embedded within the network service interface invoked by applications to define the set of services they can request.**
  - **For compatibility reasons, this service interface must remain relatively stable**
    - **(or, more properly, extensible; adding new services in the future should be possible but it is expected that it will be hard to change existing services).**
  - **Because of its enduring impact, the service model should not be designed in reference to any specific network artifact but rather should be based on fundamental service requirements.**

Integrated Service Model: Core Services - II

- **The core service model addresses those services which relate most directly to the time-of-delivery of packets.**
  - Services as routing, security, or stream synchronization are subject of other standardization venues.
- **Multicast -- Send one packet to bunch of places**
- **RTP -- real Time Protocol**
- **RSVP -- reservation Protocol**
- **will be discussed in later foilsets**

Integrated Service Model: Core Services - III

- **A service model consists of a set of service commitments: in response to a service request the network commits to deliver some service.**
  - **These service commitments can be categorized by the entity to whom they are made: they can be made to either individual flows or to collective entities (classes of flows).**
  - **The service commitments made to individual flows are intended to provide reasonable application performance, and thus are driven by the ergonomic requirements of the applications; these service commitments relate to the quality of service delivered to an individual flow.**

Integrated Service Model: Core Services - IV

- **The service commitments made to collective entities are driven by resource-sharing, or economic, requirements; these service commitments relate to the aggregate resources made available to the various entities.**
  - In the following, we will explore the service requirements of individual flows and describe a corresponding set of services.
  - We then discuss the service requirements and services for resource sharing.
  - Finally, we conclude with some remarks about packet dropping.

Integrated Service Model: Core Services- QoS - I

- **For QoS requirements, the core service model is concerned almost exclusively with the time-of-delivery of packets.**
- **Per-packet delay is the central quantity about which the network makes quality of service commitments.**
- **Strictly speaking, the only quantity about which a quantitative service commitments can be made are bounds on the maximum and minimum delays.**

Integrated Service Model: Core Services- QoS - II

- **The degree to which application performance depends on low delay service varies widely, and one can make several qualitative distinctions between applications based on the degree of their dependence.**
  - One class of applications needs the data in each packet by a certain time and, if the data has not arrived by then, the data is essentially worthless - these are real-time applications.
  - Another class of applications will always wait for data to arrive; these are "elastic" applications.
- **The taxonomy of applications into real time and elastic is neither exact nor complete. It is only used to guide the development of the core service model.**

Integrated Service Model: Core Services Playback Real-Time Apps - I

- **An important class of such real-time applications: "playback" applications.**
  - **In a playback application, the source takes some signal, packetizes it, and then transmits the packets over the network.**
  - **The network inevitably introduces some variation in the delay of the delivered packets.**
  - **The receiver depacketizes the data and then attempts to faithfully play back the signal. This is done by buffering the incoming data and then replaying the signal at some fixed offset delay from the original departure time;**

Integrated Service Model: Core ServicesPlayback Real-Time Apps - II

- The term "playback point" refers to the point in time which is offset from the original departure time by this fixed delay.
- Data arriving after the playback point is essentially useless.
- In order to choose a reasonable value for the offset delay, an application needs some "a priori" characterization of the maximum delay its packets will experience.
- This "a priori" characterization could either be provided by the network in a quantitative service commitment to a delay bound, or through the observation of the delays experienced by the previously arrived packets.

Integrated Service Model: Core ServicesPlayback Real-Time Apps - III

- The performance of a playback application is measured along two dimensions: latency and fidelity.
  - Some playback applications, in particular those that involve interaction between the two ends of a connection such as a phone call, are rather sensitive to the latency;
  - Transmitting a movie or lecture is not.
  - Similarly, applications exhibit a wide range of sensitivity to loss of fidelity.
    - Intolerant applications require an absolutely faithful playback,
    - Tolerant applications can tolerate some loss of fidelity.

Integrated Service Model: Core Services Playback Real-Time Apps - IV

- Vast majority of audio and video applications will be tolerant.
- One can however envision applications, such as circuit emulation, that are intolerant.
- Web based Computing is likely to be intolerant if synchronizing many nodes on a single application as delays at one point, will delay all nodes
- It is important to note that late packets always decrease fidelity, even if the applications has clever "adaptive" ways of dealing with them

Integrated Service Model: Core Services Playback Real-Time Apps - V -- intolerant

- Hence, intolerant applications must use a fixed offset delay.
  - For a given distribution of packet delays, this fixed offset delay must be larger than the absolute maximum delay, to avoid the possibility of late packets.
  - Such an application can only set its offset delay appropriately if it is given a perfectly reliable upper bound on the maximum delay of each packet.
- A service characterized by a perfectly reliable upper bound on delay is called "guaranteed service".
- This is the appropriate service model for intolerant playback applications.

Integrated Service Model: Core Services Real-Time Apps - Tolerant - I

- **In contrast, tolerant applications do not need to set their offset delay greater than the absolute maximum delay.**
  - They can also attempt to reduce their latency by varying their offset delays in response to the actual packet delays experienced in the recent past ("adaptivity").
- **For tolerant applications, the "right: service model is "predictive service" which supplies a fairly reliable, but not perfectly reliable, delay bound.**

Integrated Service Model: Core Services Real-Time Apps - Tolerant/Predictive Service - II

- **For "predictive service" the bound is not based on worst case assumptions on the behavior of other flows.**
- **Instead, this bound might be computed with properly conservative predictions about the behavior of other flows.**
- **If the network turns out to be wrong and the bound is violated, the application's performance will perhaps suffer, but the users are willing to tolerate such interruptions in service in return for the presumed lower cost of the service.**

Integrated Service Model: Core Services Real-Time Apps - Guaranteed v. Predictive Services

- **Why two different services?**
  - The key consideration here is efficiency:
  - When one relaxes the service requirements from **perfectly to fairly reliable bounds**, this increases the level of network utilization that can be sustained, and thus the price of the predictive service will presumably be lower than that of guaranteed service.
  - The predictive service class is motivated by the conjecture that the performance penalty will be small for tolerant applications but the overall efficiency gain will be quite large.

Integrated Service Model: Core Services - Adaptive Real-Time Apps - I

- **Let us discuss accommodation of the "true" adaptive applications in the service model**
  - **A fundamental point of the overall IS architecture is that traffic characterization and admission control are necessary for these real-time delay bound services.**
  - **So far one assumed that an application's data generation process is an intrinsic property unaffected by the network.**
  - **However, there are likely to be many audio and video applications which can adjust their coding scheme and thus can alter the resulting data generation process depending on the network service available.**

Integrated Service Model: Core Services - Adaptive Real-Time Apps - II

- This alteration of the coding scheme will present a trade-off between fidelity (of the coding scheme itself, not of the playback process) and the bandwidth requirements of the flow.
- Such "rate-adaptive" playback applications have the advantage that they can adjust to the current network conditions not just by resetting their playback point but also by adjusting the traffic pattern itself.
- For rate-adaptive applications, the traffic characterizations used in the service commitment are not immutable.
- One can thus augment the service model by allowing the network to notify (either implicitly through packet drops or explicitly through control packets) rate-adaptive applications to change their traffic characterization.

Integrated Service Model: Core Services - Elastic Applications - I

- Elastic applications will always wait for data to arrive.
  - Elastic application typically uses the arriving data immediately and will always choose to wait for the incoming data rather than proceed without it. These applications do not require any a priori characterization of the service in order for the application to function.
  - For a given distribution of packet delays, the perceived performance of elastic applications will depend more on the average delay than on the tail of the delay distribution.
- Categories of elastic applications:
- interactive burst (Telnet, X, NFS),
- interactive bulk transfer (FTP), and
- asynchronous bulk transfer (electronic mail, FAX).

Integrated Service Model: Core Services - Elastic Applications - II

- The delay requirements of elastic applications vary from rather demanding for interactive burst applications to rather lax for asynchronous bulk transfer, with interactive bulk transfer being intermediate between them.
- An appropriate service model for elastic applications is to provide "as-soon-as-possible", or ASAP (=best effort) service.
- One should offer several classes of best-effort service to reflect the relative delay sensitivities of different elastic applications.
- This service model allows interactive burst applications to have lower delays than interactive bulk applications, which in turn would have lower delays than asynchronous bulk applications.
- Applications using this service are not subject to admission control.

Integrated Service Model: Core Services -- Analysis of Taxonomy - I

- Although the applications taxonomy was rather crude, the core service model should be judged on its ability to adequately meet the needs of the entire spectrum of applications.
  - Not all real-time applications are playback applications;
  - for example, one might imagine a visualization application which merely displayed the image encoded in each packet whenever it arrived.
  - However, non- playback applications can still use either the guaranteed or predictive real-time service model, although these services are not specifically tailored to their needs.

Integrated Service Model: Core Services -- Analysis of Taxonomy - II

- Similarly, playback applications cannot be neatly classified as either tolerant or intolerant, but rather fall along a continuum.
- Offering both guaranteed and predictive service allows applications to make their own trade-off between fidelity, latency, and cost.
- Despite these obvious deficiencies in the taxonomy, there are reasons to believe that it describes the service requirements of current and future applications well enough so that the Integrated Service model core services can adequately meet essentially all application needs.

Integrated Service Model: Core Services -- Resource Sharing - I

- The quantity of primary interest in resource-sharing is aggregate bandwidth on individual links.
- "link- sharing" component of the service model addresses the question of how to share the aggregate bandwidth of a link among various collective entities according to some set of specified shares.
- Multi-entity link-sharing
  - A link may be purchased and used jointly by several organizations, government agencies or the like.
  - They may wish to insure that under overload the link is shared in a controlled way, perhaps in proportion to the capital investment of each entity.
  - At the same time, they might wish that when the link is underloaded, any one of the entities could utilize all the idle bandwidth.

Integrated Service Model: Core Services -- Resource Sharing - II

- **Multi-protocol link-sharing**
  - In a multi-protocol Internet, it may be desired to prevent one protocol family (DECnet, IP, IPX, OSI, SNA, etc.) from overloading the link and excluding the other families.
  - Note that different families may have different methods of detecting and responding to congestion, and some methods may be more "aggressive" than others.
  - This could lead to a situation in which one protocol backs off more rapidly than another under congestion, and ends up getting no bandwidth.

Integrated Service Model: Core Services -- Resource Sharing - III

- **Multi-service sharing**
  - Within a protocol family such as IP, an administrator might wish to limit the fraction of bandwidth allocated to various service classes.
  - For example, an administrator might wish to limit the amount of real-time traffic to some fraction of the link, to avoid preempting elastic traffic such as FTP.
- **In general terms**
  - The link-sharing service model is to share the aggregate bandwidth according to some specified shares.
  - A hierarchy of shares is possible.

Integrated Service Model: Core Services -- Resource Sharing - IV

- Definition of a service supporting efficient link-sharing is difficult.
- There is a number of research models, including the idealized fluid model of instantaneous link-sharing with proportional sharing of excess where at every instant the available bandwidth is shared between the active entities in proportion to the assigned shares of the resource.
- This fluid model exhibits the desired policy behavior but is an unrealistic idealization.
- The actual service model should be to approximate, as closely as possible, the bandwidth shares produced by the ideal fluid model.
- Actual implementation may be difficult!

Integrated Services: Outstanding Issues

- There is a number of issues that have not been discussed but are quite important:
  - impact of packet dropping on the proposed core services
  - Usage feedback (to prevent abuse of network resources)
  - Reservation model (will be covered)
  - Detailed traffic control mechanisms for
    - packet scheduling, controlled packet dropping, packet classification, admission control
  - Combination of the traffic control mechanisms to ensure
    - guaranteed delay bounds, link sharing, predictive real time service
  - Implementation of the soft state for the routers
- For these, access to the original literature is recommended

Multicast

Rationale, Technology, Perspectives

Marek Podgorny
NPAC
Syracuse University
111 College Place
Syracuse
New York 13244-4100

Abstract of Multicast - Rationale, Technology, Perspectives

- This module covers basics of the multicast technology
- It also introduces switching technology for traditional broadcast networks
- For multicast, the focus is on the multicast routing protocols
- The MBONE is not discussed but will be included later

Multimedia and Multicast

- Multimedia streams tend to require higher bandwidth compared to text-based applications
- Some MM applications require one-to-many connectivity
  - Desktop conferencing (many-to-many), LAN TV, collaborative computing, corporate broadcast)
- Unicast method does not scale even on LANs
- Broadcast does not scale on complex LANs and on WANs
- Multipoint connectivity also benefits traditional applications (example: e-mail, news distribution, electronic journals)

Multipoint Transmission Methods

- Unicast:
  - Application sends a copy of each packet to each member of the multicast group. Simple to implement, but does not scale for large groups. It also requires extra bandwidth as the same information has to be carried multiple times even on shared links.
- Broadcast:
  - Application sends a copy of each packet to a broadcast address. Even simpler than unicast to implement. However, the network must either stop broadcasts at the LAN boundary or send the broadcast everywhere. Sending the broadcast everywhere is a waste of network resources if only a small group actually needs to see the packets.

Multipoint Transmission Methods (Continued)

- Multicast:
  - Application sends one copy of each packet and addresses it to the group of computers that want to receive it. Multicast addresses packets to a group of receivers rather than to a single receiver, and it depends on the network to forward the packets to only these networks that registered as receivers.

Multicast support on LANs

- Most popular LAN technologies support multicast at their data link layer: Ethernet, FDDI, Token Ring. They however use different technologies to achieve the same functionality
  - An individual computer can listen to a unicast address, several multicast addresses, and the broadcast address on Ethernet and FDDI segments.
  - Token Rings have functional addresses that can be used to address groups of receivers.
- Many multipoint applications are valuable precisely because they are not limited to a single LAN.
- On internets using mixed data link technologies and other networking technologies, multicast must be implemented at the network layer.

Multicast Technology Components

- Addressing:
  - There must be a network-layer address that is used to communicate with a group of receivers rather than a single receiver. There must be also a mechanism for mapping this address onto data-link layer multicast addresses where they exist.
- Dynamic registration:
  - There must be a mechanism for the computer to communicate to the network that it is a member of a particular group. Computers must register to let the global network know which local networks need to receive traffic for each group.

Multicast Technology Components (Continued)

- Multicast routing:
  - The network must be able to build packet distribution trees that allow sources to send packets to all receivers. Packet distribution trees need to ensure that each packet exists only one time on any given network - if there are multiple receivers on a given branch, there should only be one copy of the packets on that branch.

IP Multicast Design

- **Addressing:**
  - **The IP address space is divided into four pieces: Class A, Class B, Class C, and Class D. Class D is reserved for multicast traffic. Class D addresses are allocated dynamically.**
- **Dynamic Registration:**
  - **RFC 1112 defines the Internet Group Membership Protocol (IGMP). IGMP specifies how the host should inform the network that it is a member of a particular multicast group.**

IP Multicast Design (Continued)

- **Multicast Routing:**
  - **RFC 1075 defines the Distance Vector Multicast Routing Protocol (DVMRP).**
  - **RFC 1584 defines the Multicast Open Shortest Path First (MOSPF) protocol -- OSFP extension to OSPF supporting IP Multicast.**
  - **PIM is a multicast protocol that can be used in conjunction with all unicast IP routing protocols. Relevant IETF drafts:**
    - Protocol-Independent Multicast (PIM): Motivation and Architecture
    - Protocol-Independent Multicast (PIM): Protocol Specification.

IP Multicast Routing Protocols: DVMRP

- DVMRP uses Reverse Path Forwarding. When a router receives a packet, it floods the packet out of all paths except the one that leads back to the packet's source. Data stream reaches all LANs (possibly multiple times). If a router is attached to a set of LANs that do not want to receive a particular multicast group, the router can send a "prune" message back up the distribution tree to stop subsequent packets from traveling where there are no members.
- Since new hosts may want to join the multicast group at any time, DVMRP must periodically re-flood. This creates a scaling problem, especially if pruning not effective or not implemented.

IP Multicast Routing Protocols: DVMRP (Continued)

- **DVMRP implements its own unicast routing protocol (similar to RIP) to determine which interface leads back to the source of the data stream. The path that the multicast traffic follows may not be the same as the path that the unicast traffic follows.**
- **DVMRP has been used to build the MBONE by building tunnels between DVMRP-capable machines.**
- **DVMRP is state of the art today.**

IP Multicast Routing Protocols: MOSPF

- Multicast OSPF (MOSPF) was defined as an extension to the OSPF unicast routing protocol. OSPF works by having each router in a network understand all of the available links in the network. Each OSPF router calculates routes from itself to all possible destinations.
- MOSPF works by including multicast information in OSPF link state advertisements. An MOSPF router learns which multicast groups are active on which LANs.
- **MOSPF builds a distribution tree for each source/group pair and computes a tree for active sources sending to the group. The tree state is cached, and trees must be recomputed when a link state change occurs or when the cache times out.**

IP Multicast Routing Protocols: MOSPF (Continued)

- **MOSPF works only in internetworks that are using OSPF.**
- **MOSPF is best suited for environments that have relatively few source/group pairs active at any given time. It will work less well in environments that have many active sources or environments that have unstable links.**
- **OSPF is not widely used ....**

IP Multicast Routing: PIM

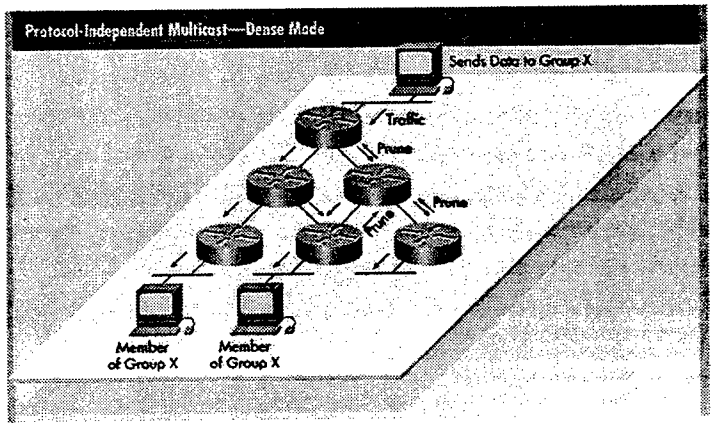
- PIM (Protocol-Independent Multicast):
  - Works with all existing unicast routing protocols.
  - Supports two different types of multipoint traffic distribution patterns:
    - Dense-mode PIM: uses Reverse Path Forwarding and looks a lot like DVMRP. However, dense-mode PIM is that PIM works with whatever unicast protocol is being used

IP Multicast Routing: PIM (Continued)

- Sparse-mode PIM: optimized for environments where there are many multipoint data streams with each data stream goes to a relatively small number of the LANs in the internetwork.
  - Reverse Path Forwarding wastes bandwidth for such traffic pattern.
  - Sparse-mode PIM works by defining a Rendezvous Point. Senders send data to and the receivers receive data from the Rendezvous Point. .
  - For each data stream the routers in the path will optimize the path automatically to remove any unnecessary hops.
  - Sparse-mode PIM assumes that no hosts want the multicast traffic unless they specifically ask for it.
- PIM can concurrently support both modes

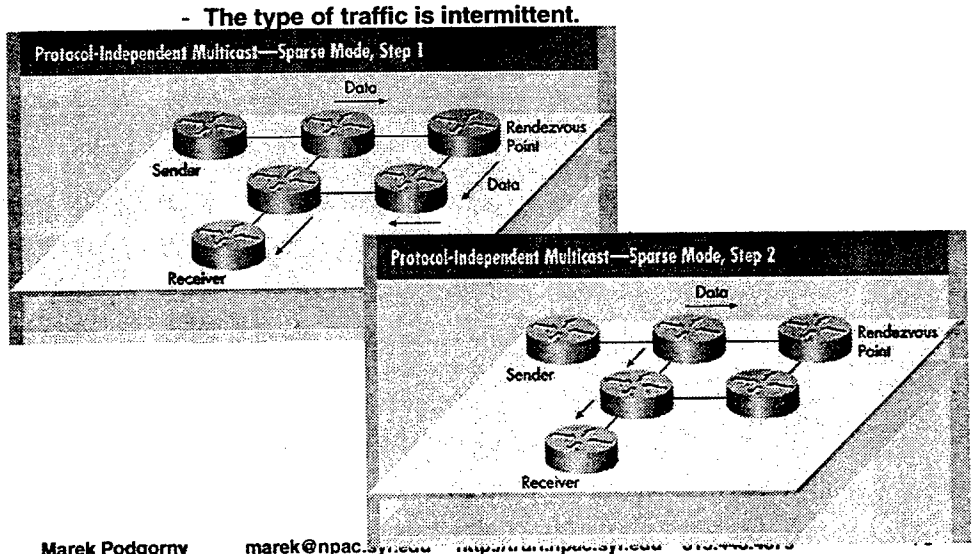
IP Multicast Routing: PIM (2)

- **Dense Mode PIM routing is useful when:**
  - Senders and receivers are in close proximity to one another.
  - There are few senders and many receivers.
  - The volume of multicast traffic is high.
  - The stream of multicast traffic is cons



IP Multicast Routing: PIM (3)

- **Sparse Mode PIM routing is useful when:**
  - There are few receivers in a group.
  - Senders and receivers are separated by WAN links.
  - The type of traffic is intermittent.

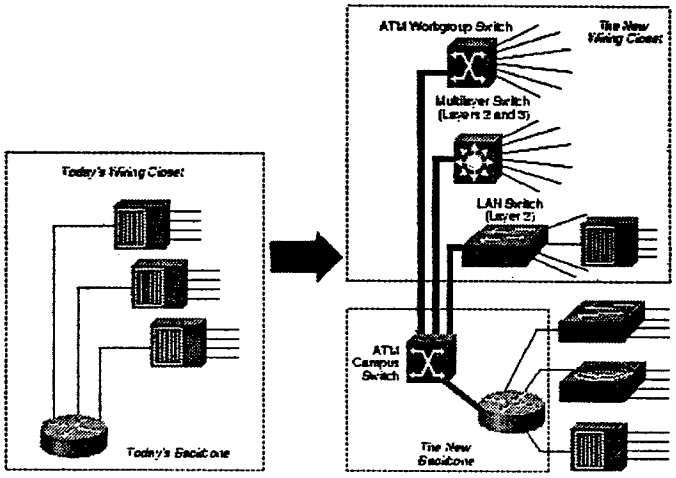


IP Multicast Services in the Workgroup

- LAN switches are a natural bandwidth enhancement tool able to improve the performance of time-critical or bandwidth-intensive multimedia applications.
- In order for multicast applications to work in the LAN, both switching and routing systems must support IP multicast capabilities.
- As bandwidth enhancement and multicast are both critical, LAN switches must have sufficient internetworking intelligence to forward multicast traffic only to those workgroup segments that will use this traffic. Otherwise, multicast traffic will be indiscriminately broadcast to all workgroup segments, needlessly robbing network bandwidth from other users' applications.

Switched Networks

- Transition from routed to switched networks



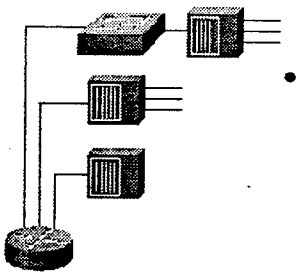
Switched Networks

- Why the transition?
  - Switched internetworks integrate switching devices into existing shared-media networks to optimize the benefits of both routing and switching.
  - Switched networks provide users with new services and capabilities, such as virtual LANs (VLANs), multimedia support, and more efficient tools for network management.
  - Higher bandwidth to the desktop (microsegmentation) is one of the main driving forces, directly related to multimedia applications.

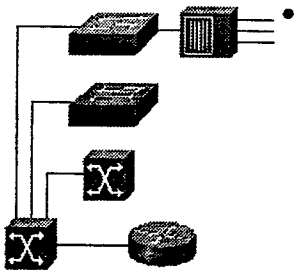
Switched Networks (Continued)

- Switches (unlike hubs and routers) can (a) forward traffic quickly and directly to its destination; (b) provide non-blocking services and concurrent connections
- In addition, switches allow to de-couple the physical and logical network layouts: they enable virtual LANs
- Switches, in general, have better scalability than bridges and routers.

Switched Networks - Evolution (1)

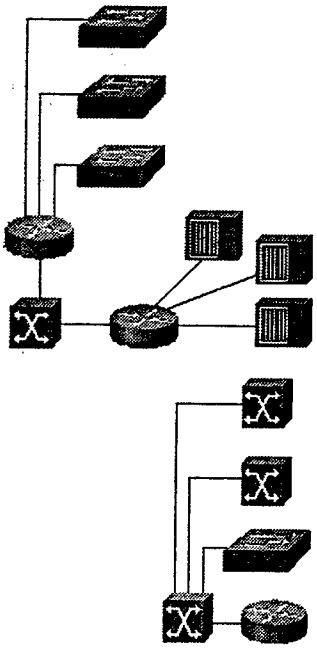


- Phase one: the microsegmentation phase. Companies retain hubs and routers but insert a LAN switch to enhance performance.



- Phase two: ATM technology and routing between switches. LAN switches perform switch processing and provide dedicated bandwidth to the desktop and to shared-media hubs. Backbone routers are clustered by ATM switches to increase backbone bandwidth, matching the increased bandwidth in the wiring closet.

Switched Networks - Evolution (2)



- Phase three connects the ATM core switch directly to LAN switches in the wiring closet and to centralized or distributed ATM routers. The network backbone is now ATM-centric, with all other devices at the periphery. Multilayer switches have the intelligence to forward packets between the different VLANs locally, or layer 2 switches, or a combination of both.
- Phase four is the end-to-end switching with integral VLAN and multilayer switching capability. Route and Switch Processors are distributed over the ATM fabric.

Switched Networks

- What are the challenges?
  - Interoperability: legacy (hub/router) networks have to coexist with switched networks
  - Switching capabilities must be implemented at both layer 2 and layer 3 of the OSI model (multilayer switching)
  - LAN emulation must become commonplace - it isn't now
  - Or, network layer protocols must be modified to operate directly over ATM in native (AAL) mode (RFC 1577). This, BTW, would make ATM native QoS guarantees available for higher level protocols

Switch Networks (Continued)

- At present, neither LANE nor RFC 1577 exploit ATM QoS capabilities
- RSVP must be implemented on both routers and switches
- Multicast routing protocols must be extended to switching platforms
- New network management methodologies and tools are required for switched networks to exploit their flexibility

RSVP: Reservation Protocol

Supporting Quality of Service over Packet Networks

Marek Podgorny
NPAC
Syracuse University
111 College Place
Syracuse
New York 13244-4100
Based on Internet Draft: RSVP-SPEC-10

Abstract

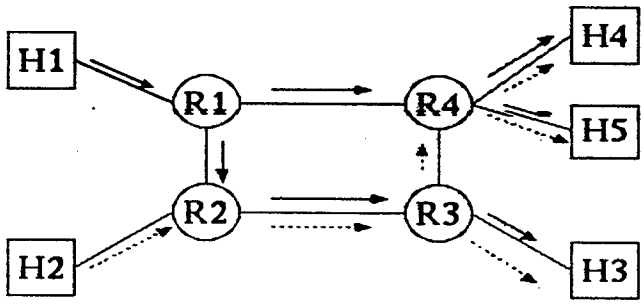
- This module discusses details of the Reservation Protocol (RSVP)
- RSVP is a part of the Integrated Services Model for Internet
- RSVP is a tool to establish Quality of Service over traditional packet networks

What is RSVP

- **RSVP is a network control protocol that will allow Internet applications to obtain special qualities-of-service for their data flows.**
  - This will generally require reserving resources along the data path(s).
  - RSVP is a component of the future "integrated services" Internet, which will provide both best-effort and real-time qualities of service
  - When an application in a host requests a specific QoS for its data stream, RSVP is used to deliver the request to each router along the path(s) of the data stream and to maintain router and host state to provide the requested service.
  - Although RSVP was developed for setting up resource reservations, it is easily adaptable to transport other kinds of network control information along data flow paths.

RSVP Basic Functionality

- **RSVP supports multicast or unicast simplex data delivery. RSVP is fundamentally designed for multicasting as well as unicasting, and it treats data flow as one directional. It distinguishes the roles of data sender (e.g., hosts H1, H2) from data receiver (hosts H3, H4, H5), although in many cases the same application will play both roles.**

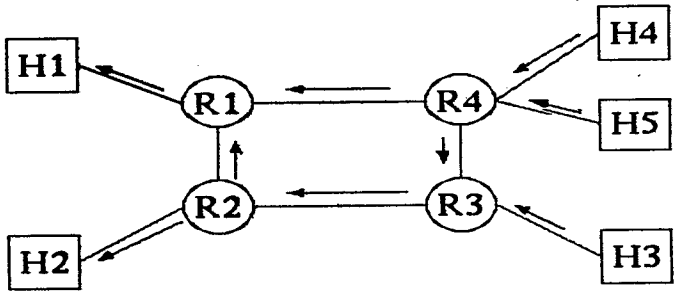


RSVP: A Receiver-Oriented Protocol

- **RSVP is receiver-oriented.**
  - To efficiently handle heterogeneous receivers and dynamic group membership, RSVP makes receivers responsible for requesting resource reservations. Each receiver can request a QoS that is tailored to its particular requirement, by sending RSVP reservation messages upstream towards the senders.

RSVP: A Receiver-Oriented Protocol (Continued)

- **RSVP reservation messagesflowing upstream. Just as thedata branches out in routers R1, R3, and R4, so the reservation messages going upstream are "merged". A single reservation message need only flow upstream until it is merged with another reservation.**

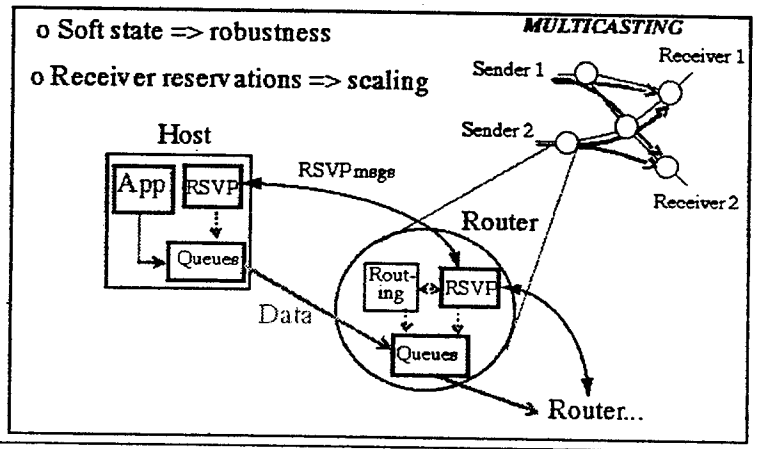


RSVP Basic Functionality

- **RSVP handles heterogeneous receivers.**
  - Different hosts on the same multicast delivery tree may have different capabilities and therefore need different QoS.
- **RSVP adapts to changing group membership as well as changing routes.**
  - For dynamic adaptability and robustness, RSVP maintains "soft state" in the routers. The only permanent state is in the end systems, which periodically send their RSVP control messages to refresh the router state. In the absence of refresh, RSVP state in routers will time out and be deleted.
- **RSVP is not a routing protocol.**
  - The RSVP daemon consults the local routing protocol(s) to obtain routes. RSVP is designed to operate with existing and future unicast and multicast routing protocols. A host sends IGMP messages to join a multicast group, but it uses RSVP messages to reserve resources along the delivery path(s) from that group.

RSVP Operational Principles

- **A QoS request from an application is passed to the local RSVP implementation (user daemon). RSVP passes the request to all the nodes along the reverse data path to the destination.**
  - RSVP provides transparent operation through routers that do not support it.



RSVP Operational Principles (Continued)

- **At each node, RSVP applies a local decision procedure (admission control) to the QoS request.**
  - If admission control succeeds, it sets the parameters to the Classifier and Packet Scheduler to obtain the desired QoS. If admission control fails at any node, RSVP returns an error indication to the application.
- **Each router in the path capable of resource reservation will pass incoming data packets to a Packet Classifier and then queue them in a Packet Scheduler.**
  - The Packet Classifier determines the route and the QoS class for each packet. The Scheduler allocates a particular outgoing link for packet transmission.

RSVP Operational Principles (Continued)

- **For QoS-active link-layer media the packet scheduler is responsible for negotiation with the link layer to obtain the QoS requested by RSVP.**
  - Mapping to the link layer QoS may be accomplished in a number of possible ways; the details will be medium-dependent. On a QoS-passive medium such as a leased line, the scheduler itself allocates packet transmission capacity. The scheduler may also allocate other system resources such as CPU time or buffers.

RSVP Operational Principles (Continued)

- **RSVP is designed to scale well for very large multicast groups.**
  - **Since both the membership of a large group and the topology of large multicast trees may change with time, the RSVP design assumes that router state for traffic control will be built and destroyed incrementally. Hence, RSVP uses "soft state" in the routers, i.e., RSVP sends periodic refresh messages to maintain the state along the reserved path; in absence of refreshes, the state will automatically time out and be deleted.**

RSVP Operational Principles (Continued)

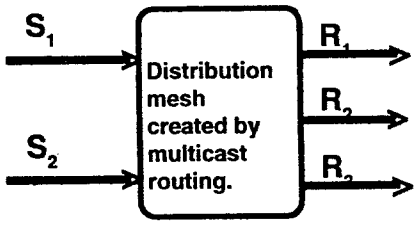
- **RSVP reserves resources for simplex data streams, i.e., it reserves resources in only one direction on a link**
  - A sender is logically distinct from a receiver. However, the same application may act as both sender and receiver.
- **RSVP mechanisms provide a general facility for creating and maintaining distributed reservation state across a mesh of multicast delivery paths.**
  - RSVP transfers reservation parameters as opaque data (except for certain well-defined operations on the data), which it simply passes to admission control and to the Packet Scheduler and Classifier for interpretation.

RSVP Data Flows

- **RSVP defines a "session" as a data flow with a particular destination and transport-layer protocol.**
  - The destination of a session is generally defined by DestAddress, the IP destination address of the data packets, and possibly by DstPort, a "generalized destination port". RSVP treats each session independently.
  - DestAddress is a group address for multicast delivery or the unicast address of a single receiver. DstPort could be defined by a UDP/TCP destination port field, by an equivalent field in another transport protocol, or by some application-specific information.

RSVP Data Flows (Continued)

- **For unicast transmission, there will be a single destination host but there may be multiple senders; RSVP can set up reservations for multipoint to single point transmission.**
- **Multicast distribution forwards a copyof each data packet from a sender S_i to every receiver R_j; a unicast distribution session has a single receiver R. Each sender S_i may be running in a unique Internet host, or a single host may containmultiple senders, distinguished by generalized source ports.**

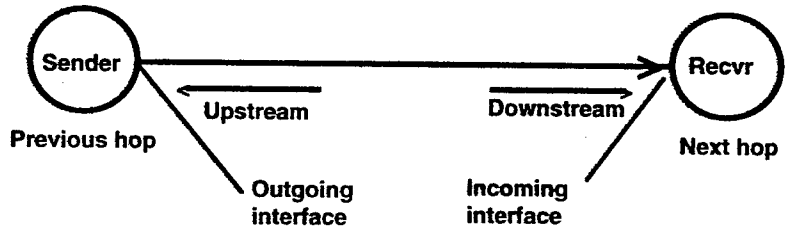


RSVP Reservation Model

- An elementary RSVP reservation request consists of a "flowspec" and a "filter spec"; this pair is called a "flow descriptor".
  - The flowspec specifies a desired QoS. It is used to set parameters to the node's packet scheduler, assuming that admission control succeeds.
  - The filter spec, together with a session specification, defines the set of data packets – the "flow" – to receive the QoS defined by the flowspec. Filter spec is used to set parameters in the packet classifier.

RSVP Reservation Model (Continued)

- Data packets that are addressed to a particular session but do not match any of the filter specs for that session are handled as best-effort traffic.
- Please, note "upstream" and "downstream" convention!



RSVP Reservation Model (Continued)

- **The flowspec in a reservation request will generally include a service class and two sets of numeric parameters:**
  - **An "Rspec" (R for `reserve') that defines the desired QoS,**
  - **A "Tspec" (T for `traffic') that describes the data flow.**
  - **The formats and contents of Tspecs and Rspecs are determined by the integrated service model, and are generally opaque to RSVP.**

RSVP Reservation Model (Continued)

- **Filter specs may select arbitrary subsets of the packets in a given session.**
  - Subsets might be defined in terms of
    - **Senders (i.e., sender IP address and generalized source port),**
    - **A higher-level protocol**
    - **Any fields in any protocol headers in the packet.**
      - Example: filter specs might be used to select different subflows in a hierarchically-encoded signal by selecting on fields in an application-layer header.
      - Current RSVP software does not yet support this option.
  - Because the UDP/TCP port numbers are used for packet classification, each router must be able to examine these fields.

RSVP Reservation Model (Continued)

- **RSVP reservation request messages originate at receivers and are passed upstream towards the sender. At each intermediate node, two general actions are taken:**
  - Make a reservation
    - **The request is passed to admission control and policy control.**
      - If either test fails, the reservation is rejected and RSVP returns an error message to the appropriate receiver.
      - If both succeed, the node uses the flowspec to set up the packet scheduler for the desired QoS and the filter spec to set the packet classifier to select the appropriate data packets.

RSVP Reservation Model (Continued)

- **Forward the request upstream**
  - The reservation request is propagated upstream towards the appropriate senders. The set of sender hosts to which a given reservation request is propagated is called the "scope" of that request.
- **Forwarded reservation request may differ from the request that it received from downstream:**
  - Reservations for the same sender from different downstream branches of the tree are "merged" as reservations travel upstream; a node forwards upstream only the reservation request with the "maximum" flowspec.
  - Reservation might be purposefully modified by traffic control.

RSVP Reservation Model (Continued)

- **The receiver sending a reservation request can request a confirmation message.**
  - **A successful reservation request propagates upstream along the multicast tree until it reaches a point where an existing reservation is equal or greater than that being requested.**
  - **At that point, the arriving request is merged with the reservation in place and the node may then send a reservation confirmation message back to the receiver.**
  - **Note that the receipt of a confirmation is only a high-probability indication, not a guarantee, that the requested service is in place. This uncertainity results from possible security policies.**

RSVP Reservation Model (Continued)

- **The basic RSVP reservation model is "one pass":**
  - This scheme provides no easy way for a receiver to find out the resulting end-to-end service.
  - RSVP supports an enhancement to one-pass service known as "One Pass With Advertising" (OPWA)
  - With OPWA, RSVP control packets are sent downstream to gather information that may be used to predict the end-to-end QoS. The results are delivered by RSVP to the receiver hosts. The results may then be used by the receiver to dynamically adjust the reservation request.

RSVP Reservation Styles (Continued)

- **A reservation request includes a set of options which are collectively called the reservation "style".**
  - Distinct or shared: the treatment of reservations for different senders within the same session: establish a "distinct" reservation for each upstream sender, or else make a single reservation that is "shared" among all packets of selected senders.

RSVP Reservation Styles (Continued)

- **Senders' selection: an "explicit" list of all selected senders, or a "wildcard" that implicitly selects all the senders to the session.**
  - Explicit sender-selection reservation: each filter spec must match exactly one sender
  - Wildcard sender-selection: no filter spec is needed.

Sender Selection	Reservations	
	Distinct	**Shared**
Explicit	*Fixed-Filer (FF Style)*	*Shared-Explicit (SE) Style*
Wildcard	*X*	*Wildcard-Filder (WF) Style*

RSVP Reservation Styles (Continued)

- **Wildcard-Filter (WF) Style**
  - The WF style implies the options: "shared" reservation and "wildcard" sender selection.
  - WF-style reservation creates a single reservation into which flows from all upstream senders are mixed. This reservation may be thought of as a shared "pipe", whose "size" is the largest of the resource requests from all receivers, independent of the number of senders using it.

RSVP Reservation Styles (Continued)

- **A WF-style reservation is propagated upstream towards all sender hosts, and automatically extends to new senders as they appear.**
- **Symbolically, a WF-style reservation request is represented by:**
- **WF(* { Q})**
- **Where the asterisk represents wildcard sender selection and Q represents the flowspec.**

RSVP Reservation Styles (Continued)

- **Fixed-Filter (FF) Style**
  - The FF style implies the options: "distinct" reservations an "explicit" sender selection.
    - **An elementary FF-style reservation request creates a distinct reservation for data packets from a particular sender, not sharing them with other senders' packets for the same session.**
    - **The total reservation on a link for a given session is the total of the FF reservations for all requested senders. On the other hand, FF reservations requested by different receivers R_j but selecting the same sender S_i must be merged to share a single reservation.**

RSVP Reservation Styles (Continued)

- **Symbolically, an elementary FF reservation request is represented by:**
- **FF(S{ Q})**
- **Where S is the selected sender and Q is the corresponding flowspec; the pair forms a flow descriptor.**
  - RSVP allows multiple elementary FF-style reservations to be requested a the same time, using a list of flow descriptors: FF(S1{ Q1} , S2{ Q2} , ...)

RSVP Reservation Styles (Continued)

- **Shared Explicit (SE) Style**
  - The SE style implies the options: "shared" reservation and "explicit" sender selection.
    - **An SE-style reservation creates a single reservation into which flows from all upstream senders are mixed. However, like the FF style, the SE style allows a receiver to explicitly specify the set of senders.**
  - We can represent an SE reservation request containing a flowspec Q and a list of senders S1, S2, ... by:
    - ***SE((S1,S2,...){ Q})***

RSVP Reservation Styles

- **Both WF and SE styles create shared reservations, appropriate for those multicast applications whose properties make it unlikely that multiple data sources will transmit simultaneously.**
  - Packetized audio is an example of an application suitable for shared reservations; since a limited number of people talk at once, each receiver might issue a WF or SE reservation request for twice the bandwidth required for one sender (to allow some over-speaking).
  - The FF style, which creates independent reservations for the flows from different senders, is appropriate for video signals.

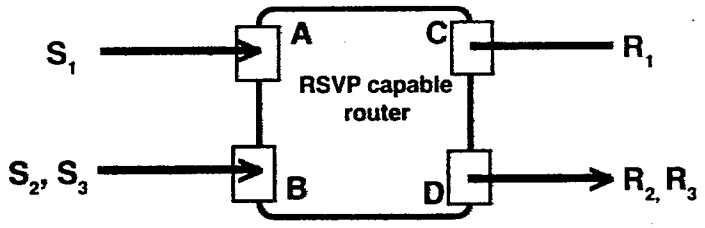
RSVP Reservation Styles (Continued)

- **The RSVP rules disallow:**
  - **Merging of shared reservations with distinct reservations, since these modes are fundamentally incompatible.**
  - **Merging explicit sender selection with wildcard sender selection, since this might produce an unexpected service for a receiver that specified explicit selection.**
  - **Therefore, WF, SE, and FF styles are all mutually incompatible.**

RSVP Reservation Styles (Continued)

- **Is it possible to simulate the effect of a WF reservation using the SE style?**
  - When an application asked for WF, the RSVP daemon on the receiver host could use local path state to create an equivalent SE reservation that explicitly listed all senders.
  - However, an SE reservation forces the packet classifier in each node to explicitly select each sender in the list, while a WF allows the packet classifier to simply "wild card" the sender address and port. When there is a large list of senders, a WF style reservation can therefore result in considerably less overhead than an equivalent SE style reservation. For this reason, both SE and WF are included in the protocol.

Examples of Reservation Styles



- Two incoming interfaces (A and B) through which data streams will arrive
- Two outgoing interfaces (C and D) through which data will be forwarded
- Three upstream senders (S1 - S3). Packets from sender S1 (S2 and S3) arrive through previous hop A (B, respectively).

Examples of Reservation Styles (Continued)

- Three downstream receivers. Packets bound for R1 (R2 and R3) are routed via outgoing interface C (D, respectively).
- It is furthermore assumed that R2 and R3 arrive via different next hops.This illustrates the effect of a non-RSVP cloud or a broadcast LAN on interface D.
- Multicast setup: data packets from each Si are routed to both outgoing interfaces

Examples of Reservation Styles

- **Example of the WF style**
  - Two possible merging situations.
    - **Each of the two next hops on interface D results in a separate RSVP reservation request.**
    - **These two requests are merged into the effective flowspec 3B, which is used to make the reservation on interface D.**

Examples of Reservation Styles (Continued)

- **To forward the reservation requests upstream, the reservations on the interfaces C and D are merged; as a result, the larger flowspec 4B is forwarded upstream to each previous hop.**
  - B is an arbitrary QoS parameter in all following examples

Send	Reserve	Receive
*WF(* {4B})* <== A	C * *{4B}*	C <== *WF(* {4B})*
*WF(*{4B})* <== B	D * *{3B}*	D <== *WF(* {3B})* <== *WF(* {2B})*

Examples of Reservation Styles

- **Fixed-Filter (FF) style reservations.**
  - The flow descriptors for senders S_2 and S_3, received from outgoing interfaces C and D, are packed into the request forwarded to previous hop B.
  - The three different flow descriptors for sender S_1 are merged into the single request FF(S_1{ 4B}), which is sent to previous hop A.
  - For each outgoing interface, there is a separate reservation for each source that has been requested, but this reservation is shared among all the receivers that made the request.

Send	Reserve	Receive
FF(S1 *{4B})* <== A	C S1 *{4B}* S2 *{5B}*	C <== *FF(* S1 *{4B}*, S2 *{5B}* *)*
FF(S2 *{5B}*, S3 *{B})* <== B	D S1 *{3B}* S3 *{B}*	D <== *FF(* S1 *{3B}*, S3 *{B}* *)* <== *FF(* S1 *{B}* *)*

Examples of Reservation Styles

- **An example of Shared-Explicit (SE) style reservations.**
  - When SE-style reservations are merged, the resulting filter spec is the union of the original filter specs.

Send	Reserve	Receive
SE(S1 *{3B}* *)* <== A	C (S1,S2) *{B}*	C <== *SE(* (S1,S2) *{B}* *)*
SE((S2,S3) *{3B})* <== B	D (S1,S2,S3) *{3B}*	D <== *SE(* (S1,S3) *{3B}* *)* <== *SE(* S2 *{2B}* *)*

Examples of Reservation Styles

- **The three earlier examples assumed that data packets from S_1, S_2, and S_3 are routed to both outgoing interfaces.**
  - Let's assume that data packets from S_2 and S_3 are not forwarded to interface C, e.g., because the network topology provides a shorter path for these senders towards R_1, not traversing this node.
  - The table below shows WF style reservations under this assumption. Since there is no route from B to C, the reservation forwarded out interface B considers only the reservation on interface D.

Send	Reserve	Receive
*WF(* {4B}) <==* A	C * {B}	C <== *WF(*{4B})*
*WF(* {3B}) <==* B	D * {3B}	D <== *WF(*{3B})* <== *WF(*{2B})*

RSVP: A list of outstanding issues

- **We have only discussed the principal features of the RSVP protocol. We have not discussed any actual implementation details and we have also omitted the following important architectural issues:**
  - Rules for Merging Flowspecs
  - Router Soft State description
  - Teardown of connections
  - Support for local Policy and Security
  - Automatic RSVP Tunneling
  - Description of the Host Model
- **All these details can be found in the Internet Draft documents.**

Reservation Protocol - Part II

marek@npac.syr.edu, (315) 443-4879 http://www.npac.syr.edu

RSVP - Part II

- This module covers details of the Reservation Protocol ommitted from in the first module. he following issues are addressed:
  - basic RSVP messages, the meaning of flow specification elements (Rspec and Tspec), and the nature of the "contract" between the data flow and the service
  - router's soft state, flowspec merging, RSVP tunneling, reservation confirmation, reservation teardown, and the host model

RSVP Protocol Mechanisms

- RSVP Messages
  - Two basic RSVP message types: Resv and Path.
    - Each receiver host sends RSVP reservation request (Resv) messages upstream towards the senders.
    - Resv messages must follow exactly the reverse of the routes the data packets will use, upstream to all the sender hosts included in the sender selection.
    - Resv messages are delivered to the sender hosts themselves so that the hosts can set up appropriate traffic control parameters for the first hop.

RSVP Protocol Mechanisms

- RSVP Messages
  - Two basic RSVP message types: Resv and Path.
    - Each RSVP sender host transmits RSVP Path messages downstream along the uni-/multicast routes provided by the routing protocol(s), following the paths of the data.
    - Path messages store "path state" in each node along the way. This path state includes at least the unicast IP address of the previous hop node, which is used to route the Resv messages hop-by-hop in the reverse direction.

RSVP Messages

- Path messages are sent with the same source and destination addresses as the data, so that they will be routed correctly through non-RSVP clouds.
- Resv messages are sent hop-by-hop. Each RSVP-speaking node forwards a Resv message to the unicast address of a previous RSVP hop.

RSVP Protocol Messages

- Path message: additional information:
  - Sender Template
    - Sender Template describes the format of data packets that the sender will originate. This template is in the form of a filter spec that could be used to select this sender's packets from others in the same session on the same link.
    - at present, only the sender IP address and optionally the UDP/TCP sender port can be specified
  - Sender Tspec
    - (Required) Sender Tspec which defines the traffic characteristics of the data stream that the sender will generate. Tspec is used by traffic control to prevent over-reservation or unnecessary Admission Control failure on upstream links.

RSVP Protocol Messages

- Adspec: a Path message may optionally carry a package of OPWA advertising information.
  - An Adspec received in a Path message is passed to the local traffic control, which returns it updated
  - The updated version is then forwarded in Path messages sent downstream.
  - Sender Tspec is never changed, Adspec may be changed by intermediate nodes
  - Adspec includes both parameters describing the properties of the data path (availability of specific QoS services) and parameters required by specific QoS control services to operate correctly.
  - Adspec is finally passed to the application via RSVP API.

RSVP Protocol Messages

- Resv Messages anatomy
  - Flow descriptor consists of flowspec and filter spec
  - Flow is defined a set of packets traversing the network all of which are covered by the same request for QoS
  - Flowspec consists of two sets of parameters: Rspec and Tspec
    - Rspec defines desired QoS
    - Tspec describes the data flow
  - Hmmm - what is this supposed to mean?

RSVP Protocol Messages

- Tspec and Rspec (Traffic and Reservation Specs)
  - Tspec has two incarnations: Receiver_Tspec in Resv messages and Sender_Tspec in Path messages
  - Tspec is is a description of the traffic pattern for which service is being requested
  - Tspec is one side of a "contract" between the data flow and the service.
  - Service module agrees to provide a specific QoS if and only if the flow's data traffic remains conformant to the Tspec
  - Example: upper bound on the peak rate

RSVP Protocol Messages

- Sender and Receiver Tspec
  - Receiver Tspec is a basis for the "contract" between the flow and the service
  - Sender Tspec is used to verify Receiver Tspec to avoid excessive reservations
- Rspec - Reservation Specification
  - Usually invokes a particular service class
  - Contents of the request depends on the service and is of little interest to RSVP (but not in general!)

Merging Flowspecs

- a single physical interface may receive multiple reservation requests from different next hops for the same session and with the same filter spec
- RSVP should install only one reservation on that interface. The installed reservation should have an effective flowspec that is the "largest" of the flowspecs requested by the different next hops
- Similarly, a Resv message forwarded to a previous hop should carry a flowspec that is the "largest" of the flowspecs requested by the different next hops
- in certain cases the "smallest" is taken rather than the largest.

Merging Flowspecs

- Flowspec merging requires calculation of the "largest" of a set of flowspecs.
  - Flowspecs are generally multi- dimensional vectors (they may contain both Tspec and Rspec components, each of which may itself be multi-dimensional), it may not be possible to strictly order two flowspecs.
  - For example, if one request calls for a higher bandwidth and another calls for a tighter delay bound, one is not "larger" than the other. In such a case, instead of taking the larger, RSVP must compute and use a third flowspec that is at least as large as each.
  - Flowspecs, Tspecs, and Adspecs are opaque to RSVP. Therefore, calculations above are actually performed by traffic control. The definition and implementation of the rules for comparing flowspecs are outside the definition of RSVP

RSVP's Soft State

- RSVP soft state is created and periodically refreshed by Path and Resv messages.
  - The state is deleted if no matching refresh messages arrive before the expiration of a "cleanup timeout" interval.
  - State may also be deleted by an explicit "teardown" message.
  - At the expiration of each "refresh timeout" period and after a state change, RSVP scans its state to build and forward Path and Resv refresh messages to succeeding hops.
  - When a route changes, the next Path message will initialize the path state on the new route, and future Resv messages will establish reservation state there
    - the state on the now-unused segment of the route will time out.

RSVP's Soft State

- RSVP sends its messages as IP datagrams with no reliability enhancement:
  - Periodic transmission of refresh messages by hosts and routers is expected to handle the occasional loss of an RSVP message.
    - If the effective cleanup timeout is set to K times the refresh timeout period, then RSVP can tolerate K-1 successive RSVP packet losses without falsely erasing a reservation.
- The state maintained by RSVP is dynamic:
  - To change the set of senders Si or to change any QoS request, a host simply starts sending revised Path and/or Resv messages. The result will be an appropriate adjustment in the RSVP state in all nodes along the path.

RSVP's Soft State

- In steady state, refreshing is performed hop-by-hop, to allow merging.
  - When the received state differs from the stored state, the stored state is updated.
  - If this update results in modification of state to be forwarded in refresh messages, these refresh messages must be generated and forwarded immediately, so that state changes can be propagated end-to-end without delay.
  - Propagation of a change stops when and if it reaches a point where merging causes no resulting state change.
    - This minimizes RSVP control traffic due to changes and is essential for scaling to large multicast groups.

Teardown

- Two types of RSVP teardown message:
  - PathTear message travels towards all receivers downstream from its point of initiation and deletes path state, as well as all dependent reservation state, along the way.
  - ResvTear message deletes reservation state and travels towards all senders upstream from its point of initiation.
- A teardown request may be initiated either by sender or receiver, or by a router as the result of state timeout.
  - State change will be propagated immediately to the next node, but only if there will be a net change after merging. As a result, an ResvTear message will prune the reservation state back (only) as far as possible.

Confirmations

- To request a confirmation for its reservation request, receiver R_j includes in the Resv message a confirmation request object containing R_j's IP address.
  - At each merge point, only the largest flowspec and any accompanying confirmation-request object is forwarded upstream.
  - If the reservation request from Rj is equal to or smaller than the reservation in place on a node, its Resv is not forwarded further
    - If the Resv included a confirmation-request object, a ResvConf message is sent back to R_j.

Confirmations

- Confirmation mechanism implications:
  - A new reservation request with a flowspec larger than any in place for a session will normally result in either a ResvErr or a ResvConf message back to the receiver from each sender. In this case, the ResvConf message will be an end-to-end confirmation.
  - The receipt of a ResvConf gives no guarantees:
    - Assume the first two reservation requests from receivers R1 and R2 arrive at the node where they are merged.
    - R2, whose reservation was the second to arrive at that node, may receive a ResvConf from that node while R1's request has not yet propagated all the way to a matching sender and may still fail.
    - Thus, R2 may receive a ResvConf although there is no end-to-end reservation in place. R2 may also receive a ResvConf followed by a ResvErr.

RSVP Tunneling

- RSVP must provide correct protocol operation across an arbitrary "cloud" of non-RSVP routers.
  - If such a cloud has sufficient capacity, it may still provide acceptable realtime service.
  - RSVP automatically tunnels through non-RSVP clouds since routing and reservation functions are independent
    - When a Path message traverses a non-RSVP cloud, it carries to the next RSVP-capable node the IP address of the last RSVP-capable router before entering the cloud. This effectively constructs a tunnel through the cloud for Resv messages, which can then be forwarded directly to the next RSVP- capable router on the path(s) back towards the source.

RSVP's Host model

- Before a session can be created, the session identification, comprised of DestAddress and perhaps the generalized destination port, must be assigned and communicated to all the senders and receivers by some out-of-band mechanism.
- How this is done is not part of the RSVP specification
- For multicast sessions, IGMP may be used

RSVP's Host model

- When an RSVP session is being set up, the following events happen at the end systems.
  1. A receiver joins the multicast group specified by DestAddress, using IGMP.
  2. A potential sender starts sending RSVP Path messages to the DestAddress.
  3. A receiver application receives a Path message.
  4. A receiver starts sending appropriate Resv messages, specifying the desired flow descriptors.
  5. A sender application receives a Resv message.
  6. A sender starts sending data packets.

RSVP's Host Model

- 1. and 2. may happen in either order.
- Suppose that a new sender starts sending data (6) but there are no multicast routes because no receivers have joined the group (1).
  - The data will be dropped at some router node until receivers(s) appear.
- Suppose that a new sender starts sending Path messages (2) and data (6) simultaneously, and there are receivers but no Resv messages have reached the sender yet.
  - The initial data may arrive at receivers without the desired QoS. The sender could mitigate this problem by awaiting arrival of the first Resv message (5); however, receivers that are farther away may not have reservations in place yet.

RSVP's Host Model

- If a receiver starts sending Resv messages (4) before receiving any Path messages (3), RSVP will return error messages to the receiver.
- The receiver may simply choose to ignore such error messages, or it may avoid them by waiting for Path messages before sending Resv messages.
- A specific application program interface (API) for RSVP is not defined in its protocol spec since it may be host system dependent.

RTP: Real-time Transport Protocol

Synchronizing Media Delivery on Packet Networks

Marek Podgorny
NPAC
Syracuse University
111 College Place
Syracuse
New York 13244-4100

RTP: Basic Functionality

- Provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video.
- The services include payload type identification, sequence numbering, timestamping and delivery monitoring.
- Primarily designed to satisfy the needs of multi-participant multimedia conferences.
- May be used with any suitable underlying network or transport protocols.
- Supports data transfer to multiple destinations using multicast distribution if provided by the underlying network.
- RTP is an application level protocol. It is not implemented in the OS kernel.
- RTP is extensible: it provides for the profile specific extensions to the protocol headers.

RTP: Basic Functionality

- **RTP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees, but relies on lower-layer services to do so.**
  - **It does not guarantee delivery or prevent out-of-order delivery, nor does it assume that the underlying network is reliable and delivers packets in sequence.**
  - **However, the sequence numbers included in RTP allow the receiver to reconstruct the sender's packet sequence. The sequence numbers might also be used to determine the proper location of a packet, for example in video decoding, without necessarily decoding packets in sequence.**

RTP: Basic Functionality (Continued)

- **Applications typically run RTP on top of UDP to make use of its multiplexing and checksum services.**
- **RTP design follows the principles of application level framing and integrated layer processing:**
  - RTP is intended to be malleable to provide the information required by a particular application and will often be integrated into the application processing rather than being implemented as a separate layer.
  - RTP is a protocol framework that is deliberately not complete. RTP can be tailored for any desired profile through modifications and/or additions to the headers as needed.

RTP: Basic Fuctionality

- **RTP consists of two closely-linked parts:**
  - **The real-time transport protocol (RTP), to carry data that has real-time properties.**
  - **The RTP control protocol (RTCP), to monitor the quality of service and to convey information about the participants in an on-going session. The latter aspect of RTCP may be sufficient for "loosely controlled" sessions, i.e., where there is no explicit membership control and set-up, but it is not necessarily intended to support all of an application's control communication requirements.**

RTP: Basic Fuctionality (Continued)

- **As RTP is extensible, a complete specification for a particular applications requires one or more accompanying documents:**
  - A profile specification document, which defines a set of payload type codes and their mapping to payload formats (e.g., media encodings). A profile may also define extensions or modifications to RTP that are specific to a particular class of applications. Typically an application will operate under only one profile. A typical profile, for audio and video data, is subject of an associated Internet draft (1890).
  - Payload format specification documents, which define how a particular payload, such as an audio or video encoding, is to be carried in RTP.

RTP: Rationale

- **Why do we need a protocol like RTP?**
  - Synchronization of multiple media streams is an absolutely fundamental property of multimedia delivery. There is no multimedia without sync.
  - There is currently no standard to provide media synchronization. Instead, there are a three dozen+ of proprietary, non-interoperable solutions:
    - **Audio on Demand: RealAudio, Streamworks, IWave, TrueSpeech, Voxware, Accousting Player**
    - **'Live' Audio Products: RealAudio 2.0, Streamworks, Cyber Radio 1**
    - **'Live' Video Products: Streamworks, FreeVue, Video Mosaic**

RTP: Rationale (Continued)

- **Video on Demand: VDOLive, Streamworks**
- **Video Conferencing: FreeVue, VidCall, CU-SeeMe**
- **Other Multimedia Tool: InSoft, Phylon, PlayLink, Voyager CDLink, Play Inc's - Snappy, I-Animate, Asymetrix Web 3D, Publishers Depot**
- **MANY OTHERS! (MPEG, QuickTime, Video for Windows....)**

RTP: Rationale (Continued)

- **Can we use existing protocols to achieve media synchronization?**
  - **Absolutely! - see the previous slide....**
  - **For delivering audio and video for playback, TCP may be appropriate. Also, with sufficiently long buffering and adequate average throughput, near-real-time delivery using TCP can be successful, as practiced by the Netscape WWW browser.**

RTP: Rationale (Continued)

- **Why can't we just use TCP for audio and video?**
  - For real-time delivery of audio and video, TCP and other reliable transport protocols such as XTP are inappropriate. The three main reasons are:
    - **1. Reliable transmission is inappropriate for delay-sensitive data such as real-time audio and video.**
      - By the time the sender has discovered the missing packet and retransmitted it, at least one round-trip time has elapsed. The receiver either has to wait for the retransmission, increasing delay and incurring an audible gap in playout, or discard the retransmitted packet, defeating the TCP mechanism. Standard TCP implementations force the receiver application to wait, so that packet losses would always yield increased delay.

RTP: Rationale (Continued)

- **Three TCP deficiencies- continued**
  - 2. TCP cannot support multicast.
  - 3. The TCP congestion control mechanisms decreases the congestion window when packet losses are detected ("slow start"). Audio and video, on the other hand, have "natural" rates that cannot be suddenly decreased without starving the receiver.

RTP: Rationale (Continued)

- **Standard PCM audio requires 64 kb/s, plus any header overhead, and cannot be delivered in less than that.**
- **Video could be more easily throttled simply by slowing the acquisition of frames at the sender when the transmitter's send buffer is full, with the corresponding delay.**
- **The correct congestion response for these media is to change the audio/video encoding, video frame rate, or video image size at the transmitter, based on feedback received through RTCP receiver report packets.**

RTP: Rationale (Continued)

- Additional problems with reliable protocols:
  - The reliable transport protocols do not contain the necessary timestamp and encoding information needed by the receiving application, so that they cannot replace RTP. (They would not need the sequence number as these protocols assure that no losses or reordering takes place.)
  - While LANs often have sufficient bandwidth and low enough losses not to trigger these problems, TCP/XTP does not offer any advantages in that scenario either, except for the recovery from rare packet losses. Even in a LAN with no losses, TCP/XTP would suffer from the initial slow start delay.

RTP: Rationale (Continued)

- RTP has no protocol state by itself and can thus be used over either connection-less networks, such as IP/UDP, or connection-oriented networks, such as XTP, ST-II or ATM (AAL3/4 or AAL5). Many real-time multimedia applications use multicast with a large fan-out, e.g., several hundred to thousands for a lecture or concert. Connection-oriented protocols like XTP have difficulty scaling to such a large number of receivers.
- TCP and XTP headers are larger than a UDP header (40 bytes for TCP and XTP 3.6, 32 bytes for XTP 4.0, compared to 8 bytes).

RTP: General Architectural Features

- **RTP is a transport protocol:**
  - **RTP has important properties of a transport protocol: it runs on end systems and it provides demultiplexing.**
  - **RTP differs from transport protocols like TCP in that it does not offer any form of reliability or a protocol-defined flow/congestion control. However, it provides the necessary hooks for adding reliability, where appropriate, and flow/congestion control (application level framing).**
  - **Although RTP so far has been mostly implemented within applications, this has no bearing on its role. (TCP implemented as part of an application rather than the operating system kernel would still be a transport protocol!)**

RTP: General Architectural Features (Continued)

- **RTP is real-time as much as it can be:**
  - No end-to-end protocol can ensure in-time delivery. This always requires the support of lower layers that actually have control over resources in switches and routers. RTP provides functionality suited for carrying real-time content, e.g., a timestamp and control mechanisms for synchronizing different streams with timing properties.

RTP: General Architectural Features (Continued)

- **Is RTP a reliable or an unreliable protocol?**
  - **RTP does not currently define any mechanisms for recovering for packet loss.**
    - Such mechanisms are likely to be highly dependent on the packet content.
    - For example, for audio, one might add low-bit-rate redundancy, offset in time.
    - For other applications, retransmission of lost packets may be appropriate. This requires no additions to RTP as the protocol probably has the necessary header information (like sequence numbers) for some forms of error recovery by retransmission.

RTP: General Architectural Features (Continued)

- **RTP can run over both IPng and ATM AAL**
  - RTP contains no specific assumptions about the capabilities of the lower layers, except that they provide framing. It contains no network-layer addresses, so that RTP is not affected by addressing changes.
    - **Any additional lower-layer capabilities such as security or quality-of-service guarantees (RSVP!) can be used by applications employing RTP.**
    - **There are several implementations of video tools that run RTP directly over AAL5.**

RTP: Related Standardization Efforts

- **There is a number of multimedia-centered standardization efforts:**
  - Conference control, application and data sharing:
    - **T.120: Introduction to the audiographics and audiovisual conferencing recommendations.**
    - **T.121: Generic application template.**
    - **T.122: Multipoint communication service for audiographics and audiovisual conferencing service definition**
    - **T.123: Protocol stack for audiographics and audiovisual teleconference applications.**

RTP: Related Standardization Efforts (Continued)

- **T.124: Generic conference control.**
- **T.125: Multipoint communication service protocol specification.**
- **T.126: Still image protocol specification.**
- **T.127: Multipoint binary file transfer protocol.**

RTP: Related Standardization Efforts (Continued)

- **For conferencing over ISDN:**
  - H.221: Frame structure for a 64 to 1920 kbit/s channel in audiovisual teleservices.
  - H.320: Framework for transmitting audio and video over circuit-switched digital networks (primarily ISDN).
  - H.323: H.320 over LAN.
- **Media formats:**
  - G.711: Audio encoding at 64 kb/s (mu-law and A-law).
  - H.261: Video encoding.
  - H.263: Video encoding, improved version of H.261.
  - H.324: Audio and video over POTS at less than 20 kb/s.

RTP Use Scenarios

- **Audio conference over Internet**
  - **The conference coordinator must obtain a multicast address and two port numbers. This information must be distributed to conference participants**
  - **The audio conferencing application sends audio data in small chunks. Each chunk of audio data is preceded by an RTP header; RTP header and data are in turn contained in a UDP packet. The RTP header indicates type of audio encoding contained in each packet so that senders can change the encoding during a conference.**

RTP Use Scenarios (Continued)

- **The Internet occasionally loses and reorders packets and delays them by variable amounts of time. RTP header contains timing information and a sequence number that allow the receivers to reconstruct the timing produced by the source, so that chunks of audio are contiguously played out the speaker. Timing reconstruction is performed separately for each source of RTP packets in the conference. The sequence number can be used be the receiver to estimate how many packets are being lost.**

RTP Use Scenarios (Continued)

- Conference can be joined or abandoned as it proceeds. It is useful to know who is participating at any moment and how well they are receiving the audio data. Hence, each instance of the audio application in the conference periodically multicasts a reception report plus the name of its user on the RTCP (control) port. The reception report may be used to control adaptive encodings.

RTP Use Scenarios (Continued)

- **Audio and Video Conference**
  - If both audio and video media are used in a conference, they are as separate RTP sessions. RTCP packets are transmitted for each medium using two different UDP port pairs and/or multicast addresses. There is no direct coupling at the RTP level between the audio and video sessions, except that a user participating in both sessions should use the same distinguished (canonical) name in the RTCP packets for both so that the sessions can be associated.
  - One motivation for this separation is to allow some participants in the conference to receive only one medium if they choose. Despite the separation, synchronized playback of a source's audio and video can be achieved using timing information carried in the RTCP packets for both sessions.

RTP: Mixers

- **Heterogenous networks: different sites wish to obtain data in different formats**
  - **Assume a user with a low-speed link to the majority of the conference participants connected via a high-speed network. Instead of enforcing a lower-bandwidth audio encoding, an RTP-level relay called a mixer may be placed near the low-bandwidth area.**

RTP: Mixers (Converted)

- This mixer resynchronizes incoming audio packets to reconstruct the constant spacing generated by the sender, mixes the reconstructed audio streams into a single stream, translates the audio encoding to a lower bandwidth one and forwards the lower- bandwidth packet stream across the low-speed link.
- These packets might be unicast to a single recipient or multicast on a different address to multiple recipients. The RTP header includes a means for mixers to identify the sources that contributed to a mixed packet so that correct talker indication can be provided at the receivers.

RTP: Translators

- Sites not supporting multicast
  - Some of the intended conference participants might not be directly reachable via IP multicast. For example, they might be behind an application-level firewall that will not let any IP packets pass. For these sites, a translator may be used.
  - Two translators are installed, one on either side of the firewall, with the outside one funneling all multicast packets received through a secure connection to the translator inside the firewall. The translator inside the firewall sends them again as multicast packets to a multicast group restricted to the site's internal network.

RTP: Translators (Continued)

- **Mixers and translators may be designed for a variety of purposes:**
  - Video mixer that scales the images of individual people in separate video streams and composites them into one video stream to simulate a group scene.
  - Connection of a group of hosts speaking only IP/UDP to a group of hosts that understand only ST-II, or the packet-by-packet encoding translation of video streams from individual sources without resynchronization or mixing.

RTP Components

- **RTP packet:**
  - **A data packet consisting of the fixed RTP header, a possibly empty list of contributing sources , and the payload data. Some underlying protocols may require an encapsulation of the RTP packet to be defined. Typically one packet of the underlying protocol contains a single RTP packet.**
- **RTCP packet:**
  - **A control packet consisting of a fixed header part similar to that of RTP data packets, followed by structured elements that vary depending upon the RTCP packet type. Typically, multiple RTCP packets are sent together as a compound RTCP packet in a single packet of the underlying protocol.**

RTP Components (Continued)

- **Transport address:**
  - The combination of a network address and port that identifies a transport-level endpoint, for example an IP address and a UDP port.
- **RTP session:**
  - The association among a set of participants communicating with RTP. For each participant, the session is defined by a particular pair of destination transport addresses (one network address plus a port pair for RTP and RTCP). In a multimedia session, each medium is carried in a separate RTP session with its own RTCP packets. The multiple RTP sessions are distinguished by different port number pairs and/or different multicast addresses.

RTP Components (Continued)

- **Synchronization source (SSRC):**
  - **The source of a stream of RTP packets, identified by a 32-bit numeric SSRC identifier carried in the RTP header so as not to be dependent upon the network address. All packets from a synchronization source form part of the same timing and sequence number space, so a receiver groups packets by synchronization source for playback.**
  - **Examples of synchronization sources include the sender of a stream of packets derived from a signal source such as a microphone or a camera, or an RTP mixer.**

RTP Components (Continued)

- A synchronization source may change its data format, e.g., audio encoding, over time. The SSRC identifier is a randomly chosen value meant to be globally unique within a particular RTP session. A participant need not use the same SSRC identifier for all the RTP sessions in a multimedia session; the binding of the SSRC identifiers is provided through RTCP. If a participant generates multiple streams in one RTP session, for example from separate video cameras, each must be identified as a different SSRC.

RTP Components (Continued)

Contributing source (CSRC):

- A source of a stream of RTP packets that has contributed to the combined stream produced by an RTP mixer. The mixer inserts a list of the SSRC identifiers of the sources that contributed to the generation of a particular packet into the RTP header of that packet.
- An example application is audio conferencing where a mixer indicates all the talkers whose speech was combined to produce the outgoing packet, allowing the receiver to indicate the current talker, even though all the audio packets contain the same SSRC identifier (that of the mixer).

RTP Components (Continued)

- **End system:**
  - An application that generates the content to be sent in RTP packets and/or consumes the content of received RTP packets. An end system can act as one or more synchronization sources in a particular RTP session, but typically only one.
- **Mixer:**
  - An intermediate system that receives RTP packets from one or more sources, possibly changes the data format, combines the packets in some manner and then forwards a new RTP packet. Mixer will make timing adjustments among the streams and generate its own timing for the combined stream.

RTP Components (Continued)

- **Translator:**
  - An intermediate system that forwards RTP packets with their synchronization source identifier intact. Examples of translators: devices that convert encodings without mixing, replicators from multicast to unicast, and application- level filters in firewalls.
- **Monitor:**
  - An application that receives RTCP packets sent by participants in an RTP session, in particular the reception reports, and estimates the current quality of service for distribution monitoring, fault diagnosis and long-term statistics. The monitor function is likely to be built into the application(s) participating in the session, but may also be a separate application that does not otherwise participate and does not send or receive the RTP data packets. These are called third party monitors.

RTP Components (Continued)

- **Non-RTP means:**
  - Protocols and mechanisms that may be needed in addition to RTP to provide a usable service. In particular for multimedia conferences a conference control application may distribute multicast addresses and keys for encryption, negotiate the encryption algorithm to be used, and define dynamic mappings between RTP payload type values and the payload formats they represent for formats that do not have a predefined payload type value. For simple applications, electronic mail or a conference database may also be used.

RTCP: RTP Control Protocol

- **The RTP control protocol (RTCP) is based on the periodic transmission of control packets to all participants in the session, using the same distribution mechanism as the data packets.**
  - RTCP performs four functions:
    - **(1).The primary function is to provide feedback on the quality of the data distribution.**
      - The feedback may be directly useful for control of adaptive encodings. Sending reception feedback reports to all participants allows one who is observing problems to evaluate whether those problems are local or global.
      - With a distribution mechanism like IP multicast, it is also possible for an entity such as a network service provider who is not otherwise involved in the session to receive the feedback information and act as a third-party monitor to diagnose network problems.

RTCP: RTP Control Protocol (Continued)

- **(2) RTCP carries a persistent transport-level identifier for an RTP source called the canonical name or CNAME.**
  - Receivers require the CNAME to keep track of each participant. Receivers also require the CNAME to associate multiple data streams from a given participant in a set of related RTP sessions, for example to synchronize audio and video.

RTCP: RTP Control Protocol (Continued)

- **RTCP functions, continued:**
  - (3) The first two functions require that all participants send RTCP packets, therefore the rate must be controlled in order for RTP to scale up to a large number of participants. By having each participant send its control packets to all the others, each can independently observe the number of participants. This number is used to calculate the rate at which the packets are sent.

RTCP: RTP Control Protocol (Continued)

- **(4) A fourth, optional function is to convey minimal session control information, for example participant identification to be displayed in the user interface.**
  - This is most likely to be useful in "loosely controlled" sessions where participants enter and leave without membership control or parameter negotiation. RTCP serves as a convenient channel to reach all the participants, but it is not necessarily expected to support all the control communication requirements of an application.

RTCP: RTP Control Protocol (Continued)

- **Functions 1-3 are mandatory when RTP is used in the IP multicast environment, and are recommended for all environments.**
  - RTP application designers are advised to avoid mechanisms that can only work in unicast mode and will not scale to larger numbers.

RTCP: RTP Control Protocol (Continued)

- **RTCP Transmission Interval**
  - RTP is designed to allow an application to scale automatically over session sizes ranging from a few participants to thousands.
    - **Note that the data traffic is inherently self-limiting: only one or two people will speak at a time, so with multicast distribution the data rate on any given link remains relatively constant independent of the number of participants.**

RTCP: RTP Control Protocol (Continued)

- **The control traffic is not self-limiting. If the reception reports from each participant were sent at a constant rate, the control traffic would grow linearly with the number of participants.**
- **For each session, the data traffic is subject to an aggregate limit called the "session bandwidth" to be divided among the participants.**
- **The control traffic should be limited to a small and known fraction of the session bandwidth: small so that the primary function of the transport protocol to carry data is not impaired; known so that the control traffic can be included in the bandwidth specification given to a resource reservation protocol, and so that each participant can independently calculate its share.**

RTP: Translators and Mixers

- **An RTP translator/mixer connects two or more transport-level "clouds".**
  - **Typically, each cloud is defined by a common network and transport protocol (e.g., IP/UDP), multicast address or pair of unicast addresses, and transport level destination port. One system may serve as a translator or mixer for a number of RTP sessions, but each is considered a logically separate entity.**
    - Each of the clouds connected by translators and mixers participating in one RTP session either must be distinct from all the others in at least one of these parameters (protocol, address, port), or must be isolated at the network level from the others.

RTP: Translators and Mixers (Continued)

- **Hence, there must not be multiple translators or mixers connected in parallel unless by some arrangement they partition the set of sources to be forwarded.**
- **Similarly, all RTP end systems that can communicate through one or more RTP translators or mixers share the same SSRC space, that is, the SSRC identifiers must be unique among all these end systems.**
- **There may be many varieties of translators and mixers designed for different purposes and applications. Some examples are to add or remove encryption, change the encoding of the data or the underlying protocols, or replicate between a multicast address and one or more unicast addresses.**

RTP: Translators and Mixers

- Translators and mixers: a translator passes through the data streams from different sources separately; a mixer combines them to form one new stream:
  - Translator: Forwards RTP packets with their SSRC identifier intact. This makes it possible for receivers to identify individual sources even though packets from all the sources pass through the same translator and carry the translator's network source address. Encoding changes will result in change of the RTP data payload type and timestamp. If multiple data packets are re-encoded into one, a translator must assign new sequence numbers to the outgoing packets. Receivers cannot detect the presence of a translator .

RTP: Translators and Mixers (Continued)

- Mixer: Receives streams of RTP data packets from one or more sources, possibly changes the data format, combines the streams in some manner and then forwards the combined stream. Mixer will make timing adjustments among the streams and generate its own timing for the combined stream, so it is the synchronization source. Thus, all data packets forwarded by a mixer will be marked with the mixer's own SSRC identifier. To preserve the identity of the original sources contributing to the mixed packet, the mixer should modify appropriate elements of the RTP headers.

RTP over other transport protocols

- **The following rules should be observed:**
  - **RTP relies on the underlying protocol(s) to provide demultiplexing of RTP data and RTCP control streams.**
    - For UDP and similar protocols, RTP uses an even port number and the corresponding RTCP stream uses the next higher (odd) port number. If an application is supplied with an odd number for use as the RTP port, it should replace this number with the next lower (even) number.
  - **RTP data packets contain no length field or other delineation, therefore RTP relies on the underlying protocol(s) to provide a length indication. The maximum length of RTP packets is limited only by the underlying protocols.**

RTP over other transport protocols (Continued)

- **If RTP packets are to be carried in an underlying protocol that provides the abstraction of a continuous octet stream rather than messages (packets), an encapsulation of the RTP packets must be defined to provide a framing mechanism.**
  - A profile may specify a framing method even when RTP is carried in protocols that do provide framing in order to allow carrying several RTP packets in one lower-layer protocol data unit, such as a UDP packet. Carrying several RTP packets in one network or transport packet reduces header overhead and may simplify synchronization between different streams.

RTP Profiles and Payload Format Specifications

- **A complete specification of RTP for a particular application will require one or more companion documents: profiles and payload format specifications.**
  - **RTP may be used for a variety of applications with somewhat differing requirements.**
    - The flexibility to adapt to those requirements is provided by allowing multiple choices in the main protocol specification, then selecting the appropriate choices or defining extensions for a particular environment and class of applications in a separate profile document.

RTP Profiles and Payload Format Specifications (Continued)

- **Typically an application will operate under only one profile so there is no explicit indication of which profile is in use. A typical profile is the profile for audio and video applications.**
- **The second type of companion document is a payload format specification, which defines how a particular kind of payload data, such as H.261 encoded video, should be carried in RTP.**
  - These documents are typically titled "RTP Payload Format for XYZ Audio/Video Encoding". Payload formats may be useful under multiple profiles and may therefore be defined independently of any particular profile. The profile documents are then responsible for assigning a default mapping of that format to a payload type value if needed.

Research Efforts

- **Multicast address allocation:**
  - Eventually many thousands of multicast sessions may exist concurrently. Currently the IPv4 multicast address space is very limited and thus requires careful global allocation to avoid collisions. The IPv6 multicast address space is much larger, supporting administrative scoping and may allow random allocation.
- **Scalable multicast routing:**
  - Multicast routing needs to work for both a very large number of small groups and a smaller number of large groups; without routers not on the multicast tree having to know about groups.
- **Compensating for packet loss:**
  - For the foreseeable future, the Internet will have areas and times of high packet loss (1% to 10% and higher).
- **Playout delay compensation:**
  - End systems need to compensate for network delay variations.
- **Synchronization of different media:**
  - Several audio and video streams coming from one or, less commonly, several sources need to be synchronized (lip sync).
- **(Semi)reliable multicast for (near) real-time services:**
  - In some circumstances, such as near-real-time video and audio on demand, it may be possible to improve quality by retransmission.

Research Efforts

- **Internet integrated service models:**
  - It is not clear yet how many different services beyond the current best effort model are needed to support a wide variety of real-time and near real-time services.
- **Making use of ATM quality-of-service features:**
  - It is desirable if Internet services can make use of the quality-of-service guarantees offered by subnetworks.
- **New conference control mechanisms:**
  - Different modes of establishing one-on-one and group communication are being explored.

Research Efforts (Continued)

- **Composition of multimedia applications:**
  - Rather than having a single application that handles all media, it may be preferable to compose conferencing and other multimedia applications from reusable building blocks.
- **Integration of real-time services with WWW:**
  - Both delivery and interactive multimedia services need to be more closely woven into the World Wide Web.
- **Interoperation with POTS, ISDN, T.120, ...:**
  - For at least the next year, the Internet will not replace the telephone network for voice calls. Thus, interoperability with plain old telephony, ISDN and some of the ITU-defined circuit-switched video conferencing equipment is needed.

9. Software packages

The software packages developed during the course of this project include:

1) Video server for NT workstations (production version)
2) Video server for the SGI workstation
3) Oasis video server software
4) Video client for the Optibase PCMotion M PEG decoder, Windows 3.1
5) ActiveMovie video client (Windows 95, Windows NT)
6) A packet of routines implementing database back-end interface.

10. Appendices

10.1 Appendix 1: CBR Transmission of VBR Encoded Continuous Media

This Appendix presents three articles that have resulted from our research on optimized VBR stream transmission over CBR transmission lines. The papers are:

1. CBR Transmission of VBR Encoded Continuous Media in Video on Demand Servers, by M. Del Rosario and Geoffrey C. Fox (presented at the 2nd IASTED/ISMM International Conference, Palo Alto, California, August 95);
2. Constant Bit Rate Network Transmission of Variable Bit Rate Continuous Media in Video on Demand Servers, by M. DelRosario and Geoffrey C. Fox, published in Multimedia Tools and Applications 3, 215-232 (1996); and
3. m-Frame granular transport and buffer requirements for VBR encoded media in VoD servers, by M. DelRosario, M. Podgorny, and Geoffrey C. Fox, NPAC Technical Report SCCS-733, March 1996.

CBR Transmission of VBR Encoded Continuous Media in Video On-Demand Servers *

Juan Miguel del Rosario† Geoffrey Fox ‡

Presented at The Second IASTED/ISMM International Conference Distributed Multimedia Systems and Applications, Palo Alto, California, August 1995.

Northeast Parallel Architectures Center
111 College Place, RM 3-201, Syracuse University, Syracuse, NY 13244-4100

Abstract

We present an algorithm for determining the minimum buffer requirement for avoiding overflow or underflow at the client video display process, allowing the network scheduler at the VOD server to enforce a constant bit rate delivery of variable bit rate encoded media. Initial results indicate that buffer requirements for typical video streams range from 3.5 to 20.6 Megabytes. Further, we show that this approach increases the number of streams that can be multiplexed by a factor of 4.6 to 9.9 times when compared to peak and 90%-of-peak bandwidth allocation strategies.

Keywords: multimedia, video server, real-time scheduling, video-on-demand

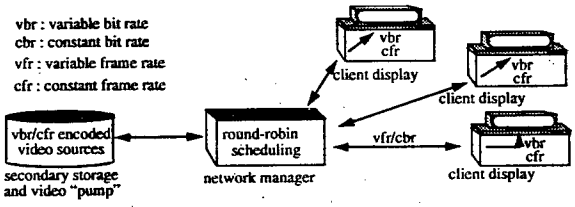


Figure 1: VOD transmission model

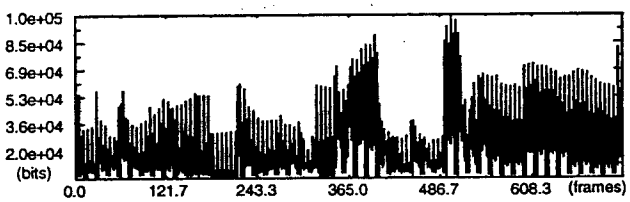


Figure 2: Sample VBR source

1 Introduction

In a VOD environment, geographically dispersed clients interactively access continuous media streams from a remote server. This type of service is an essential component to providing multimedia educational and entertainment material to the classroom and the home.

Much research has been devoted to the study of video transmission. A key issue has been whether or not using variable bit rate (VBR) encoding schemes offer a performance advantage over constant bit rate (CBR) encoding schemes. In CBR encoded video sources, picture quality parameters are adjusted to maintain a constant rate requirement for delivery. In VBR encoding, the video source is encoded with a constant picture quality. This results in a variable number of bits from frame to frame. A measure of the effectiveness of VBR transmission schemes has been formulated by Heeke [1] and is called the ***statistical multiplexing gain***; it is defined as the ratio of the number of multiplexed VBR sources to the number of multiplexed CBR sources while maintaining an equivalent subjective picture quality. When VBR sources are multiplexed without peak bandwidth reservations for each, the source bandwidth requirements and available network bandwidth fluctuate independently of each other over time, resulting in congestion (causing overall end-to-end delay) and in increased jitter. Although cell loss can be tolerated to some extent, real-time video have limited delay tolerance with very strict bounds.

In this paper we describe how selected information from an encoded video source can be used to determine buffer size bounds which enable VBR sources to be deterministically transmitted as CBR streams.

2 Transmission Model

Our model for transmitting a VBR video source as CBR traffic is illustrated in Figure 1. An example of the bursty nature of such a stream is exemplified in Figure 2.

Consider a video stream being transmitted from the

*This work was sponsored in part by the US Airforce under Rome Laboratory contract # F30602-94-C-0256. An extended version of this paper is available from the authors: email mrosario@npac.syr.edu.

†ECE Dept., Syracuse University
‡CIS Dept., Syracuse University

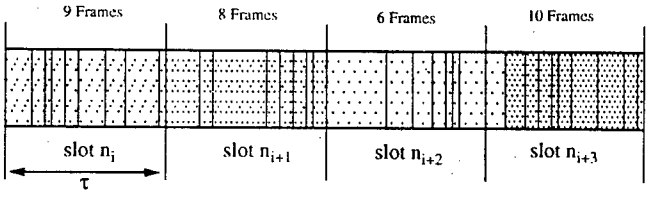


Figure 3: Variable frame rate / Constant bit rate

network manager, the *server*, to a remote display device, the *client*. Each video stream is composed of a finite amount, N_F^s, frames of varying sizes; the size of each frame in bits is denoted by X_i. Such a video stream is naturally subdivided into time *slots*, where each slot, n_i (i $\in \{0,1,2,3,\ldots,N_s\}$), refers to the time interval $[t_i, t_{i+1}]$ in the video stream transmission period. This is illustrated by Figure 3.

For any given video stream, s, its total length in bits is given by $L^s = \sum_{i=1}^{N_F^s} X_i$. Assuming a desired playout rate in frames/sec of R_F^s, the playout time in seconds for the entire stream is $T^s = \frac{N_F^s}{R_F^s}$, and the corresponding average bit rate requirement is $R_b^s = \frac{L^s}{T^s}$.

Let T_c represent the *cycle* time in seconds for the server scheduler; this is the time it takes for the scheduler to service every supported stream once in a round-robin schedule. At each cycle the scheduler must deliver $B_c^s = R_b^s x T_c$ bits of data. If the scheduler allocates to each stream some portion, t_c^s, of T_c, the server must deliver B_c^s bits of data in time t_c^s. Hence, the bandwidth requirement is equal to $R_{b_c}^s = \frac{B_c^s}{t_c^s} = R_b^s \frac{T_c}{t_c^s}$. At the client, the video stream is buffered and displayed at the appropriate constant frame rate.

Our objective is to make $R_{b_c}^s$ constant in order to achieve maximum network transmission efficiency. In this paper, we assume the simplest possible scheduling algorithm; the time, t_c^s, allocated to each stream by the server is fixed for each stream.

2.1 Buffer Allocation Requirements

Let $C(n_i)$ be the number of frames consumed by the client in slot n_i (e.g., 30 frames for 1 sec for full motion video). Let $B(n_i)$ be the number of frames in the buffer at the beginning of slot n_i (i.e., at time t_i), and let $A(n_i)$ be the number of frames arriving during slot n_i. Then, for each of the N_s time slots, the number of frames remaining in the buffer is given by the equation, $\forall i \in \{0,1,2,\ldots,N_s\}$

$$B(n_{i+1}) = B(n_i) + A(n_i) - C(n_i) \quad (1)$$

Since we transmit the video stream at a constant bit rate and the size of each frame, X_i, may vary, so that $A(n_i)$ is not constant. This is illustrated in Figure 3. For a given time slot, if $A(n_i) < C(n_i)$ we call the slot an *underflow* slot; if $A(n_i) > C(n_i)$, we call it an *overflow* slot; if $A(n_i) = C(n_i)$, we call it an *even* slot. During the playout of a video stream, there will be a combination of overflow, underflow and even slots and the display will be interrupted if the buffer is ever empty for any of the time slots. Thus we have,

$$B(n_0) + \sum_{i \in M_u} A(n_i) + \sum_{i \in M_o} A(n_i) - \sum_{i \in M_u \cup M_o} C(n_i) > 0 \quad (2)$$

where M_u, M_o, and M_e are the set of underflow, overflow, and even slots respectively, and $|M_u| + |M_o| + |M_e| = M$, where $M \in \{0,1,2,\ldots,N_s\}$. Two conditions must be accounted for in buffering, *buffer underflow*, and *buffer overflow*.

2.1.1 Buffer Underflow Condition

A buffer underflow condition comes about when for some time slot, the total number of frames that have arrived is less than that required for display (consumed). Therefore for equation 2 to hold, we require $\forall M$,

$$B(n_0) > \left| \sum_{i=0}^{N_M} (A(n_i) - C(n_i)) \right| \quad (3)$$

Hence, to prevent the buffer underflow condition, we need to transmit a number of frames into the client buffer before initiating the display. In a true implementation, the video stream will comprise a combination of overflow and underflow slots for each segment, N_M, of the stream. Therefore, since equation 3 must be true for all M, we let F be defined as,

$$F = \min_M \left(\sum_{i=0}^{N_M} (A(n_i) - C(n_i)) \right) \quad (4)$$

The underflow buffer requirement will now be given as

$$B(n_o) = \begin{cases} -F & \text{if } F < 0 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Note that we compute the buffer size here in units of frames because consumption is in terms of frames (i.e., VBR/CFR) independent of individual frame size.

2.1.2 Buffer Overflow Condition

For the buffer overflow condition, we consider buffer requirements in terms of bits rather than frames because the arrival of data is in bits (CBR/VFR) and we want to buffer against over-arrivals. We define M_u', M_o' and M_e' to be analogous to M_u, M_o, and M_e respectively, but it units of bits. We assume the initial underflow buffer, $B(n_0)$, is computed as discussed above. During overflow, for some time slot, $N_{M'}$, where $M' = |M_u'| + |M_o'| + |M_e'|$

$$\hat{B}(n_0) + \sum_{i \in M_u'} \hat{A}(n_i) + \sum_{i \in M_o'} \hat{A}(n_i) - \sum_{i \in M_u' \cup M_o'} \hat{C}(n_i) > 0 \quad (6)$$

which is equivalent to equation 2 above but in units of bits rather than frames. To prevent buffer overflow we require that $\forall M'$

$$\hat{B}_{max} \geq \hat{B}(n_0) + \max_{M'} \left(\sum_{i=0}^{N_{M'}} (\hat{A}(n_i) - \hat{C}(n_i)) \right) \quad (7)$$

Since it is important not to alter the arrival stream's overflow/underflow characteristics, the evaluation must be conducted at the same time slot boundaries as those used for underflow computation, we make the following modifications. Define $n_{B(n_o)}$ as the number of the last slot occupied by the underflow buffer frames. Let $V(n_{B(n_o)})$ be the number of frames from $B(n_o)$ in its last slot, $n_{B(n_o)}$. Then, the amount of overflow for the first slot of overflow computation is given by

$$\hat{W}(n_{B(n_o)}) = \hat{A}(n_{B(n_o)}) - \hat{V}(n_{B(n_o)}) - \hat{C}(n_{B(n_o)}) \quad (8)$$

where $\hat{V}(n_{B(n_o)})$ is $V(n_{B(n_o)})$ in units of bits.

Finally, calculating for the overall overflow condition we have, for all M',

$$\hat{B}_{max} \geq \hat{B}(n_0) + \max_{M'}\{\hat{W}(n_{B(n_o)}) + \sum_{i=n_{B(n_o)}+1}^{N_{M'}} (\hat{A}(n_i) - \hat{C}(n_i))\}$$

where $\hat{B}(n_0)$ is $B(n_0)$ in units of bits, and $\hat{B}_{max}$ defines the maximum buffer allocation requirement for the video stream.

The primary difference between previous work and our own is that in previous work the assumption is made that no knowledge of the video source characteristics is available *a priori*. For VOD servers, complete a priori information about each video source is available. All compressed file formats which accommodate frame indexing will contain the necessary frame size information, and can be made subject to this strategy.

3 Preliminary Results

A recent experiment [2], a large VOD storage server was employed with up to 768 MBytes of main memory, and RAID storage devices. The storage system could support delivery of up to 86% of maximum theoretical number of concurrent streams, but utilization reached only 30% due primarily to memory space limitations. Our approach requires a negligible amount of buffer memory at the video server because the CBR delivery can be made to match that of the storage device.

Table 1 illustrates the degree of variation that exists between the maximum and average bandwidth allocation requirement for four video sources encoded with an MPEG-1 variable bit rate encoder [3]. The gain in multiplexing here is simply taken as the ratio of the VBR requirement over the average bandwidth requirement.

We present in Table 2 some preliminary results for analyses conducted on the same four videos presented in Table 1 above. The table shows the buffer requirements computed by our algorithm for both the overflow and underflow conditions. Note that the overflow requirements excludes the $B(n_0)$ term in equation 5. The jump in buffer size requirements between the first two movies and the last two is a result of the higher resolution determined by the quality factor.

Table 1: Bit rate req. for CBR strategy vs. VBR delivery

Movies (MPEG-1)	Req. Bit-rate (x10^6 bps)			Multiplexing Gain	
	Peak	0.9xPeak	Ave.	Peak	0.9xPeak
Jurassic	12.386	11.147	1.915	6.46	5.81
Speed	12.781	11.503	2.478	5.16	4.64
MTV	22.385	20.146	2.257	9.92	8.93
NBA	21.099	18.989	4.105	5.14	4.63

Table 2: Buffer req. for CBR delivery strategy

Movies (MPEG-1)	Part. Overflow Req.		Underflow Req.		Total
	Mbytes	Frames	Mbytes	Frames	Mbytes
Jurassic	3.280	315	0.593	91	3.873
Speed	3.391	299	0.159	18	3.549
MTV	4.882	291	1.869	220	6.751
NBA	14.682	462	5.964	301	20.683

4 Conclusion

We have described a transmission strategy for the constant bit rate delivery of VBR encoded continuous media. A theoretical framework was presented for determining buffer requirements at the client end; we have shown that by properly computing the size of such a buffer, CBR delivery can be accomplished in a deterministic, real-time fashion. We have shown that this strategy shows promise in being able to generate significant statistical multiplexing gains. Finally, preliminary results indicate that buffer requirements are acceptable for workstations and for current multimedia PC configurations making it useful for a number of projects such as those currently being undertaken jointly by researchers in the education and multimedia communities. One such project is the Living Textbook [3] project whose objective is to deliver real-time multimedia information on demand for use in classroom instruction; the project will use the NYNET [4] regional commercial ATM network to link several K-12 schools in the New York state area.

References

[1] H. Heeke. "Statistical Multiplexing Gain for Variable Bit Rate Video Codecs in ATM networks". In *4th International Packet Video Workshop*, Tokyo, Japan, March 1991.

[2] J. Hsieh and M. Liu and J.C.L. Liu and H.C. Du and T.M. Ruwart. "Performance of a Mass Storage System for Video-On-Demand". Technical report, University of Minnesota, 1995.

[1] A collaborative effort involving the NYNEX Corporation, Syracuse University, Columbia Teacher's College, and Northeast Parallel Architectures Center.

[2] NYNET is owned by the NYNEX Corporation.

[3] University of California, Berkeley. *Berkeley MPEG-1 Video Coder*, March 1994.

Multimedia Tools and Applications 3, 215–232 (1996)
© 1996 Kluwer Academic Publishers. Manufactured in The Netherlands.

Constant Bit Rate Network Transmission of Variable Bit Rate Continuous Media in Video-On-Demand Servers

JUAN MIGUEL DEL ROSARIO* mrosario@npac.syr.edu, http://www.npac.syr.edu
GEOFFREY FOX† gcf@nova.npac.syr.edu, http://www.npac.syr.edu
Northeast Parallel Architectures Center, 111 College Place, RM 3-201, Syracuse University, Syracuse, NY 13244-4100

Abstract. Multimedia computing is rapidly emerging as the next generation standard for human-computer interaction. One class of multimedia applications that has been gaining much attention is the real-time display of continuous media data such as video and audio, commonly known as Video-On-Demand (VOD) service. Although advances in computer and network technologies have made VOD service feasible, providing guaranteed quality, real-time video delivery still poses many technical challenges. One such challenge involves the transmission of continuous media traffic over high-speed networks.

In this paper, we present an algorithm for determining the minimum buffer requirement for avoiding overflow or underflow at the client video display process, allowing the network scheduler at the VOD server to enforce a constant bit rate delivery of variable bit rate encoded continuous media. This strategy results in reduced congestion and cell loss at the network switch, and in simplified admission control parameters. Initial results indicate that buffer requirements for typical video streams range from 3.7 to 14.6 Megabytes, which is acceptable by today's multimedia PC standards. Further, we show that this approach increases the number of streams that can be multiplexed by a factor of 4.6 to 9.9 times when compared to peak and 90%-of-peak bandwidth allocation strategies.

Keywords: multimedia, video server, buffer, real-time scheduling, video-on-demand

1. Introduction

A significant amount of research effort is currently being directed towards expanding the dimensions of human-computer interaction in an endeavor to bring computers to use in more commonplace aspects of everyday life. In this effort, multimedia computing is a rapidly emerging application area that is being promoted by many as the next generation standard for human-computer interaction. One class of multimedia applications that has been gaining much attention lately is the real-time display of continuous media data such as video and audio. This is commonly known as Video-On-Demand (VOD) service. In a VOD environment, geographically dispersed clients interactively access continuous media streams from a remote server. This type of service is an essential component to providing multimedia educational and entertainment material to the classroom and the home.

*ECE Dept., Syracuse University.
†CIS Dept., Syracuse University.

Combined advances in computer and communication technologies such as secondary mass storage, video compression standards, and high speed networks serve as the foundation for enabling and supporting large-scale VOD service. Although these technological advances have made VOD service feasible, providing guaranteed quality, real-time video delivery still poses many technical challenges. One such challenge involves the transmission of continuous media traffic over high-speed networks.

Fiber-optic and high-speed packet switched technology form the basis of future integrated services networks. The Asynchronous Transfer Mode (ATM) protocol, which is based on fixed sized packets called cells, is evolving as the underlying transport mechanism for such networks. One of the primary uses to which ATM networks will be put to, as envisioned by researchers and telecommunication companies, is the delivery of real-time VOD applications. VOD applications generate extremely delay-sensitive network traffic that, in addition, has high-bandwidth requirements, and may sometimes have stringent loss requirements. ATM networks seem to be the most suitable of the available network technologies to be used for this purpose; however, questions still remain as to whether they are capable of satisfying the strict performance requirements demanded by VOD applications, and to what degree large-scale VOD applications (i.e., very large numbers of video streams) can be supported (multiplexed) by them.

1.1. Constant and variable bit rate encoding

In evaluating network performance, an important consideration is the type of network traffic that is expected. Much research work has been has been devoted to the study of video transmission using ATM networks [20, 8, 9, 27, 18, 30, 29, 19, 17, 31]. The key issue that needs addressing here is whether or not using variable bit rate (VBR) encoding schemes offer a performance advantage over constant bit rate (CBR) encoding schemes.

In CBR encoded video sources, picture quality parameters are adjusted to maintain a constant requirement for delivery. As a result, although the video traffic can be transmitted via a fixed, reserved bandwidth, degradation of picture quality often occurs during encoding. In VBR encoding, the video source is encoded with a constant picture quality. This results in a variable number of bits from frame to frame.

To maximize the efficient use of network bandwidth, it is necessary to multiplex several video sources onto the same transmission channel. It is ATM's ability to provide variable bandwidth dynamically (through statistical multiplexing) that makes it an attractive choice as a variable bit rate transport mechanism. A measure of the effectiveness of VBR transmission schemes has been formulated by Heeke [8] and is called the *statistical multiplexing gain*; it is defined as the ratio of the number of multiplexed VBR sources to the number of multiplexed CBR sources while maintaining an equivalent subjective picture quality.

Multiplexing several VBR sources can be accomplished very simply by reserving the maximum bandwidth required by each source. Obviously, this results in very inefficient use of the network bandwidth. Problems start to arise however when VBR sources are multiplexed without peak bandwidth reservations for each. Under these conditions, the source bandwidth requirements and available network bandwidth fluctuate independently

of each other over time. Such fluctuations result in congestion at the network switches which, in turn, cause cells to be queued, and consequently delayed, and sometimes dropped. Furthermore, because the fluctuations (and thus the degree of congestion) occur randomly over time, the switch delay, aside from causing overall end-to-end delay, also results in increased jitter (variations in end-to-end delay) in the transmitted video stream. Although cell loss is undesirable because it leads directly to picture degradation, it can be tolerated to some extent. However, real-time video sources are extremely delay-sensitive and have very strict bounds on delay tolerance. Jitter leads to choppiness in the output as well as possible loss of synchronization for phase sensitive streams.

1.2. Related work

Finding a solution to the problem of multiplexing VBR signals over a single transmission channel is an active area of research. The problem of resource allocation for broadband networks is addressed by Hui [11], who uses a multilevel congestion evaluation mechanism to determine statistical congestion characteristics. A multilayer bandwidth allocation algorithm is then employed which uses these measures to perform resource allocation.

The problem of VBR traffic packet multiplexing and error control are addressed by Dempsey et al. [7]. Here the problem of congestion, delay and cell loss is dealt with at the level of the network switch. Issues dealing with packet congestion and traffic characteristics, and scheduling algorithms are addressed.

Several researchers have studied the problem of VBR transmission from the level of the scheduler at the video source. Most of these schemes employ some type of predictive method to allow for some sort of "smoothing" of the video source. In [20], an exponential forecast function is used to smooth teleconferencing video streams with the objective of reducing cell loss probabilities. Simulation results show a reduction of cell loss probability average from 2.076×10^{-3} down to 7.8×10^{-7}. It remains to be shown what the equivalent cell loss probabilities will be for full-motion video where scene changes play a significant role in determining traffic characteristics. Initial results presented in [24] indicated that, based upon techniques in [20], the effects of smoothing on intraframe coding (used during scene changes) was almost imperceptible. In [28], Rodriguez-Dagnino et al. present a strategy for predicting bit rate characteristics of encoded video from the uncompressed source. Knightly et al. [12] devise a method for empirically characterizing the video source called the *empirical envelope*. This characterization model is then used along with an earliest-deadline-first packet scheduler to provide deterministic service of VBR video traffic. Their results show an 18% to 38.3% utilization is achievable, but only under ideal conditions. Pancha and El Zarki [23, 21, 22], employ a simple predictive scheme along with prioritized partitioning of the video source to modulate the allocation of network bandwidth to the video source. For trial cases examined (which employed MPEG compressed video sources), a maximum cell loss percentage of between 18% to 49% and an average of between 1% to 3% is observed.

In this paper we describe how selected information from an encoded video source can be used to determine buffer size bounds which enable VBR sources to be deterministically transmitted as CBR streams. We focus here on its application to the network

scheduler at the VOD server. (The subject of disk scheduling and retrieval will be addressed in a subsequent paper.) The rest of the paper is organized as follows. In Section 2, we describe our transmission model and our algorithm for determining buffer requirements. In Section 3 we describe the video source information necessary for CBR transmission. Section 4 describes our scheduling strategy, stream multiplexing, and admission control parameters. Section 5 presents preliminary results and Section 6 concludes.

2. Transmission model

Our model for transmitting a VBR video source as CBR traffic is illustrated in figure 1. As shown, the video source is assumed to have been stored as a VBR stream in secondary storage. An example of the bursty nature of such a stream is shown in figure 2. This video stream is retrieved in segments and placed into a buffer for the network manager.

Consider a video stream being transmitted from the network manager, which we will call the *server*, to a remote display device which we call the *client*. We assume that each video stream is composed of a set of frames of varying sizes, and that it has a finite amount, N_F, of these frames; the size of each frame in bits is denoted by X_i. Such a real-time video stream is naturally subdivided into time *slots*, n_i for $i \in \{0, 1, 2, 3, \ldots, N_s\}$ where N_s is the number of slots in the video stream. Each slot, n_i refers to the time interval $[t_i, t_{i+1}]$ in the video stream transmission period. This representation is illustrated by figure 3.

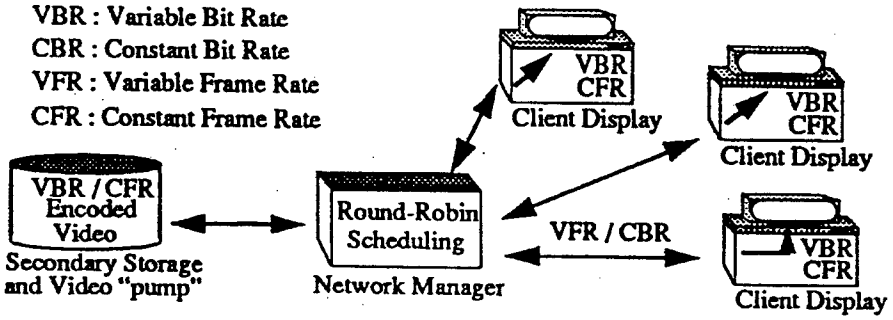


Figure 1. VOD transmission model.

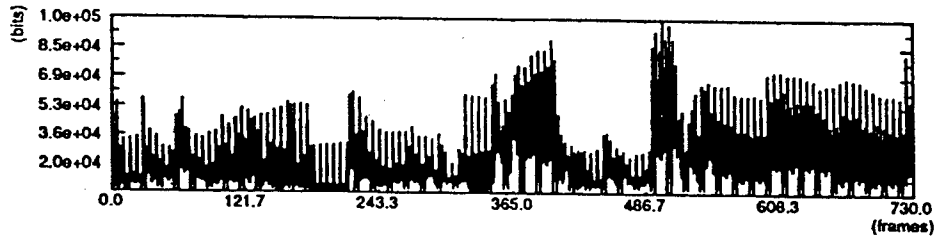


Figure 2. Sample VBR source.

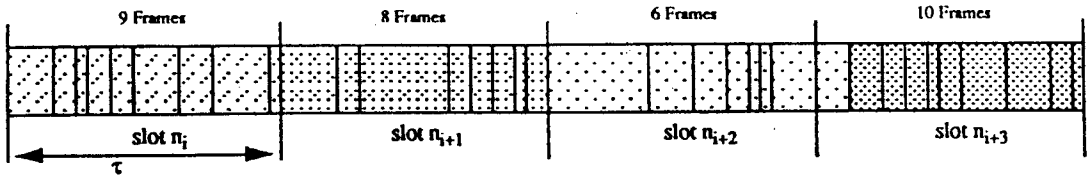


Figure 3. Variable frame rate/constant bit rate.

For any given video stream, s, its total length in bits is given by $L^s = \sum_{i=1}^{N_F^s} X_i$. Assuming a desired playout rate in frames/sec of R_F^s, the playout time in seconds for the entire stream is $T^s = \frac{N_F^s}{R_F^s}$, and the corresponding average bit rate requirement is $R_b^s = \frac{L^s}{T^s}$.

Let T_c represent the *cycle* time in seconds for the server scheduler. We define the cycle time as the time it takes for the scheduler to service every supported stream once. The server employs a round-robin schedule to transmit the video stream. During each cycle of the scheduler, a pre-determined time slot is allocated to each stream. Due to the real-time and continuous nature of a video stream, at each cycle the scheduler must deliver a sufficient amount of video data to satisfy consumption for the entire cycle, T_c; that is, it must deliver $B_c^s = R_b^s \times T_c$ bits of data per cycle. Suppose that the scheduler allocates to each stream some portion, t_c^s, of T_c. In order to satisfy the client's consumption requirement per cycle, the server must deliver B_c^s bits of data in time t_c^s. Hence, for each stream, the bandwidth requirement for delivering the video stream is equal to $R_{b_c}^s = \frac{B_c^s}{t_c^s} = R_b^s \frac{T_c}{t_c^s}$, a constant bit rate. At the receiving end (the client), the video stream is buffered and displayed at the appropriate constant frame rate.

Note that the delivery strategy we propose is dependent upon the assumptions we make about the scheduling algorithm. Our objective is to make $R_{b_c}^s$ constant in order to achieve maximum network transmission efficiency. In this paper, we assume the simplest possible scheduling algorithm; we make the assumption that the time, t_c^s, allocated to each stream by the server is fixed for each stream. In this case, t_c^s can be a fixed constant for all streams (in which case $R_{b_c}^s$ will vary for each), or it can be made proportional to R_b^s (in which case the value of $R_{b_c}^s$ will be independent of the stream). Future papers will investigate the use of more complex scheduling algorithms which will allow t_c^s to be dynamically adjusted resulting in a reduction in buffer size requirements.

2.1. *Buffer allocation requirements*

The ability to transmit a VBR source as CBR streams depends upon buffering at the client end. In this section, we quantify the buffer requirement for a given video stream.

Consider the client process. Let $C(n_i)$ be the number of frames consumed by the client in slot n_i; this value is a constant and is equivalent to the playout frame rate (e.g., 30 frames/sec for full motion video). Let $B(n_i)$ be the number of frames in the buffer at the beginning of slot n_i (i.e., at time t_i), and let $A(n_i)$ be the number of frames arriving during slot n_i. Then, for each time slot, the number of frames remaining in the buffer is given by the

recursive equation

$$B(n_{i+1}) = B(n_i) + A(n_i) - C(n_i), \qquad \forall i \in \{0, 1, 2, \ldots, N_s\}$$

Consider the frame arrival rate, $A(n_i)$, at the client process. Since we transmit the video stream at a constant bit rate and the size of each frame, X_i, may vary, there is a variable frame arrival rate (i.e., $A(n_i)$ is not constant). This is illustrated in figure 3. Note that in each slot we count only the number of complete frames within the slot; thus from the figure, the last frame in the third slot is counted towards the fourth slot. Since the number of incoming frames for any slot may be less than that necessary for consumption, we cannot rely on incoming frames to satisfy the current slot consumption requirements. Therefore, we have to satisfy the current display requirements solely from the buffered frames, display will be interrupted if the buffer is ever empty for any of the time slots. Hence, we have the requirement

$$B(n_{i+1}) = B(n_i) + A(n_i) - C(n_i) \geq 0 \qquad \forall i \in \{0, 1, 2, \ldots, N_s\}$$

Since this is true for all slots n_i, without loss of generality it is true for some slot n_m. Then, the recursive formula above for the buffer requirement can be written as

$$B(n_m) = B(n_0) + \sum_{i=0}^{m-1} A(n_i) - \sum_{i=0}^{m-1} C(n_i) \geq 0 \qquad \forall m \in \{0, 1, 2, \ldots, N_s\} \qquad (1)$$

Thus, for any point throughout the transmission of the video stream (from beginning to end), the client buffer must never underflow (i.e., it must never be less than zero).

2.2. *Flow conditions*

For a given time slot, without considering the amount buffered, if the number of frames that arrive in that slot is less than the amount consumed (i.e., $A(n_i) < C(n_i)$) we call this slot an *underflow* slot; if it is greater than the amount consumed (i.e., $A(n_i) > C(n_i)$), we call it an *overflow* slot; if it is equal to the amount consumed (i.e., $A(n_i) = C(n_i)$), we call it an *even* slot. We make the assumption that data for a given slot will arrive at the client and be buffered prior to the start of its consumption.

During the playout of a video stream, there will be a combination of overflow, underflow and even slots. Thus we have

$$\sum_{i=0}^{M} A(n_i) = \sum_{i \in M_u} A(n_i) + \sum_{i \in M_o} A(n_i) + \sum_{i \in M_e} A(n_i) \qquad (2)$$

where M_u, M_o, and M_e are the set of underflow, overflow, and even slots respectively, and $|M_u| + |M_o| + |M_e| = M$, where $M \in \{0, 1, 2, \ldots, N_s\}$.

From Eqs. (1) and (2) we have, $\forall M \in \{0, 1, 2, \ldots, N_s\}$

$$B(n_0) + \sum_{i \in M_u} A(n_i) + \sum_{i \in M_o} A(n_i) + \sum_{i \in M_e} A(n_i) - \sum_{i=0}^{M} C(n_i) \geq 0$$

$$\iff B(n_0) + \sum_{i \in M_u} A(n_i) + \sum_{i \in M_o} A(n_i) + \sum_{i \in M_e} A(n_i) - \sum_{i \in M_u \cup M_o} C(n_i) - \sum_{i \in M_e} C(n_i) \geq 0$$

$$\iff B(n_0) + \sum_{i \in M_u} A(n_i) + \sum_{i \in M_o} A(n_i) - \sum_{i \in M_u \cup M_o} C(n_i) \geq 0 \tag{3}$$

where, by definition, $A(n_i) < C(n_i) \quad \forall j \in M_u$ and $A(n_i) \geq C(n_i) \forall j \in M_o$.

There are now two conditions which must be accounted for in buffering. The first is the *buffer underflow* condition which describes the situation where, during some point in the video transmission, the buffer runs empty and there is nothing for the client to display. The second condition is the *buffer overflow* condition which describes the situation where the buffer is full and can not accommodate the incoming frames for the current slot. We address these two cases below.

2.2.1. Buffer underflow condition. Consider the case where the video stream is transmitted to the client without prefetching a few frames into the client buffer. That is, at the start of display at time t_0, the buffer is empty (i.e., $B(n_0) = 0$). Then, the buffer underflow condition comes about when for some time slot during the transmission of the video, the total number of frames that have arrived is less than that required for display (consumed). Thus we have, for some time slot, N_M, where $M = |M_u| + |M_o| + |M_e|$

$$\sum_{i \in M_u} A(n_i) + \sum_{i \in M_o} A(n_i) < \sum_{i \in M_u \cup M_o} C(n_i)$$

$$\iff \sum_{i \in M_u} A(n_i) + \sum_{i \in M_o} A(n_i) - \sum_{i \in M_u \cup M_o} C(n_i) < 0$$

Therefore for Eq. (3) to hold, we require

$$\forall M, \quad B(n_0) \geq \left| \sum_{i \in M_u} A(n_i) + \sum_{i \in M_o} A(n_i) - \sum_{i \in M_u \cup M_o} C(n_i) \right|$$

$$\geq \left| \sum_{i \in M_u} A(n_i) - \sum_{i \in M_u} C(n_i) + \sum_{i \in M_o} A(n_i) - \sum_{i \in M_o} C(n_i) \right| \tag{4}$$

$$\geq \left| \sum_{i=0}^{N_M} (A(n_i) - C(n_i)) \right| \tag{5}$$

where Eq. (5) follows from (4) because $\sum_{i \in M_e} A(n_i) = \sum_{i \in M_e} C(n_i)$ by definition. Hence,

to prevent the buffer underflow condition, we need to transmit a number of frames into the client buffer before initiating the display. Since Eq. (5) must be true for all M,

$$B(n_0) = \max_M \left(\left| \sum_{i=0}^{N_M} (A(n_i) - C(n_i)) \right| \right) \tag{6}$$

gives the required number of frames to buffer prior to display in order to avoid buffer underflow. Note that we compute the buffer size here in units of frames because consumption is in terms of frames (i.e., VBR/CFR) independent of individual frame size.

To facilitate the discussion above, assumptions were made about the nature of the frame sequence with regards to generating underflow within a segment. In a true implementation, the video stream will comprise a combination of overflow and underflow slots for each segment, N_M, of the stream. Therefore, Eq. (6) is modified as follows. We let F be defined, for all M, as

$$F = \min_M \left(\sum_{i=0}^{N_M} (A(n_i) - C(n_i)) \right)$$

where the difference term, $A(n_i) - C(n_i)$, above is derived from the second and third terms of Eq. (1) which becomes negative for underflows. Thus, the maximum underflow condition will be represented by the most negative value of F. We can now define the underflow buffer requirement as

$$B(n_0) = \begin{cases} -F & \text{if } F < 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

2.2.2. Buffer overflow condition. The buffer overflow condition is similar to the underflow case but the initial condition is now slightly different. The buffer overflow condition comes about when for some time slot during the transmission of the video, the difference between the total number of frames that have arrived and those consumed is greater than the amount containable in the buffer. Unlike the underflow condition, here we consider buffer requirements in terms of bits rather than frames because the arrival of data is in bits (CBR/VFR) and we want to buffer against over-arrivals. To do this we modify our definitions of M_u, M_o, and M_e into M'_u, M'_o and M'_e where M'_u is the set of slots for which the number of bits arriving is less than the number of bits consumed; M'_o and M'_e follow analogously. To minimize the overflow buffer requirement, we assume the initial underflow buffer frames to exist (i.e., $B(n_0)$ is given by Eq. (6)). Thus we have, for some time slot, $N_{M'}$, where $M' = |M'_u| + |M'_o| + |M'_e|$

$$\hat{B}(n_0) + \sum_{i \in M'_u} \hat{A}(n_i) + \sum_{i \in M'_o} \hat{A}(n_i) - \sum_{i \in M'_u \cup M'_o} \hat{C}(n_i) \geq 0 \tag{8}$$

which is equivalent to Eq. (3) above with the difference that $\hat{A}(n_i)$, $\hat{B}(n_i)$, and $\hat{C}(n_i)$ are $A(n_i)$, $B(n_i)$, and $C(n_i)$ respectively, in units of bits rather than frames. (Note that $\hat{C}(n_i)$

represents a variable bit consumption rate.) Similarly, it follows that

$$\sum_{i\in M'_u} \hat{A}(n_i) + \sum_{i\in M'_o} \hat{A}(n_i) > \sum_{i\in M'_u \cup M'_o} \hat{C}(n_i)$$
$$\iff \sum_{i\in M'_u} \hat{A}(n_i) + \sum_{i\in M'_o} \hat{A}(n_i) - \sum_{i\in M'_u \cup M'_o} \hat{C}(n_i) > 0$$

Therefore Eq. (8) holds. But, to prevent buffer overflow we require that

$$\forall M' \quad \hat{B}_{\max} \geq \hat{B}(n_0) + \sum_{i\in M'_u} \hat{A}(n_i) + \sum_{i\in M'_o} \hat{A}(n_i) - \sum_{i\in M'_u \cup M'_o} \hat{C}(n_i)$$
$$\geq \hat{B}(n_0) + \sum_{i\in M'_u} \hat{A}(n_i) - \sum_{i\in M'_u} \hat{C}(n_i) + \sum_{i\in M'_o} \hat{A}(n_i) - \sum_{i\in M'_o} \hat{C}(n_i) \tag{9}$$
$$\geq \hat{B}(n_0) + \sum_{i=0}^{N_{M'}} (\hat{A}(n_i) - \hat{C}(n_i)) \tag{10}$$

again Eq. (10) follows from Eq. (9) because $\sum_{i\in M'_e} \hat{A}(n_i) = \sum_{i\in M'_e} \hat{C}(n_i)$ by definition. Since Eq. (10) is must be true for all M',

$$\hat{B}_{\max} \geq \hat{B}(n_0) + \max_{M'} \left(\sum_{i=0}^{N_{M'}} (\hat{A}(n_i) - \hat{C}(n_i)) \right) \tag{11}$$

gives the required buffer size to avoid overflow.

Note that in Eq. (11) we begin computing the overflow buffer at $i = 0$. In order to minimize buffer allocation, an additional bit of computation is required. Since it is important not to alter the arrival stream's overflow/underflow characteristics, the evaluation process for overflow buffer requirement must be conducted at time slot boundaries. This guarantees that the associated overflow computations hold by assuring that they begin at the same time slot boundaries as those used for underflow computation. Therefore, instead of beginning at the first time slot (i.e., $i = 0$), we first round the number of frames given by Eq. (7) to the nearest time slot worth of frames, then, we begin overflow evaluation at the start of that time slot. Further, we subtract from the computation for the starting time slot the number of frames that are to be included in the underflow buffer. We describe this algorithm more formally in the following.

Define $n_{B(n_0)}$ as the number of the last slot occupied by the underflow buffer frames. Let $V(n_{B(n_0)})$ be the number of frames from $B(n_0)$ in its last slot, $n_{B(n_0)}$. Then, the amount of overflow for the first slot of overflow computation is given by

$$\hat{W}(n_{B(n_0)}) = \hat{A}(n_{B(n_0)}) - \hat{V}(n_{B(n_0)}) - \hat{C}(n_{B(n_0)})$$

where $\hat{V}(n_{B(n_0)})$ is $V(n_{B(n_0)})$ in units of bits. The general idea of the formulation is illustrated in figure 4.

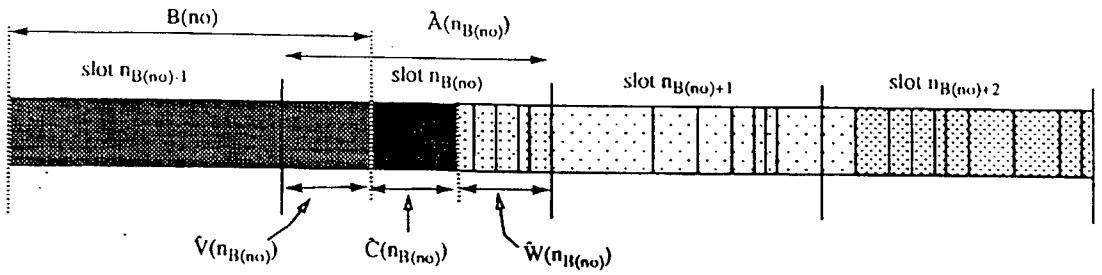


Figure 4. Overflow buffer computation—first slot.

Finally, calculating for the overall overflow condition we have, for all M',

$$\hat{B}_{\max} \geq \hat{B}(n_0) + \max_{M'} \left\{ \hat{W}\left(n_{B(n_0)}\right) + \sum_{i=n_{B(n_0)}+1}^{N_{M'}} (\hat{A}(n_i) - \hat{C}(n_i)) \right\} \tag{12}$$

where $\hat{B}(n_0)$ is $B(n_0)$ in units of bits, and $\hat{B}_{\max}$ defines the maximum buffer allocation requirement for the video stream.

3. Required video source information

The primary difference between previous work and our own is that in previous work (with the exception of [12] and [20]) the assumption is made that no knowledge of the video source characteristics is available *a priori*. In the case of Ott et al. [20], the information that is assumed (the size of each frame and the amount arriving per time period) is used only to clarify the exposition. In their model, a true implementation would need to forecast these values. In the case of Knightly et al. [12], video trace information (i.e., exact frame sizes and their arrival times) is assumed to be present. This information is used to create a traffic characterization "envelope" producing the results mentioned in Section 1.2.

For VOD servers, complete a priori information about each video source is available. In our transmission model, we assume that the video sources have been encoded via MPEG [34] or JPEG [35] standard compression algorithms. However, the algorithm truly only requires that the encoded output possess some frame header information (or its equivalent) which permits the extraction of frame size data. All compressed file formats which accommodate frame indexing during playback will contain such information, and can be made subject to this strategy. Obtaining the information necessary to implement our strategy is surprisingly simple. By collecting a frame size trace of the given video source, we can completely characterize the source for purposes of scheduling and transmission. In an implementation, this can very simply be done (off-line) as the video source is placed into secondary or tertiary storage. For JPEG, and MPEG compressed sources this is computationally inexpensive and can be accomplished by reading the frame headers (without decoding the frame images). Then, by examining Eqs. (7) and (12) we determine the exact buffering requirements.

4. Scheduling algorithm

Once proper information is available, deterministic CBR scheduling is uncomplicated. When a request is initiated by a client, and the admission control checks (discussed below) have been passed, the scheduler begins to transmit the video source at a constant bit rate until the amount buffered, $B(n_0)$, is equal to the maximum underflow state value given by Eq. (6). At this point, the client process can begin real-time display of the video stream. Since we have buffered in anticipation of the maximum underflow state, underflow can no longer occur. Likewise, for the overflow case, after allocating the buffer space required by Eq. (12), we are guaranteed that overflow situations will not occur, so CBR delivery can continue without frame loss after the initial buffering has been accomplished.

4.1. Stream multiplexing

Consider the case where multiple streams are being delivered. For an additional stream to be scheduled, the algorithm in figure 5 is used. We assume that the time slot duration, t_c^s, is fixed. The number of streams that can be admitted is dependent upon the overall available bandwidth and the current amount of slack time available.

The bandwidth requirement for a VBR stream transmitted at a constant bit rate equivalent to the streams average bit rate requirement is considerably less than allocations for peak or close-to-peak bandwidth. Therefore, considerable gains can be made in multiplexing. This will be discussed further in Section 5 below.

4.2. Admission control parameters

The admission control parameters are greatly simplified by our CBR transmission strategy. The following is a list of the primary admission control parameters required.

1. Total remaining bandwidth (or slack time).
2. Required playback rate (frames/second).
3. Required per-cycle network transmission rate for the current stream.
4. Client buffer size requirement for the current stream.

---

1. Compute the required video stream bit rate: R_b^s.
2. Compute the required scheduler per-cycle network transmission rate.
   Given by: $R_{b_c}^s = R_b^s \frac{T_c}{t_c^s}$
3. Admission control check (includes bandwidth requirements).
4. If admission control passed, transmit video stream.

Figure 5. Stream Multiplexing Algorithm.

5. Preliminary results and discussion

The CBR transmission strategy we have proposed was motivated by a number of problems associated with existing approaches to VBR scheduling and transmission. Among these is the problem of cell loss and delay at the network switches caused by variable bandwidth utilization profiles. Transmitting video sources in CBR fashion should greatly reduce the degree of switch congestion (and consequently its negative effects) by simplifying admission control and policing requirements. This section presents some preliminary results and discusses implications for some of the other significant problems that have motivated our development of this strategy.

5.1. Server memory requirement

Another major concern is the memory requirement at the video server. In a recent experiment [10], a large VOD storage server was designed which contained up to 768 MBytes of main memory, and RAID storage devices. Although the storage system could support delivery of up to 86% of maximum theoretical number of concurrent streams, the reported number of concurrent streams supported reached only 30% of the theoretical maximum due primarily to memory space limitations. Our approach reduces the amount of buffer memory necessary at the video server. In a VBR delivery scheme, at every cycle of the scheduler, a constant number of frames (variable number of bits) must be delivered. Reduced memory requirement is made possible by our strategy because constant bit rate delivery precludes any need to anticipate (through excess buffer allocation) time slots with extremely large frames and thus increases the limit on supportable concurrent video streams.

5.2. Multiplexing gain

One of the principal advantages of the CBR strategy is the statistical multiplexing gain that is achieved. In one experiment, we analyzed a 10,000 frame MPEG-1 VBR encoded video stream. The average bit rate requirement from this stream was 1.92 Mbps. For the same quality parameters, a CBR encoding would require an average bit rate of 12.38 Mbps. Thus, using our strategy, we are able to multiplex up to 6 more video streams compared with the usual CBR transmission of the CBR encoded source.

Thus, the bandwidth requirement for a VBR stream transmitted at a constant bit rate which is equivalent to the stream's average bit rate requirement is considerably less than allocations for peak or close-to-peak bandwidth. Table 1 illustrates the degree of variation that exists between the maximum and average bandwidth allocation requirement for four video sources encoded with an MPEG-1 variable bit rate encoder [33] (the encoding procedure employed is described further below). The entry for "Average" bandwidth represents the bandwidth requirement for the CBR strategy where the frame average has been selected as the constant bit rate for transmission. The two rightmost column of values show the gain in multiplexing obtained from using the average bandwidth versus the peak and 90% of peak bandwidth requirements respectively. The gain in multiplexing here is simply taken as the ratio of the VBR requirement over the average bandwidth requirement.

Table 1. Bandwidth requirement for proposed CBR strategy vs. VBR delivery.

Movies (MPEG-1)	Req. bandwidth ($\times 10^6$ bits/sec)			Multiplexing gain vs.	
	Peak	90% of peak	Average	Peak	90% of peak
Jurassic park	12.386	11.147	1.915	6.46	5.81
Speed	12.781	11.503	2.478	5.16	4.64
MTV music videos	22.385	20.146	2.257	9.92	8.93
NBA basketball	21.099	18.989	4.105	5.14	4.63

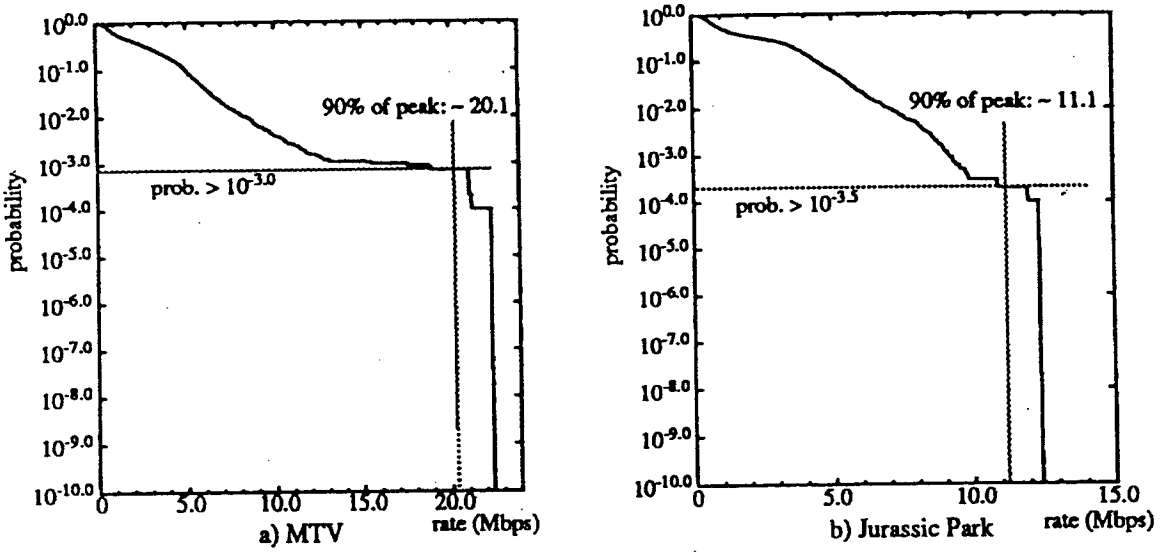


Figure 6. Negative pdf's for VBR encoding of Jurassic Park and MTV samples.

It is interesting to note the effects of transmitting at 90% of peak. Figure 6 shows the negative cumulative distribution functions for both MTV and Jurassic Park for required transmission bit-rates. The ordinate axis represents the probability that a given bit rate is exceeded and is proportional to the cell loss probability for the multiplexed signal (i.e., $\bar{F}(x) = \Pr\{X \geq x\}$ where $F(x)$ is the cdf of the signal). The figures show that transmitting at 90% of peak will result in a probability between $10^{-3.0}$ and $10^{-4.0}$ and that a greater bit rate will be required. If the system is unable to support this need for higher bit rates, unacceptable data loss will occur. Thus, this situation results in extremely poor reliability when measured against typical transmission reliability requirements of $\bar{F}(x) \leq 10^{-8.0}$. On the other hand a constant bit rate transmission, as is suggested in this paper, will appear on the graph as a vertical line at the point on the x-axis equivalent to the constant transmission rate.

5.3. *Client memory requirement*

The primary drawback of our CBR approach is that it requires the client to satisfy the buffer memory requirement as an admission control parameter. Further, the memory requirement

Table 2. Buffer requirements for CBR delivery strategy.

Movies (MPEG-1)	Partial overflow Req.		Underflow Req.		Total Megabytes
	MBytes	Frames	MBytes	Frames	
Jurassic park	3.112	315	0.593	91	3.705
Speed	3.555	299	0.159	18	3.714
MTV music videos	3.700	291	1.869	220	5.569
NBA basketball	8.686	462	5.964	301	14.650

is a function of the video source and may greatly vary. However, there are a number of mitigating factors that offset this drawback. First, alternative schemes likewise impose memory requirements, except that these requirements are made on the video server. Such memory requirements become more demanding of the system when we consider that the video server needs to allocate one buffer per video stream that it concurrently supports. Second, multiplexing congestion and cell loss are likely to be considerably diminished by our approach. Third, we view memory devices as commodity items whose price is only expected continue to decrease, and whose quantity we expect will greatly increase in typical work stations or multimedia display devices; on the other hand, network bandwidth is a very expensive and scarce resource.

To give some sense of what the magnitudes are like for the buffer requirements imposed by this algorithm, we present in Table 2 some preliminary results for analyses conducted on the same four videos presented in Table 1 above. The table shows the buffer requirements computed by our algorithm for both the overflow and underflow conditions. Note that the overflow requirements listed exclude the amount buffered for underflow (i.e., it excludes the $B(n_0)$ term in Eq. (7)).

The video sources comprise a 10,000 frame (approx. 15 mins.) segment of the full movies and were captured using a MultiVideo[1] card and were later compressed to MPEG-1. Jurassic Park and Speed were captured with a MultiVideo Q-factor of 105, and the MTV and Basketball videos had a Q-factor of 50 (the higher the Q-factor, the lower the resolution)[2].

MPEG compression was performed at the same quality factors for all the videos. The jump in buffer size requirements between the first two movies and the last two is a result of the higher resolution determined by the Q-factor. The basketball film showed the largest buffer requirement due to its many scene changes.

From the results in Table 2, buffer size requirements range from 3.705 to 14.650 Megabytes. For multimedia PC's, currently being marketed with 64, 256 and even over 512 Megabytes of RAM, this buffer requirement has a corresponding memory consumption in the range 5.7% to 22.9%, 1.4% to 5.7%, and 0.65% to 2.9% respectively.

In order to reduce the buffer size requirement further, transmission rates other than the average bit rate can be used. In the following table, Table 3, we show the buffer sizes obtained using the optimal bit rates for the sample video streams. The optimal bit rates were found by varying the average bit rate by plus or minus some fraction of the standard deviation, referred to as "std. dev. factor" in the table (i.e., optimal bit rate = average rate + (average rate × std. dev. factor)). Note that all buffer size figures are in units of MegaBytes.

Table 3. Buffer requirements using optimal transfer rates.

	Ave. bit rate		Optimal bit rate		
Movies (MPEG-1)	Underflow req. (MBytes)	Total (B_{max})	Std. dev. factor	Underflow req. (MBytes)	Total (B_{max})
Jurassic park	0.593	3.705	+0.23	0.194	3.295
Speed	0.159	3.714	−0.07	0.272	3.650
MTV music videos	1.869	5.569	−0.81	2.485	3.665
NBA basketball	5.964	14.650	+0.93	1.579	10.084

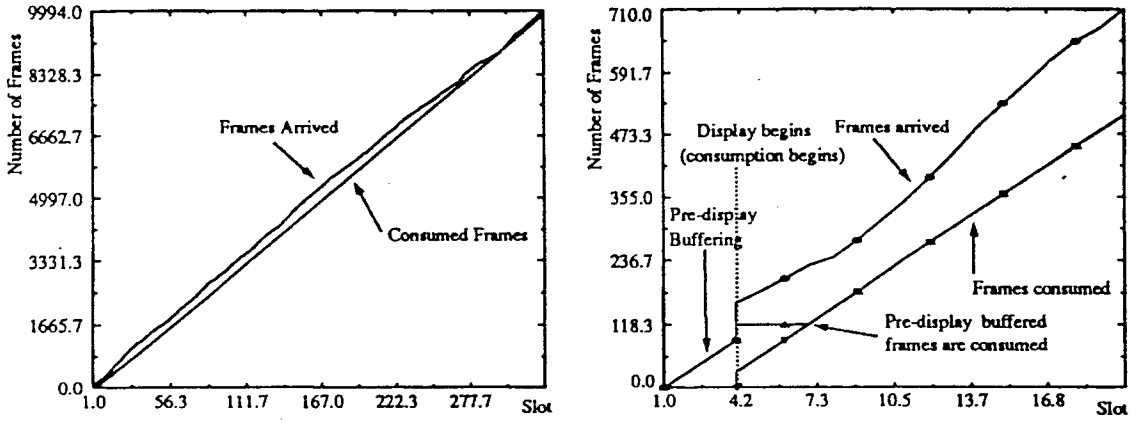


Figure 7. Buffer usage profile.

In figure 7 we show on the left the buffer utilization profile for the Jurassic Park sample. Although the buffer may come very close to being empty at some points (e.g., around time slot 300) it never completely does so. The graph on the right is a macro view of the first 20 frames; it shows that the amount buffered prior to display is quickly used up but that it provides sufficient delay to allow arriving frames to develop a wide enough gap between the consumption profile and the arrival profile.

Initial tests on a one hour sample of an MTV video resulted in a buffer requirement of about 8 MBytes. Although this provides some indication that the approach extends to longer video streams, the fact that the one hour MTV segment requires more buffer than the 15 minute segment simply highlights the dependence of this approach on the profile of a given stream.

6. Conclusion and future work

We have described a transmission strategy for the constant bit rate delivery of VBR encoded continuous media. A theoretical framework was presented for determining buffer requirements at the client end; we have shown that by properly computing the size of such a buffer, CBR delivery can be accomplished in a deterministic, real-time fashion. We described how our strategy can potentially reduce congestion and cell loss at the network switch, and how it greatly simplifies admission control. We have shown that this strategy shows promise in

being able to generate significant statistical multiplexing gains. Finally, preliminary results indicate that buffer requirements are acceptable for workstations and for current multimedia PC configurations making it useful for a number of projects such as those currently being undertaken jointly by researchers in the education and multimedia communities. One such project is the Living Textbook[3] project whose objective is to deliver real-time multimedia information on demand for use in classroom instruction; the project will use the NYNET[4] regional commercial ATM network to link several K-12 schools in the New York state area.

Our plan for future work in this area includes: exploration into the use of more complex scheduling algorithms and alternative delivery strategies; a detailed characterization of a more comprehensive set of video source samples of varying compression parameters to obtain more statistically accurate bounds on buffer size requirements [5]; exploration of scheduling schemes for cases with bounded buffers [6]; detailed analyses on congestion effects arising as a result of using this strategy versus other existing approaches.

Acknowledgment

This work was sponsored in part by the US Airforce under Rome Laboratory contract # F30602-94-C-0256. A condensed version of this paper was presented at the Second IASTED/ISMM International Conference on Distributed Multimedia Systems and Applications.

We would like to thank Mahesh Subramanyan and Marek Podgorny for many interesting discussions, and for their helpful comments and suggestions.

Notes

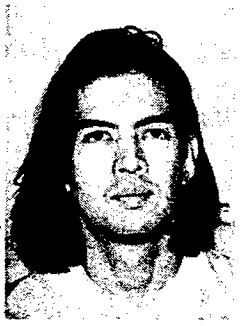
1. MultiVideo is a trademark of Parallax Graphics, Inc.
2. The manufacturer recommended Q-factor for general use is 150.
3. A collaborative effort involving the NYNEX Corporation, Syracuse University, Columbia Teacher's College, and Northeast Parallel Architectures Center.
4. NYNET is owned by the NYNEX Corporation.

References

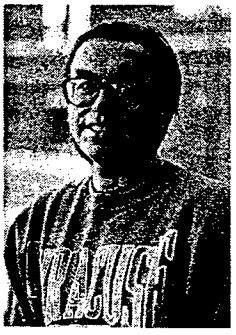
1. C. Blondia and O. Casals, "Performance Analysis of Statistical Multiplexing of VBR Sources," IEEE INFOCOM '92, pp. 828–838, 1992.
2. A. Burchard, J. Liebeherr, Y. Oh, and S.H. Son, Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems, University of Virginia, No. CS-94-01, January 1994.
3. B. DeCleene, P. Pancha, M. El Zarki, and H. Sorensen, "Comparison of Priority Partition Methods for VBR MPEG," IEEE Infocom 94, Toronto, Canada, May 1994.
4. J.M. del Rosario and G. Fox, "Constant Bit Rate Network Transmission of Variable Bit Rate Continuous Media in Video-On-Demand Servers," No. SCCS-677, Northeast Parallel Architectures Center (NPAC), Syracuse University, December 1994, Submitted to: Second IASTED/ISMM International Conference on Distributed Multimedia Systems and Applications.
5. J.M. del Rosario, M. Subramanyan, and G. Fox, "Spectral Analysis of MPEG-1 and MPEG-2 Variable Bit Rate Encoded Video Streams with Implications for ATM Network Transmission," Northeast Parallel Architectures Center (NPAC), Syracuse University, December 1994 (in preparation).

6. J.M. del Rosario, M. Podgorny, and G. Fox, "Scheduling Transmission of VBR Encoded Continuous Media in Bounded Buffer Environments (in preparation)," Northeast Parallel Architectures Center at Syracuse University, 1995.
7. B.J. Dempsey, J. Liebeherr, and A.C. Weaver, "A Delay-Sensitive Error Control Scheme for Continuous Media Communications," University of Virginia, No. CS-93-45, October 1993.
8. H. Heeke, "Statistical Multiplexing Gain for Variable Bit Rate Video Codecs in ATM networks," 4th International Packet Video Workshop, Tokyo, Japan, March 1991.
9. D. Heyman, A. Tabatabai, and T.V. Lakshman, "Statistical Analysis and Simulation Study of Video Teleconference Traffic in ATM Networks," IEEE Global Telecom. Conference, December 1991.
10. J. Hsieh, M. Liu, J.C.L. Liu, H.C. Du, and T.M. Ruwart, "Performance of a Mass Storage System for Video-On-Demand," University of Minnesota, Submitted to: Journal of Parallel and Distributed Computing—Special Issue on Multimedia Processing and Technology, 1995.
11. J.Y. Hui, "Resourse Allocation for Broadband Networks," IEEE Journal on Selected Areas in Communications, Vol. 6, No. 9, pp. 1598–1608, December 1988.
12. E.W. Knightly, D.E. Wrege, J. Liebeherr, and H. Zhang, "Fundamental Limits and Tradeoffs of Providing Deterministic Guarantees to VBR Video Traffic," University of Virginia, temp, 1994.
13. S.S. Lam, S. Chow and D.K.Y. Yau, "An Algorithm for Lossless Smoothing of MPEG Video," ACM SIGCOMM '94, August 1994.
14. D. LeGall, "MPEG: A Video Compression Standard for Multimedia Applications," Communications of the ACM, Vol. 34, No. 4, pp. 305–313, April 1991.
15. J. Liebeherr and D.E. Wrege, "Design and Analysis of a High-Performance Packet Multiplexer for Multiservice Networks with Delay Guarantees," University of Virginia, No. CS-94-30, July 1994,
16. J. Liebeherr, D.E. Wrege, and D. Ferrari, "Exact Admission Control for Networks with Bounded Delay Services," University of Virginia, No. CS-94-29, July 1994.
17. B. Maglaris, D. Anastassiou, P. Sen, G. Karlsson, and J.D. Roberts, "Performance Models of Statistical Multiplexing in Packet Video Communications," IEEE Trans. Commun., Vol. Com-36, No. 7, pp. 834–843, July 1988.
18. D.G. Morrison, "Variable Bit-Rate Video Coding for Asynchronous Transfer Mode Networks," British Telecom. Tech. Journal, Vol. 8, No. 3, pp. 70–80, July 1990.
19. M. Nomura, T. Fujii, and N. Ohta, "Basic Characteristics of Variable Bit Rate Video Coding in ATM Environment," IEEE J. on Select. Areas Commun., Vol. SAC-7, pp. 752–760, June 1989.
20. T. Ott, T.V. Lakshman and A. Tabatabai, "A Scheme for Smoothing Delay-Sensitive Traffic Offered to ATM Networks," IEEE INFOCOM '92, pp. 776–785, 1992.
21. P. Pancha and M. El Zarki, "Modeling of Video Sources for BISDN," Third International Packet Video Workshop, Morristown, NJ, March 1990,
22. P. Pancha and M. El Zarki, "Bandwidth Allocation Schemes for Variable Bit Rate MPEG Sources in ATM Networks," University of Pennsylvania, 1992.
23. P. Pancha and M. El Zarki, "A Look at the MPEG Video Coding Standard for Variable Bit Rate Video Transmission," IEEE INFOCOM '92, Florence, Italy, May 1992.
24. P. Pancha and M. El Zarki, "Prioritized Transmission of VBR Video Traffic," IEEE GLOBECOM '92, Orlando, Florida, pp. 1135–1139, December 1992.
25. P. Pancha and M. El Zarki, "Bandwidth Requirements of VBR Video Coders in ATM based BISDN," IEEE Transactions on Circuits and Systems for Video Technology, June 1993, Vol. 3, No. 3, pp. 190–198, Appeared in part in Proceedings of IEEE Infocom 93, San Francisco, CA, March 1993.
26. P. Pancha and M. El Zarki, "A Study of MPEG Coding for Variable Bit Rate Video Transmission," IEEE Communications Magazine, May 1994.
27. G. Ramamurthy and B. Sengupta, "Modeling and Analysis of a Variable Bit Rate Video Multiplexer," 7th Specialist Seminar, International Teletraffic Congress, Morristown, New Jersey, October 1990.
28. R.M. Rodriguez-Dagnino, M.R.K Khansari, and A. Leon-Garcia, Prediction of Bit Rate Sequences of Encoded Video Signals, IEEE Journal on Selected Areas in Communications, 1991.
29. P. Sen, B. Maglaris, N. Rikli, and D. Anastassiou, "Models for Packet Switching of Variable Bit-Rate Video Sources," IEEE J. on Select. Areas Commun., Vol. SAC-7, pp. 865–869, June 1989.

30. H. Shino et al., "Analysis of Multiplexing Characteristics of Variable Bit Rate Video Signals," Third International Workshop on Packet Video, Morristown, New Jersey, March 1990,
31. W. Verbiest, L. Pinnoo, and B. Voeten, "The Impact of the ATM Concept on Video Coding," IEEE J. on Select. Areas Commun., Vol. SAC-6, No. 9, December 1988.
32. G.K. Wallace, "The JPEG Still-Picture Compression Standard," Communications of the ACM, Vol. 34, No. 4, April 1991.
33. Berkeley MPEG-1 Video Coder, University of California, Berkeley, March 1994.
34. Committee Draft of Standard ISO11172: ISO/MPEG 90/176, *Coding of Moving Pictures and Associated Audio*, December 1990.
35. ISO 10918, JPEG Digital Compression and Conding of Continuous-Tone Still Images, Draft, 1991.



Juan Miguel del Rosario is a Ph.D. candidate with the department of Electrical and Computer Engineering at Syracuse Univerisity, and is a research assistant at the Northeast Parallel Architectes Center (NPAC). He earned an M.S. degree in Computer Science in 1992 and a B.S. degree in Mathematics, Physics, and Chemistry in 1988, both from the University of San Francisco. Prior to attending Syracuse Univeristy, he worked at nCUBE where he played a key role in designing and was principal implementor of one of the first parallel file system's for distributed multiprocessors, The "Video On-Demand Technologies" group at NPAC, of which he is a member, concentrates on the development of technologies necessary for the construction of large scale, parallel multimedia servers for use in education. His research interests include: image compression and related transmission and detection problems; high-speed communications; and parallel I/O.



Geoffrey Charles Fox is Director of NPAC and Professor of Computer Science and Physics and an internationally recognized expert in the use of parallel architectures and the development of concurrent algorithms. He leads a major project to develop prototype high performance Fortran (Fortran90D) compilers with language independent runtime. He has always emphasized the role of applications in driving and validating technology. This is illustrated by his recent book "Parallel Computing Works" which decribes the use of HPCC technologies in 50 significant application examples. Fox directs InfoMall, which is focused on accelerating the introduction of high speed communications and parallel computing into New York State industry and developing the coorresponding software and systems industry. Much of this activity is centered on NYNET with ISDN and ATM connectivity throughout the state including schools where Fox is leading developments of new K-12 applicatons that exploit modern technology.

Appendix 1(3)

m-Frame granular transport and buffer requirements for VBR encoded media in VOD servers *

Juan Miguel del Rosario† Marek Podgorny‡ Geoffrey Fox§

NPAC Technical Report SCCS-733

Northeast Parallel Architectures Center
111 College Place, RM 3-201
Syracuse University
Syracuse, NY 13244-4100

Mar. 16, 1995

Abstract

Combined advances in computer and communication technologies such as secondary mass storage, video compression standards, and high speed ATM networks have served as a foundation for enabling and supporting large-scale video on-demand (VOD) service. However, although these technological advances have made VOD services feasible, providing guaranteed quality, real-time video delivery still poses many technical challenges.

In this paper, we describe a framework for the transmission of smoothed VBR streams in video on-demand servers called m-frame transport. Our model allows for the specification of varying degrees of smoothing in the video source signal and provides formulae for computing the required buffer sizes for both the network server and the client.

We show that by a proper selection of the m-frame transport parameter (smoothness value), appreciable statistical multiplexing gains can be achieved. Further, we show that for a given range of smoothness values minima exist such that buffer size requirements can be reduced greatly, as compared to full CBR transmission, without significant loss in statistical multiplexing gain. Video transmission can then be accomplished in a deterministic, real-time fashion, without frame loss.

*This work was sponsored in part by the US Airforce under Rome Laboratory contract # F30602-94-C-0256.
†ECE Dept., Syracuse University
‡NPAC, Syracuse University
§CIS Dept., Syracuse University

Contents

1 Introduction **2**

1.1 Constant and variable bit rates 2

1.2 Server/client buffering 3

1.3 Contribution of this paper 3

2 m-Frame transport model **4**

2.1 Scheduler assumptions 5

2.2 Time slots 6

2.2.1 Network server 6

2.2.2 Video server 7

3 Buffer requirements **7**

3.1 Video to network server 8

3.1.1 Flow conditions 9

3.1.2 Buffer underflow 10

3.1.3 Buffer overflow 11

3.1.4 Simplified algorithm 11

3.2 Network server to client 13

3.2.1 Flow conditions 13

3.2.2 Buffer underflow 14

3.2.3 Buffer overflow 15

4 Experimental results **16**

4.1 Statistical multiplexing gain 16

4.2 Buffer requirements 24

4.3 Discussion 27

5 Conclusion **28**

1 Introduction

Combined advances in computer and communication technologies such as secondary mass storage, video compression standards, and high speed ATM networks have served as a foundation for enabling and supporting large-scale video on-demand (VOD) service. However, although these technological advances have made VOD services feasible, providing guaranteed quality, real-time video delivery still poses many technical challenges.

The problem of providing real-time scheduling and delivery mechanisms for VOD servers is often decomposed into two smaller, component problems. One deals with the problem of secondary storage allocation, and real-time scheduling and retrieval of continuous media. Another deals with scheduling and transmission of continuous media over networks (often, ATM networks). In reality, these components are interdependent and design or operational decisions for one will limit options available for the other. This situation is particularly true of factors involving transmission rate characteristics and buffer size requirements between the video server, network server, and client.

1.1 Constant and variable bit rates

In evaluating network performance, a consideration of the expected network traffic characteristics is of paramount importance. Much research work has been has been devoted to the study of video transmission using ATM networks [19], [7], [3], [6], [4], [8], [16], [12], [1], [20]. The primary issue being addressed here revolves around the performance benefits and trade-offs of using variable bit rate (VBR) encoding schemes versus constant bit rate (CBR) encoding schemes.

In CBR encoded video sources, picture quality parameters are adjusted to maintain a constant requirement for delivery. As a result, although the video traffic can be transmitted via a fixed, reserved bandwidth, degradation of picture quality often occurs during encoding. In VBR encoding, the video source is encoded with a constant picture quality. This results in a variable number of bits from frame to frame.

To maximize the efficient use of network bandwidth, it is necessary to multiplex several video sources onto the same transmission channel. It is ATM's ability to provide variable bandwidth dynamically (through statistical multiplexing) that makes it an attractive choice as a variable bit rate transport mechanism. Obviously, multiplexing several VBR sources by simply reserving the maximum bandwidth required by each source results in very inefficient use of the network bandwidth. However, problems start to arise when VBR sources are multiplexed without peak bandwidth reservations for each. Under these conditions, the source bandwidth requirements and available network bandwidth fluctuate independently of each other over time. Such fluctuations result in congestion at the network switches which, in turn, cause cells to be queued, and consequently delayed, and sometimes dropped. Furthermore, because the fluctuations (and thus the degree of congestion) occur randomly over time, the switch delay, aside from causing overall end-to-end delay, also results in increased jitter (variations in end-to-end delay) in the transmitted video stream. Although cell loss is undesirable because it leads directly to picture degradation, it can be tolerated to some extent. However, real-time video sources are extremely delay-sensitive and have very strict bounds on delay tolerance. Jitter

leads to choppiness in the output as well as possible loss of synchronization for phase sensitive streams.

A measure of the effectiveness of VBR transmission schemes has been formulated by Heeke [7] and is called the *statistical multiplexing gain*; it is defined as the ratio of the number of multiplexed VBR sources to the number of multiplexed CBR sources while maintaining an equivalent subjective picture quality. Finding a solution to the problem of multiplexing VBR signals over a single transmission channel is an active area of research. Several solution approaches have been taken including multilayer bandwidth allocation [11], switch level error control and multiplexing [2], "smoothing" of the video source [18], [19], [17], [5], and stream rate adjustment [13], [15], [14].

1.2 Server/client buffering

Delivering VBR encoded video streams at other than variable bit rates require some measure of buffering at either the server or client processors. In a recent experiment [9], a large VOD storage server was designed which contained up to 768 MBytes of main memory, and RAID storage devices. Although the storage system could support delivery of up to 86% of maximum theoretical number of concurrent streams, the reported number of concurrent streams supported reached only 30% of the theoretical maximum due primarily to server memory space limitations.

An alternative approach proposed in [10] employs buffering at the client process to achieve CBR transmission of VBR data. This strategy results in an increase in the maximum number of streams that can be multiplexed by a factor of 4.6 to 9.9 times. However, most of the buffering is placed at the client process. In a set of sample video streams, it was shown that from 3.7 to 14.6 Megabytes of buffer memory may be required. For multimedia PCs, currently being marketed with 64, and 512 Megabytes of RAM, the required buffer sizes amount to between 5.7% to 22.9%, and 0.65% to 2.9%, respectively, of the available memory. Although these buffer sizes may seem reasonable for larger capacity display devices such as workstations or PCs, they are still expected to greatly exceed the storage resources which is anticipated will be available in at least first generation set-top boxes.

1.3 Contribution of this paper

From the preceding discussion, it is clear that any strategy for video delivery in video on-demand systems must incorporate expressions for both transmission rate and buffer size components. Motivated by these issues, we present in this paper a framework for providing end-to-end delivery of variable bit rate encoded continuous media in video on-demand servers. Our model allows for arbitrary degrees of variability in the transmission rate and provides formulae for computing buffer requirements, for any given transmission rate, at both the server and client end-points.

The rest of this paper is organized as follows. Section 2 contains a description of the m-Frame transport model. Section 3 derives the formulae for resolving buffer requirements for m-Frame transport. In section 4 we present some experimental results and we conclude with section 5.

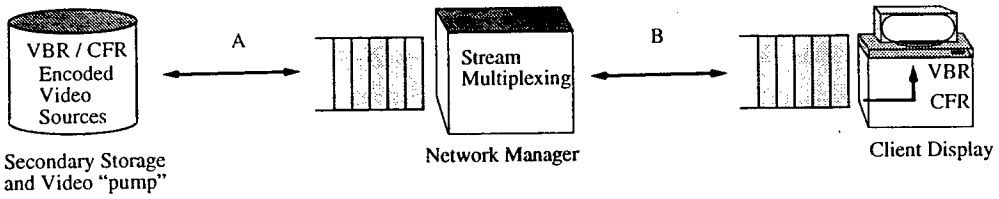


Figure 1: m-Frame transport model

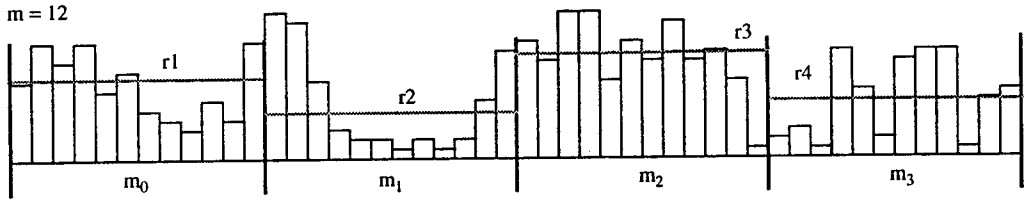


Figure 2: m-Frame granular stream

2 m-Frame transport model

A video on-demand server comprises three components: a *video server* which stores and retrieves video files from long-term storage devices, a *network server* which receives the video stream data from the video server and injects it into the network, and a client display process; and transmission connections between these as illustrated in Figure 1. The connection between the video and network servers (labelled A in the figure) is a local one (i.e., within the server machine) and the connection from the network server to the client (labelled B in the figure) is a remote one. We assume that the video source has been stored as a VBR encoded stream in secondary storage as shown. Further, we make the assumption that the network server to client connection has transmission properties similar to ATM networks in the sense that it is capable of providing variable bit rate transmission service.

Consider a typical VBR encoded video stream[1] being transmitted from the video server to the network server, or from the network server to the client. The stream is composed of a finite number, N_f, of frames of varying sizes; the size of each frame in bits is given by X_i. Given such a VBR encoded video stream, let the stream be divided into a set of sequences of some integral number of frames, $1 \leq m \leq N_f$, the last sequence of which may or may not have less than m frames. We call each sequence in the set an *m-frame sequence* and the video stream so divided an *m-frame granular stream.* Further, we say that an m-frame granular stream possesses *m-frame granularity.*

An m-frame granular stream is illustrated in Figure 2. In this case, $m = 12$. Note that each m-frame sequence is composed of variable size frames. We can define a transmission rate to be associated with each m-frame sequence as follows.

[1] In actuality, the frame sizes in an MPEG encoded stream depends upon the encoder implementation. It is possible for a CBR encoded stream to have variable frame sizes although with much greater periodicity than VBR encoded streams (e.g., a 20:4:1 bit ratio for frames I, P, and Q has been used for CBR encoding).

Take an arbitrary m-frame sequence, m_i, from an m-frame granular stream, s. Let l be the number of frames in the m-frame sequence. (In general, $l \leq m$, with equality holding if m_i is not the last sequence in s). Let R_f represent the overall required frame playout rate. We define

$$R_b^{(m,i)} = \frac{R_f \sum_{i=1}^{l} X_i}{l}$$

as the *sequence rate* for the ith m-frame sequence in s. The sequence rate for an m-frame sequence is the constant bit rate (i.e., average bit rate for the m-frame sequence) corresponding to the constant frame rate playout requirement for the stream. In Figure 2, the labels r1 through r4 represent the sequence rate for the m-frame sequences shown.

The transmission rate profile for an m-frame granular stream is determined by the source video stream profile and the granularity of the m-frame granular stream (i.e., the size of the m-frame sequence determined by the magnitude of m). An 1-frame granular stream corresponds to CBR[2] transmission (i.e., the rate is averaged over the entire video stream). An N_f-frame granular stream corresponds to fully VBR transmission. Magnitudes of m between 1 and N_f describe various degrees of smoothing. For these values of m, transmission rates are still variable, but the period of variation has increased to m times the full VBR case (the frequency also decreases proportionately).

Thus, given a video stream, we can view the stream as being divided into a set of sequences each of which is composed of the same fixed number of frames (except possibly the last sequence) with an associated constant bit rate; each sequence can then be delivered at its associated constant bit rate. With such a construction, it is possible to describe the transmission of a video stream for a full range of bit rates from CBR to full VBR.

The significance of using m-frame granular transport is that it provides a means for controlling the degree of variability in the VBR video source stream. This is equivalent to specifying the degree of smoothing applied to the stream, which has a direct effect upon achievable statistical multiplexing gain and buffer requirements. Experimental results which demonstrate the effects m-frame transport on both these quantities will be presented below.

2.1 Scheduler assumptions

Let T_c represent the *cycle* time in seconds for the server scheduler. We define the cycle time as the time it takes for the scheduler to service every concurrently supported stream once. We assume that the server employs a round-robin schedule to transmit the video stream and that during each cycle of the scheduler, a pre-determined time slice $t_c^s < T_c$ is allocated to each stream, s, as illustrated in Figure 3. The delivery framework we propose is dependent upon these assumptions about the scheduling algorithm. Note that for this paper we have assumed the simplest possible scheduling algorithm – we assume that the time, t_c^s, allocated to each stream by the server is fixed for all streams. Future papers will investigate the use of more complex scheduling algorithms which will allow t_c^s to be dynamically adjusted.

[2]For the remainder of this paper, the term CBR will refer to transmission at the constant bit rate defined by the required playout rate in frames/sec.

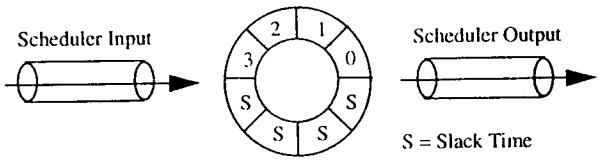


Figure 3: Round-robin scheduler

2.2 Time slots

A real-time video stream can be naturally subdivided along the temporal dimension into time *slots*, where each slot, $n_i : i \in \{0,1,2,3, \ldots, N_s\}$, refers to the time interval $[t_i, t_{i+1}]$ in the video stream transmission period. This representation is illustrated by Figure 4 where τ represents some constant time interval for each time slot.

2.2.1 Network server

Theoretically, the network server is capable of transmission at rates associated with any m-frame sequence granularity. A number of possible configurations may occur in dividing a given m-frame granular stream into time slots of some arbitrary length. Figure 5 illustrates some of the possible options. For all diagrams in the figure, each pattern identifies a distinct m-frame sequence with its associated sequence rate.

In general, there may be any number of m-frame sequences within an arbitrary time slot. Without loss of generality, suppose that a time slot contains k m-frame sequences[3] $\{m_1, m_2, \ldots, m_i, \ldots, m_k\}$ where $1 \geq k \geq N_f$, and m_1 and m_k are contained either wholly or partially. The total number of bits contained in such a time slot, n_i, for a stream, s, is given by the sum

$$\hat{C}(n_i) = \alpha_1 X^{(m_1)} + \alpha_2 X^{(m_2)} + \cdots + \alpha_k X^{(m_k)} \tag{1}$$

where $X^{(m_i)}$ represents the size in bits of m-frame sequence m_i, and α_i denotes the fraction of each m-frame sequence, m_i, that is contained in the time slot, n_i.

Alternatively, we can view time slots in terms of frames of the video stream. Let $C(n_i)$ be the number

[3]The number of m-frame sequences in each time slot is dependent upon the sequence rate profile of the m-frame granular stream, which in turn is dependent upon the frame size profile of the video source.

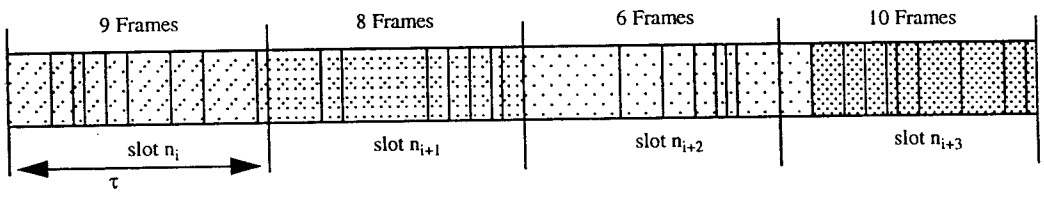


Figure 4: Video stream time slots

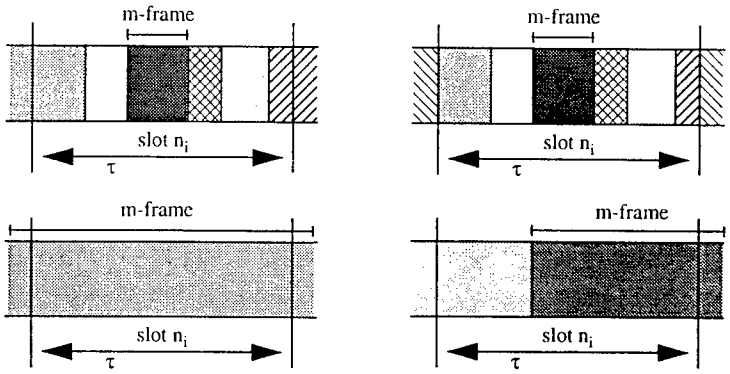


Figure 5: m-Frame granular stream and time slots

of frames contained in time slot n_i. $C(n_i)$ can be determined by a modification of equation 1 into

$$C(n_i) = \beta_1 X_f^{(m_1)} + \beta_2 X_f^{(m_2)} + \cdots + \beta_k X_f^{(m_k)} \tag{2}$$

where $X_f^{(m_i)}$ represents the size in frames of m-frame sequence m_i, and β_i denotes the number of frames from each m-frame sequence, m_i, that is wholly contained or terminates in the time slot, n_i. This is illustrated in Figure 4, the m-frame boundaries are not shown for simplicity. Note that in each slot we count only the number of complete frames within the slot; thus from the figure, the last frame in the third slot is counted towards the fourth slot.

In an actual implementation, the network interface unit usually transmits data at some constant bit rate greater than or equal to the required maximum transmission rate. Under this condition, the varying sequence rate (induced by m-frame granularity is) is effected by regulating the duration in transmission time per slot. This results in an idle network during portions of the allocated time slot. Alternative scheduling algorithms may be used to take advantage of this condition and allow for an increase in the number of streams servicable. We are currently investigating the use of such scheduling algorithms.

2.2.2 Video server

A fundamental contraint is imposed on our system by the fact that video streams are stored in secondary storage devices. By nature, such devices are restricted to providing constant bit rate service. Therefore our model must accommodate this limitation. In accordance with this, we view the video stream as being divided into time slots of equal size in bits, ignoring frame sizes and boundaries as illustrated previously in Figure 4.

3 Buffer requirements

Any departure in transmission rates from full VBR transmission is constrained by the real-time property of motion video to require buffering. In this section, we quantify the buffer requirement for any given video

stream with some arbitrary m-frame granularity. Two different cases must be considered, first, transmission from the video server to the network server, second, transmission from the network server to the client.

3.1 Video to network server

In this section, we describe the nature of data flow from the video server to the network server, and analyze the buffer size requirement at the network server for an arbitrarily selected m-frame granularity.

A video server transfers data at a constant bit rate. We assume that the network server transmits data into the network at a variable bit rate whose profile is determined by an associated m-frame granularity. In other words, we assume that the network server views the video stream as an m-frame granular stream and transmits each m-frame sequence at its associated sequence rate, resulting in an overall variable bit rate transmission for the video stream. Buffering is performed at the network server in proportion with the degree of variability in the network server transmission rate.

For any given video stream, s, its total length in bits is given by $L^s = \sum_{i=1}^{N_f} X_i$. Assuming a desired playout rate in frames/sec of R_f, the playout time in seconds for the entire stream is $T^s = \frac{N_f}{R_f}$, and the corresponding average bit rate requirement is $R_b = \frac{L^s}{T^s}$. We assume that the video server transfers data to the network server at this constant bit rate.

Due to the real-time and continuous nature of a video stream, at each cycle of the video server scheduler, the video server must deliver a sufficient amount of video data to satisfy the network server consumption for the entire cycle time, T_c. That is, it must deliver $\hat{A} = R_b \text{x} T_c$ bits of data per cycle. We assume, as mentioned previously, that each stream gets allocated some portion, t_c^s, of T_c by the video server scheduler. In order to satisfy the network server's consumption requirement per cycle, the video server must deliver $\hat{A}$ bits of data in time t_c^s. Hence, for each stream, the bandwidth requirement for delivering the video stream is equal to $R_{b_c} = \frac{\hat{A}}{t_c^s} = R_b \frac{T_c}{t_c^s}$.

The rate of consumption at the network server is equivalent to its rate of transmission. From section 2.2.1, the number of bits contained in a given time slot, n_i, for a given stream, s, is given by equation 1. Let $\hat{B}(n_i)$ be the number of bits in the network server buffer at the beginning of slot n_i (i.e., at time t_i), and let $\hat{A}(n_i)$ be the number of bits arriving at the network server buffer during slot n_i. Then, for each time slot, the number of bits remaining in the buffer is given by the recursive equation

$$\hat{B}(n_{i+1}) = \hat{B}(n_i) + \hat{A}(n_i) - \hat{C}(n_i), \qquad \forall i \in \{0, 1, 2, \ldots, N_s\}$$

Consider the arrival rate, $\hat{A}(n_i)$, at the network server. Since we transmit the video stream at a constant bit rate and the size of each frame, X_i, may vary, there is a variable frame arrival rate. Since the number of incoming bits for any slot may be less than that necessary for consumption, we cannot rely on incoming data to satisfy the current slot consumption requirements. Therefore, we have to satisfy the current transmission requirements solely from the buffer, network transmission will be interrupted if the buffer never underflows for any of the time slots. Hence, we have the requirement

$$\hat{B}(n_{i+1}) = \hat{B}(n_i) + \hat{A}(n_i) - \hat{C}(n_i) \geq 0 \qquad \forall i \in \{0, 1, 2, \ldots, N_s\}$$

Since this is true for all slots n_i, without loss of generality it is true for some slot n_k. Then, the recursive formula above for the buffer requirement can be written as

$$\hat{B}(n_k) = \hat{B}(n_0) + \sum_{i=0}^{k-1} \hat{A}(n_i) - \sum_{i=0}^{k-1} \hat{C}(n_i) \geq 0 \qquad \forall k \in \{0, 1, 2, \ldots, N_s\} \tag{3}$$

Thus, for any point throughout the transmission of the video stream (from beginning to end), the network server buffer must never underflow.

3.1.1 Flow conditions

For a given time slot, if the number of bits or frames that arrive in that slot is less than the amount consumed (e.g., $\hat{A}(n_i) < \hat{C}(n_i)$) we call this slot an *underflow* slot; if it is greater than the amount consumed (e.g., $\hat{A}(n_i) > \hat{C}(n_i)$), we call it an *overflow* slot; if it is equal to the amount consumed (e.g., $\hat{A}(n_i) = \hat{C}(n_i)$), we call it an *even* slot. (Whether bits or frames is the metric currently being used will be made clear from the context.) We make the assumption that data for a given slot will arrive at the client and be buffered prior to the start of its consumption. During the playout of a video stream, there will be a combination of overflow, underflow and even slots. Thus we have

$$\sum_{i=0}^{K'} \hat{A}(n_i) = \sum_{i \in K'_u} \hat{A}(n_i) + \sum_{i \in K'_o} \hat{A}(n_i) + \sum_{i \in K'_e} \hat{A}(n_i) \tag{4}$$

where K'_u, K'_o, and K'_e are the set of underflow, overflow, and even slots respectively, and $|K'_u| + |K'_o| + |K'_e| = K'$, where $K' \in \{0, 1, 2, \ldots, N_s\}$.

From equation 3 and 4 we have, $\forall K' \in \{0, 1, 2, \ldots, N_s\}$

$$\begin{aligned}
&\hat{B}(n_0) + \sum_{i \in K'_u} \hat{A}(n_i) + \sum_{i \in K'_o} \hat{A}(n_i) + \sum_{i \in K'_e} \hat{A}(n_i) - \sum_{i=0}^{K'} \hat{C}(n_i) \geq 0 \\
\iff &\hat{B}(n_0) + \sum_{i \in K'_u} \hat{A}(n_i) + \sum_{i \in K'_o} \hat{A}(n_i) + \sum_{i \in K'_e} \hat{A}(n_i) - \sum_{i \in K'_u \cup K'_o} \hat{C}(n_i) - \sum_{i \in K'_e} \hat{C}(n_i) \geq 0 \\
\iff &\hat{B}(n_0) + \sum_{i \in K'_u} \hat{A}(n_i) + \sum_{i \in K'_o} \hat{A}(n_i) - \sum_{i \in K'_u \cup K'_o} \hat{C}(n_i) \geq 0
\end{aligned} \tag{5}$$

where, by definition, $\hat{A}(n_i) < \hat{C}(n_i) \;\; \forall j \in K'_u$ and $\hat{A}(n_i) > \hat{C}(n_i) \;\; \forall j \in K'_o$.

There are now two conditions which must be accounted for in buffering. The first is the *buffer underflow* condition which describes the situation where, during some point in the video transmission, the buffer runs empty and there is nothing for the network server to transmit. The second condition is the *buffer overflow* condition which describes the situation where the buffer is full and can not accommodate the incoming bits for the current slot. We address these two cases below.

3.1.2 Buffer underflow

Consider the case where the video stream is transmitted to the network server without prefetching a few frames into its buffer. That is, at the start of transmission at time t_0, the buffer is empty (i.e., $\hat{B}(n_0) = 0$). Then, the buffer underflow condition comes about when for some time slot during the transmission of the video, the total number of bits that have arrived is less than that required for tranmission (consumed). Thus we have, for some time slot, $N_{K'}$, where $K' = |K'_u| + |K'_o| + |K'_e|$

$$\begin{aligned} \sum_{i \in K'_u} \hat{A}(n_i) + \sum_{i \in K'_o} \hat{A}(n_i) &< \sum_{i \in K'_u \cup K'_o} \hat{C}(n_i) \\ \iff \sum_{i \in K'_u} \hat{A}(n_i) + \sum_{i \in K'_o} \hat{A}(n_i) - \sum_{i \in K'_u \cup K'_o} \hat{C}(n_i) &< 0 \end{aligned}$$

Therefore for equation 5 to hold, we require

$$\begin{aligned} \forall K', \qquad \hat{B}(n_0) &\geq \left| \sum_{i \in K'_u} \hat{A}(n_i) + \sum_{i \in K'_o} \hat{A}(n_i) - \sum_{i \in K'_u \cup K'_o} \hat{C}(n_i) \right| \\ &\geq \left| \sum_{i \in K'_u} \hat{A}(n_i) - \sum_{i \in K'_u} \hat{C}(n_i) + \sum_{i \in K'_o} \hat{A}(n_i) - \sum_{i \in K'_o} \hat{C}(n_i) \right| \qquad (6) \\ &\geq \left| \sum_{i=0}^{N_{K'}} (\hat{A}(n_i) - \hat{C}(n_i)) \right| \qquad (7) \end{aligned}$$

where equation 7 follows from 6 because $\sum_{i \in K'_e} \hat{A}(n_i) = \sum_{i \in K'_e} \hat{C}(n_i)$ by definition. Hence, to prevent the buffer underflow condition, we need to transmit a number of frames into the client buffer before initiating the display. Since equation 7 must be true for all K',

$$\hat{B}(n_0) = \max_{K'} \left(\left| \sum_{i=0}^{N_{K'}} (\hat{A}(n_i) - \hat{C}(n_i)) \right| \right) \qquad (8)$$

gives the required number of bits to buffer prior to network transmission in order to avoid buffer underflow at the network server.

To facilitate the discussion above, assumptions were made about the nature of the stream with regards to generating either overflow or underflow conditions. In a true implementation, the video stream will comprise a combination of overflow and underflow slots for each segment, $N_{K'}$, of the stream. Therefore, we need to modify equations 8 to accommodate this situation. Equation 8 is modified as follows. We let $\hat{F}$ be defined, for all K', as

$$\hat{F} = \min_{K'} \left(\sum_{i=0}^{N_{K'}} (\hat{A}(n_i) - \hat{C}(n_i)) \right)$$

where the maximum underflow condition will be represented by the most negative value of $\hat{F}$. We can now define the underflow buffer requirement as

$$\hat{B}(n_o) = \begin{cases} -\hat{F} & \text{if } \hat{F} < 0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

3.1.3 Buffer overflow

The buffer overflow condition is similar to the underflow case but the initial condition is now slightly different. The buffer overflow condition comes about when for some time slot during the transmission of the video, the difference between the total number of bits that have arrived and those consumed is greater than the amount containable in the buffer. To minimize the overflow buffer requirement, we assume the initial underflow bits to exist in the network server buffer (i.e., $\hat{B}(n_0)$ is given by equation 9). It follows for the overflow condition that

$$\begin{aligned} & \sum_{i \in K'_u} \hat{A}(n_i) \quad + \quad \sum_{i \in K'_o} \hat{A}(n_i) > \sum_{i \in K'_u \cup K'_o} \hat{C}(n_i) \\ \Longleftrightarrow & \sum_{i \in K'_u} \hat{A}(n_i) \quad + \quad \sum_{i \in K'_o} \hat{A}(n_i) - \sum_{i \in K'_u \cup K'_o} \hat{C}(n_i) > 0 \end{aligned}$$

Therefore equation 5 holds. But, to prevent buffer overflow we require that

$$\begin{aligned} \forall K' \qquad \hat{B}_{max} &\geq \hat{B}(n_0) + \sum_{i \in K'_u} \hat{A}(n_i) + \sum_{i \in K'_o} \hat{A}(n_i) - \sum_{i \in K'_u \cup K'_o} \hat{C}(n_i) \\ &\geq \hat{B}(n_0) + \sum_{i \in K'_u} \hat{A}(n_i) - \sum_{i \in K'_u} \hat{C}(n_i) + \sum_{i \in K'_o} \hat{A}(n_i) - \sum_{i \in K'_o} \hat{C}(n_i) \qquad (10) \\ &\geq \hat{B}(n_0) + \sum_{i=0}^{N_{K'}} (\hat{A}(n_i) - \hat{C}(n_i)) \qquad (11) \end{aligned}$$

again equation 11 follows from equation 10 because $\sum_{i \in K'_c} \hat{A}(n_i) = \sum_{i \in K'_c} \hat{C}(n_i)$ by definition. Since equation 11 is must be true for all K',

$$\hat{B}_{max} \geq \hat{B}(n_0) + \max_{K'} \left(\sum_{i=0}^{N_{K'}} (\hat{A}(n_i) - \hat{C}(n_i)) \right) \tag{12}$$

gives the required buffer size to avoid overflow.

3.1.4 Simplified algorithm

A simple algorithm for performing a computation equivalent to that defined by equation 12 is given below.

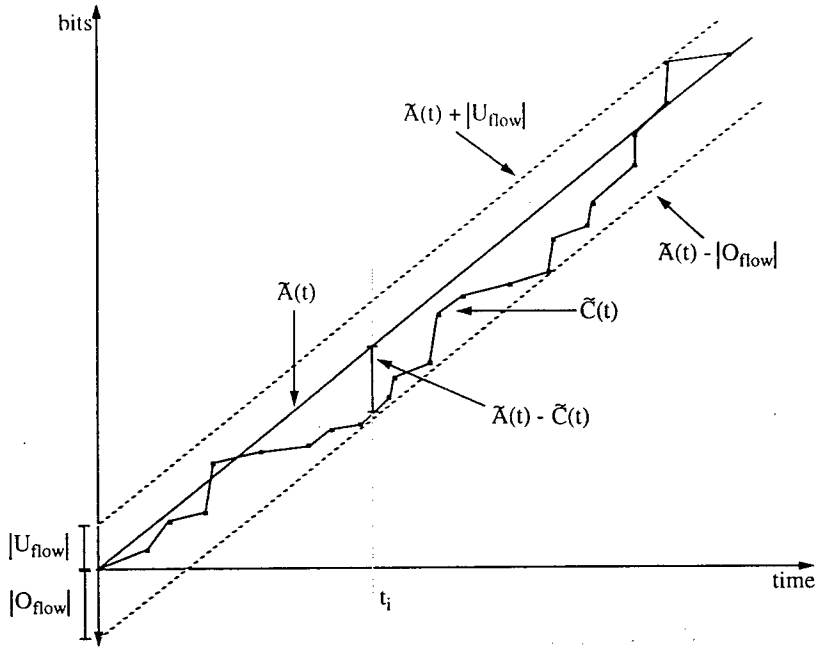


Figure 6: Video to network server buffer overflow and underflow limits

For each m-frame sequence m_i, $i = \{1, 2, \ldots, \frac{N_f}{m}\}$

Compute the size of m_i in bits from $\hat{C}^{(m_i)} = \sum_{i=1}^{l} X_i$.

Compute the playout duration for m_i by $t_p^{(m_i)} = \frac{\hat{B}^{(m_i)}}{R_b^{(m_i)}}$, where $R_b^{(m_i)}$ is the sequence rate for m_i.

For the same duration, compute the equivalent number of bits for delivery at the constant arrival rate, R_b, by $\hat{A}^{(m_i)} = R_b t_p^{(m_i)}$.

Compute difference in arrival and consumption quantities by $D = \hat{A}^{(m_i)} - \hat{C}^{(m_i)}$.

If $(D < U_{flow})$ $U_{flow} = D$. Store underflow buffer requirement.

If $(D > O_{flow})$ $O_{flow} = D$. Store overflow buffer requirement.

End for.

Return buffer requirement $\hat{B}_{max} = U_{flow} + O_{flow}$.

This algorithm capitalizes upon the cumulative nature of equations 9 and 12. To describe the main idea, we define an accumulation function, $\tilde{C}(t)$, which returns the sum in bits of all the m-frame sequences that would be consumed (either wholly or partially) in the time interval $[0, t]$ when each is transmitted at its sequence rate. We further define the number of arriving bits in the interval $[0, t]$ assuming a constant bit rate R_b as $\tilde{A} = R_b t$. As illustrated in the diagram of Figure 6, for each time point, t_i, we can compute the difference $D = \tilde{A} - \tilde{C}(t)$. The minimum value (the most negative) of D will give the underflow buffer requirement; similarly, the maximum value will give the overflow buffer requirement as shown in the figure.

Notice that superimposing time slot intervals over the diagram and computing for D at time slot boundaries would, in effect, represent equations 9 and 12. However, since $C(t)$ is piece-wise linear according to m-frames, we can simplify the operation by computing for D at m-frame sequence boundaries rather than at time slot boundaries, and still arrive at the same results; this is the basis for the above algorithm.

3.2 Network server to client

In this section, we describe the nature of data flow from the network server to the client display process, and analyze the buffer size requirement at the client process for a stream delivered with some arbitrarily selected m-frame granularity. Assumptions made in the section above regarding the network server transmission properties remain the same. On the client side, we assume that the display process consumes data at some constant frame rate.

Consider the client process. Let $E(n_i)$ be the number of frames consumed by the client in slot n_i; this value is a constant and is equivalent to the playout frame rate (e.g., 30 frames/sec for full motion video). Let $D(n_i)$ be the number of frames in the buffer at the beginning of slot n_i (i.e., at time t_i). Let $C(n_i)$ be the number of frames arriving during slot n_i; this is the frame arrival rate at the client process. $C(n_i)$ is determined by equation 2 from section 2.2.1.

As in the video to network server case, we have the requirement

$$D(n_{i+1}) = D(n_i) + C(n_i) - E(n_i) \geq 0 \qquad \forall i \in \{0, 1, 2, \ldots, N_s\}$$

Since this is true for all slots n_i, without loss of generality it is true for some slot n_m. Thus, we have

$$D(n_m) = D(n_0) + \sum_{i=0}^{k-1} C(n_i) - \sum_{i=0}^{k-1} E(n_i) \geq 0 \qquad \forall k \in \{0, 1, 2, \ldots, N_s\} \tag{13}$$

3.2.1 Flow conditions

For the following discussion, underflow, overflow, and even slots are determined from units of frames rather than bits. That is, the case $C(n_i) < E(n_i)$ defines an underflow slot, the case $C(n_i) > E(n_i)$ defines an overflow slot, the case and $C(n_i) = E(n_i)$ defines an even slot. As before, there will be a combination of overflow, underflow and even slots during the playout of a video stream, and we make the same assumption with regards to having slot data arrive at the buffer prior to its scheduled consumption. Thus we have

$$\sum_{i=0}^{K} C(n_i) = \sum_{i \in K_u} C(n_i) + \sum_{i \in K_o} C(n_i) + \sum_{i \in K_e} C(n_i) \tag{14}$$

where K_u, K_o, and K_e are the set of underflow, overflow, and even slots respectively, and $|K_u| + |K_o| + |K_e| = K$, where $K \in \{0, 1, 2, \ldots, N_s\}$.

From equation 13 and 14 we have, $\forall K \in \{0, 1, 2, \ldots, N_s\}$

$$D(n_0) \quad + \quad \sum_{i \in K_u} C(n_i) + \sum_{i \in K_o} C(n_i) + \sum_{i \in K_e} C(n_i) - \sum_{i=0}^{K} E(n_i) \geq 0$$

$$\iff D(n_0) \quad + \quad \sum_{i \in K_u} C(n_i) + \sum_{i \in K_o} C(n_i) + \sum_{i \in K_e} C(n_i) - \sum_{i \in K_u \cup K_o} E(n_i) - \sum_{i \in K_e} E(n_i) \geq 0$$

$$\iff D(n_0) \quad + \quad \sum_{i \in K_u} C(n_i) + \sum_{i \in K_o} C(n_i) - \sum_{i \in K_u \cup K_o} E(n_i) \geq 0 \tag{15}$$

where, by definition, $C(n_i) < E(n_i) \quad \forall j \in K_u$ and $C(n_i) > E(n_i) \quad \forall j \in K_o$.

3.2.2 Buffer underflow

Consider the case where the video stream is transmitted to the client without prefetching a few frames into the client buffer. That is, at the start of display at time t_0, the buffer is empty (i.e., $D(n_0) = 0$). Then, the buffer underflow condition comes about when for some time slot during the transmission of the video, the total number of frames that have arrived is less than that required for display (consumed). Thus we have, for some time slot, N_K, where $K = |K_u| + |K_o| + |K_e|$

$$\sum_{i \in K_u} C(n_i) + \sum_{i \in K_o} C(n_i) \quad < \quad \sum_{i \in K_u \cup K_o} E(n_i)$$

$$\iff \sum_{i \in K_u} C(n_i) + \sum_{i \in K_o} C(n_i) - \sum_{i \in K_u \cup K_o} E(n_i) \quad < \quad 0$$

Therefore for equation 15 to hold, we require

$$\forall K, \qquad D(n_0) \quad \geq \quad \left| \sum_{i \in K_u} C(n_i) + \sum_{i \in K_o} C(n_i) - \sum_{i \in K_u \cup K_o} E(n_i) \right|$$

$$\geq \quad \left| \sum_{i \in K_u} C(n_i) - \sum_{i \in K_u} E(n_i) + \sum_{i \in K_o} C(n_i) - \sum_{i \in K_o} E(n_i) \right| \tag{16}$$

$$\geq \quad \left| \sum_{i=0}^{N_K} (A(n_i) - C(n_i)) \right| \tag{17}$$

where equation 17 follows from 16 because $\sum_{i \in K_e} C(n_i) = \sum_{i \in K_e} E(n_i)$ by definition. Hence, to prevent the buffer underflow condition, we need to transmit a number of frames into the client buffer before initiating the display. Since equation 17 must be true for all K,

$$D(n_0) = \max_K \left(\left| \sum_{i=0}^{N_K} (C(n_i) - E(n_i)) \right| \right) \tag{18}$$

gives the required number of frames to buffer prior to display in order to avoid buffer underflow. Note that we compute the buffer size here in units of frames because consumption is in terms of frames (i.e., VBR/CFR) independent of individual frame size.

As we had done previously for the video to network server buffer computations (section 3.1.1, equation 9), we modify equation 18 to accommodate the general case for a true implementation. We let F be defined, for all K, as

$$F = \min_{K}(\sum_{i=0}^{N_K}(C(n_i) - E(n_i)))$$

where the maximum underflow condition will be represented by the most negative value of F. We can now define the underflow buffer requirement as

$$D(n_o) = \begin{cases} -F & \text{if } F < 0 \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

3.2.3 Buffer overflow

The buffer overflow condition here is analogous to that computed in section 3.1.1 which resulted in equation 12. In this case we have, for all K'

$$\hat{D}_{max} \geq \hat{D}(n_0) + \max_{K'}(\sum_{i=0}^{N_{K'}}(\hat{C}(n_i) - \hat{E}(n_i))) \tag{20}$$

as the required buffer size to avoid overflow.

An additional bit of computation is required for this overflow case. Since it is important not to alter the arrival stream's overflow/underflow characteristics, the evaluation process for overflow buffer requirement must be conducted at time slot boundaries. This guarantees that the associated overflow computations hold by assuring that they begin at the same time slot boundaries as those used for underflow computation. In order to minimize the buffer allocation, instead of beginning at the first time slot, we first round the number of frames given by equation 19 to the nearest time slot worth of frames, then, we begin overflow evaluation at the start of that time slot. Further, we subtract from the computation for the starting time slot the number of frames that are to be included in the underflow buffer. We describe this algorithm more formally in the following.

Define $n_{D(n_o)}$ as the number of the last slot occupied by the underflow buffer frames. Let $V(n_{D(n_o)})$ be the number of frames from $D(n_o)$ in its last slot, $n_{D(n_o)}$. Then, the amount of overflow for the first slot of overflow computation is given by

$$\hat{W}(n_{D(n_o)}) = \hat{C}(n_{D(n_o)}) - \hat{V}(n_{D(n_o)}) - \hat{C}(n_{D(n_o)})$$

where $\hat{V}(n_{D(n_o)})$ is $V(n_{D(n_o)})$ in units of bits. The general idea of the formulation is illustrated in Figure 7

Finally, calculating for the overall overflow condition we have, for all K',

$$\hat{D}_{max} \geq \hat{D}(n_0) + \max_{K'}\{\hat{W}(n_{D(n_o)}), \sum_{i=n_{D(n_o)}+1}^{N_{K'}}(\hat{C}(n_i) - \hat{E}(n_i))\} \tag{21}$$

where $\hat{D}(n_0)$ is $D(n_0)$ in units of bits, and $\hat{D}_{max}$ defines the maximum buffer allocation requirement for the video stream.

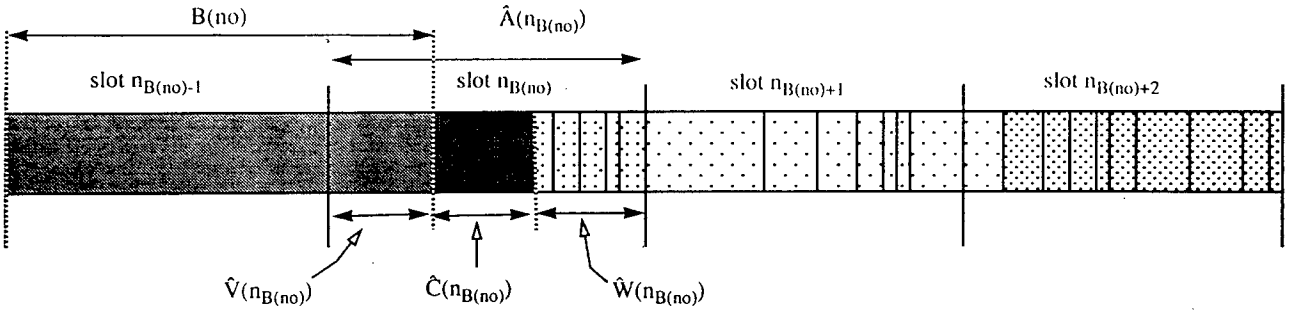


Figure 7: Overflow buffer computation – first slot

4 Experimental results

In this section we present some results for two sample video encodings, one for Jurassic Park and another for MTV. The video sources comprise a 10,000 frame (approx. 15 mins.) segment of the full movies and were captured using a MultiVideo[4] card and were later compressed to MPEG-1. Jurassic Park was captured with a MultiVideo Q-factor of 105, and MTV had a Q-factor of 50 (the higher the Q-factor, the lower the resolution) [5]. MPEG compression was performed at the same quality factors for both videos.

4.1 Statistical multiplexing gain

Figure 8 and Figure 9 show the pdf's for the original VBR encoding of Jurassic Park and MTV respectively. Figure 10 shows the pdf for MTV with $m = 100$, and Figure 11 shows the pdf for the MTV source with $m = 500$.

The effects of m-frame transport on statistical multiplexing gain is illustrated for MTV in Figures 12 , 13, and 14. Figure 12 shows the statistical multiplexing gain for the original VBR encoding of MTV (i.e., $m = 1$) for various n-fold multiplexed signals. Figures 13 and 14 show the statistical multiplexing gain for the MTV sample with values $m = 500$ and $m = 1000$ respectively. The graphs are obtained by computing the pdf of the n-fold convolution of the individual source signal at each m value. The pdf's are displayed as the negative cumulative distribution of each n-fold pdf; this represents the probability that a given bit rate is exceeded and is proportional to the cell loss probability for the multiplexed signal. For all the graphs, we have assumed that the sources are independent and that the queue lengths and maximum bandwidth are unlimited [20].

From the figures we notice that significant statistical multiplexing gain can be obtained versus the original source signal (i.e., $m = 1$), and that the statistical multiplexing gain obtained for $m = 500$ and $m = 1000$ are comparable in magnitude. To better compare the statistical multiplexing gain obtainable as a function of m, we examine Table 1. The tables show the rate requirement and statistical multiplexing gain for each channel of 32 multiplexed signals (i.e., $m = 32$) versus the original VBR signal (i.e., $n = 1$ and $m = 1$). Further, Table 2 shows the gain for a single signal (i.e., $n = 1$) as a function of m. Note that the gains

[4] MultiVideo is a trademark of Parallax Graphics, Inc.
[5] The manufacturer recommended Q-factor for general use is 150.

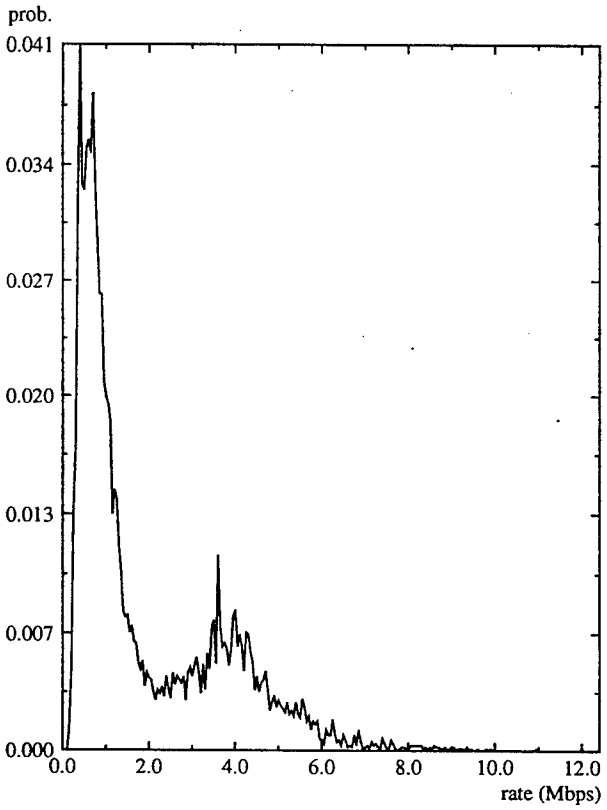


Figure 8: pdf for VBR encoding of Jurassic Park sample

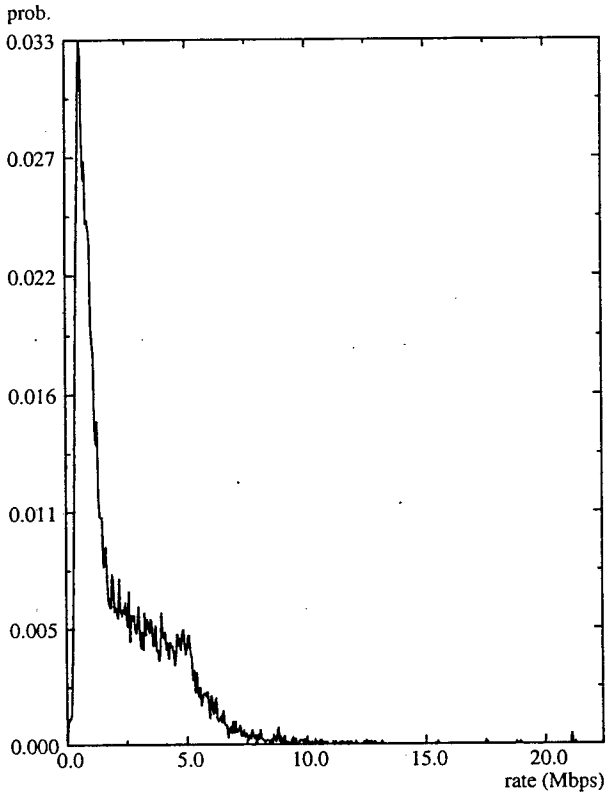


Figure 9: pdf for VBR encoding of MTV sample

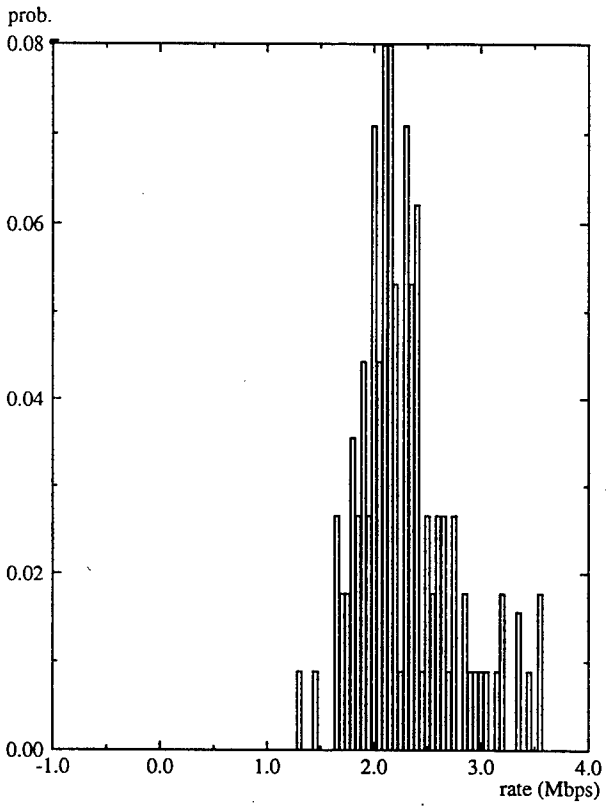


Figure 10: pdf for MTV sample with $m = 100$

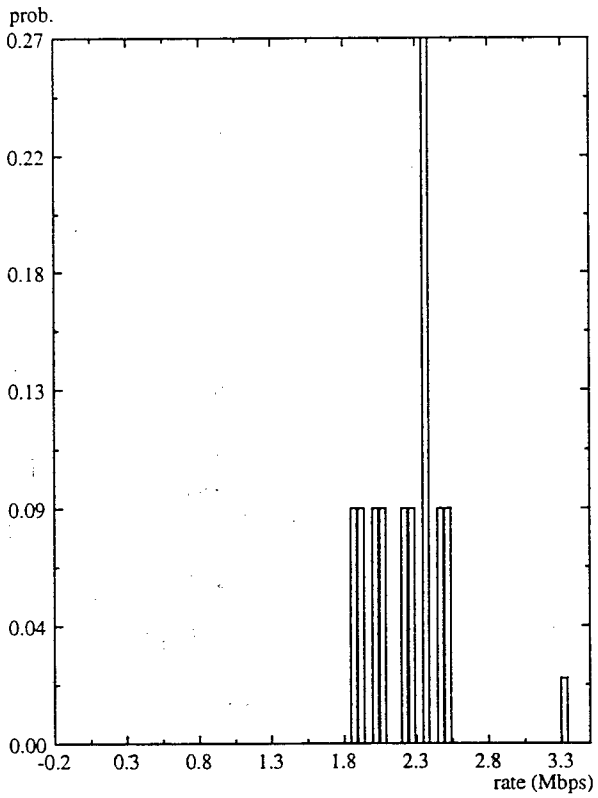


Figure 11: pdf for MTV sample with $m = 500$

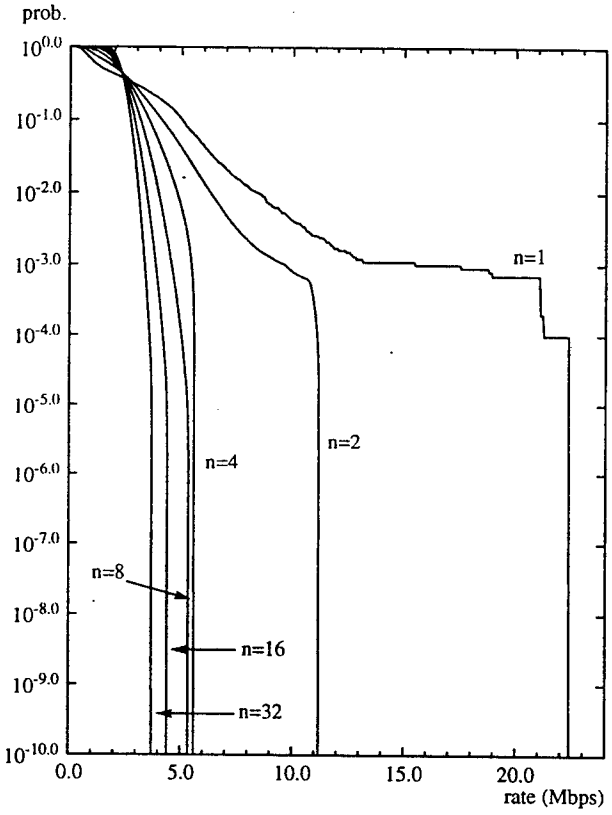


Figure 12: Statistical multiplexing for MTV with $m = 1$

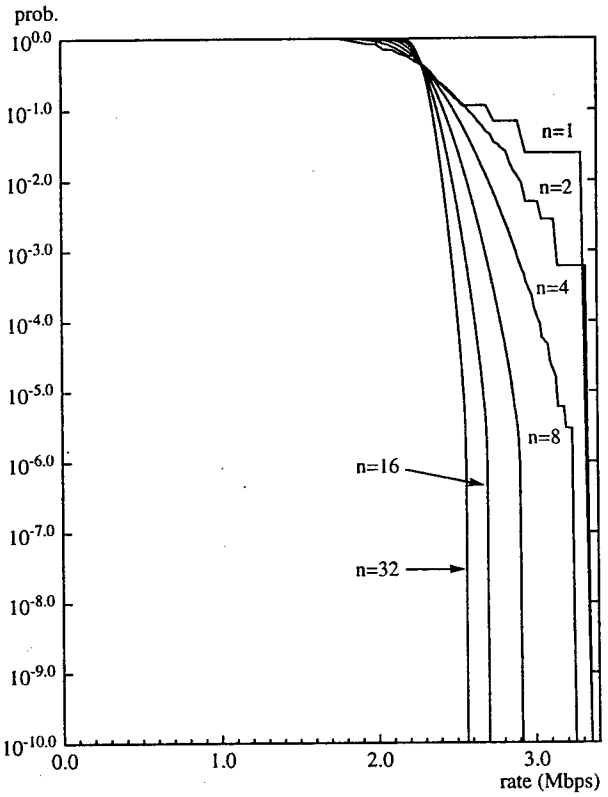


Figure 13: Statistical multiplexing for MTV with $m = 500$

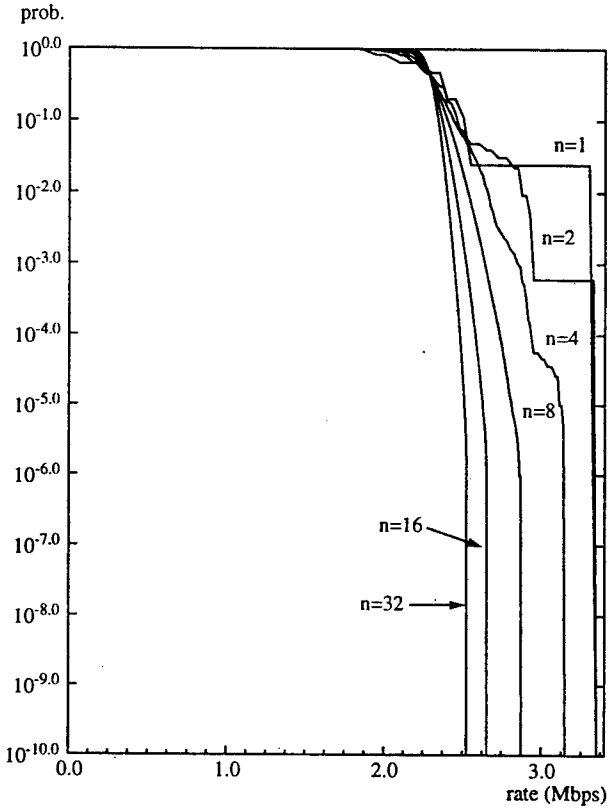


Figure 14: Statistical multiplexing for MTV with $m = 1000$

Table 1: Statistical multiplexing gain as a function of m, for $n = 32$, and for probability $\leq 10^{-8}$ (required rate for (m,n) = (1,1) is 22.4 Mbps for MTV, and 12.4 Mbps for Jurassic Park)

m	MTV		Jurassic Park	
	Rate	Gain	Rate	Gain
1	3.707895	6.04	3.309433	3.75
20	2.825029	7.92	2.387502	5.19
30	2.796902	8.01	2.357813	5.26
40	2.743774	8.17	2.340624	5.30
50	2.745336	8.16	2.324998	5.34
100	2.650018	8.45	2.270307	5.46
200	2.612516	8.58	2.224992	5.58
300	2.596890	8.63	2.185927	5.68
400	2.584389	8.67	2.143737	5.79
500	2.565638	8.73	2.162488	5.74
1000	2.526573	8.87	2.104672	5.89
2000	2.457819	9.12	2.031230	6.11
3000	2.342187	9.56	2.057794	6.03
4000	2.346874	9.55	1.995291	6.21
5000	2.432817	9.21	2.031230	6.11
6000	2.337499	9.58	2.012479	6.16
7000	2.362500	9.48	2.043731	6.07
8000	2.357813	9.50	1.959354	6.48
9000	2.339061	9.58	1.949979	6.36
10000	2.446881	9.15	1.949979	6.36

shown in Table 2 are guaranteed and result from the smoothing effects of the m-frame scheme, they are not dependent upon any statistical averaging as are the results from the previous table for $n = 32$.

The table shows that significant additional statistical multiplexing gain can be obtained from the smoothing effects of m-frame transport although the incremental gain is expected to diminish as m approaches the size of the video stream.

4.2 Buffer requirements

Figure 15 describes the buffer requirement for Jurassic Park at various values of m; Figure 16 shows the equivalent results for the MTV source. The graphs represent buffer requirements for the network server to client segment of the transmission path (i.e., m-frame transport for transmission, and VBR for consumption).

From the figure it is clear that varying the values of m can result in significant reductions in the client side buffer requirement. For instance, from Figure 15 we see that the maximum buffer requirement approaches approximately 4 Megabytes as one approaches the average transmission rate (i.e., as $m \rightarrow 10,000$). However, there is a minumum at around $m = 3000$ which results in a buffer requirement of only about 1.5 Mbytes. Further, from the discussion above we have seen that there is not a significant loss in statistical multiplexing gain in going from $m = 10,000$ to $m = 3000$.

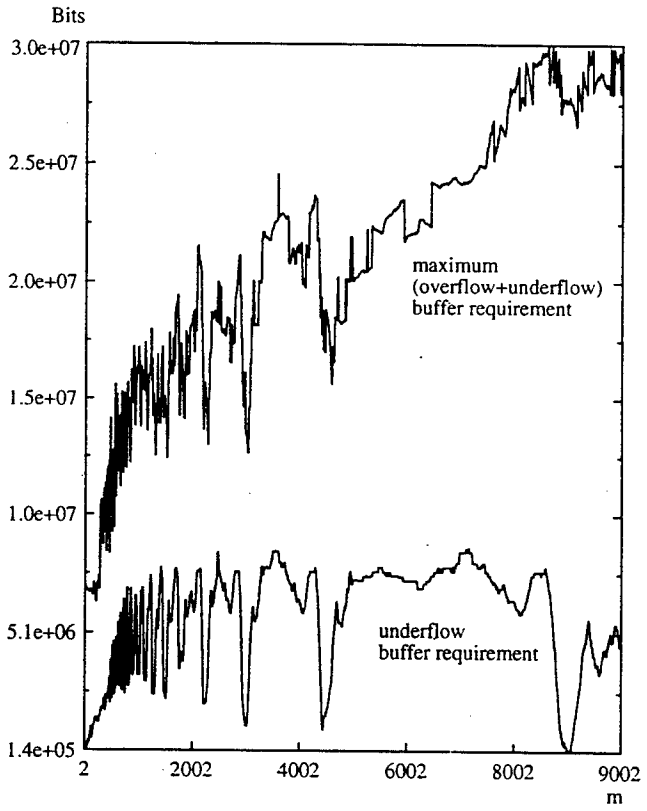


Figure 15: Jurassic Park: Buffer requirement at various m values

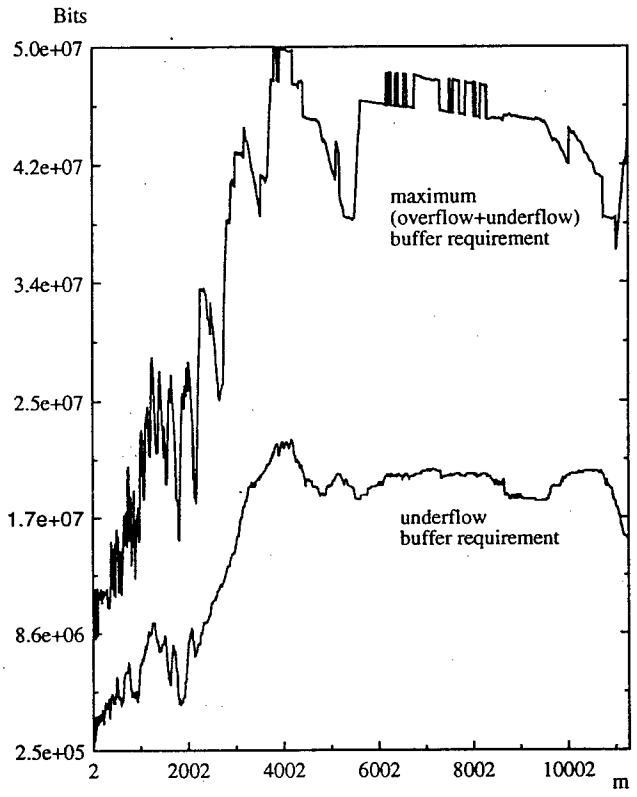


Figure 16: MTV: Buffer requirement at various m values

Table 2: Guaranteed multiplexing gain as a function of m, for $n = 1$, and for probability $\leq 10^{-8}$

m	MTV		Jurassic Park	
	Rate	Gain	Rate	Gain
1	12.35	1	22.35	1
20	4.35	2.84	4.85	4.60
30	3.85	3.20	5.25	4.25
40	3.65	3.38	4.65	4.80
50	3.45	3.57	4.30	5.19
100	3.25	3.80	3.50	6.38
200	3.10	3.98	3.35	6.67
300	2.70	4.57	3.30	6.77
400	2.55	4.84	3.30	6.77
500	2.60	4.75	3.30	6.77
1000	2.20	5.61	3.30	6.77
2000	2.05	6.02	2.65	8.43
3000	2.10	5.88	2.35	9.51
4000	1.95	6.33	2.35	9.51
5000	2.00	6.17	2.65	8.43
6000	2.00	6.17	2.30	9.71
7000	2.05	6.02	2.35	9.51
8000	1.95	6.33	2.35	9.51
9000	1.90	6.50	2.25	9.93
10000	1.90	6.50	2.65	8.43

4.3 Discussion

Notice from Figures 15 and 16 that in general, there is an increase in buffer size requirement as a function of m, which agrees with intuition. This indicates that in order to minimize the buffer size requirements for a segment, it is necessary to minimize m; this occurs however at the expense of statistical multiplexing gain. Moreover, both segments of transmission, video server to network server and network server to client will have their own functional dependencies for buffer requirement as a function of m. These will tend to be contradictory since the video server transfers data at constant bit rate to the network server which consumes the stream according to the selected m-frame granularity. This implies that the buffer requirements for the video server to network server segment will tend to vary inversly as a function of m (i.e., minimum buffer requirement exists when going from constant a bit rate data transfer at the video server to constant bit rate consumption by the network server). Therefore, the buffer requirement will tend to shift from the client to the network server with decreasing m values. Note however, that these are only general tendencies since the actual functional profile for buffer requirement as a function of m depends upon the source signal profile which is a stochastic process.

In an implementation, a VOD server can divide the range of m values for a given video stream into regular intervals. The minimum buffer requirement for each interval can then be stored as metadata along with the video stream. An admission control procedure can then be formulated where client buffer capacity and an acceptable range of m values (derived from bandwidth or acceptable cell loss probability limits) are

supplied by the user as quality of service parameters.

5 Conclusion

In this paper, we have described a framework for the transmission of smoothed VBR streams in video on-demand servers. Our model allows for the specification of varying degrees of smoothness in the video source signal and provides formulae for computing the required buffer sizes for both the network server (within the VOD server) and the client.

We have shown that by a proper selection of m-frame granularity, buffer size requirements can be reduced greatly (as compared to full CBR transmission) without much loss in statistical multiplexing gain. Video transmission can then be accomplished in a deterministic, real-time fashion without frame loss.

Plan for future work include: exploration into the use of more complex scheduling algorithms; storage and resource allocation and hierarchy management in parallel machine implementations of video on-demand servers.

References

[1] B. Maglaris and D. Anastassiou and P. Sen and G. Karlsson and J.D. Roberts. Performance models of statistical multiplexing in packet video communications. *IEEE Trans. Commun.*, Com-36(7):834–843, July 1988.

[2] B.J. Dempsey and J. Liebeherr and A.C. Weaver. A Delay-Sensitive Error Control Scheme for Continuous Media Communications. Technical Report CS-93-45, University of Virginia, October 1993.

[3] D. Heyman and A. Tabatabai and T.V. Lakshman. Statistical Analysis and Simulation Study of Video Teleconference Traffic in ATM Networks. In *IEEE Global Telecom. Conference*, December 1991.

[4] D.G. Morrison. Variable bit-rate video coding for asynchronous transfer mode networks. *British Telecom Tech. Journal*, 8(3):70–80, July 1990.

[5] E.W. Knightly and D.E. Wrege and J. Liebeherr and H. Zhang. Fundamental Limits and Tradeoffs of Providing Deterministic Guarantees to VBR Video Traffic. Technical Report temp, University of Virginia, 1994.

[6] G. Ramamurthy and B. Sengupta. Modeling and Analysis of a Variable Bit Rate Video Multiplexer. In *7th Specialist Seminar, International Teletraffic Congress*, Morristown, New Jersey, October 1990.

[7] H. Heeke. Statistical Multiplexing Gain for Variable Bit Rate Video Codecs in ATM networks. In *4th International Packet Video Workshop*, Tokyo, Japan, March 1991.

[8] H. Shino and et. al. Analysis of Multiplexing Characteristics of Variable Bit Rate Video Signals. In *Third International Workshop on Packet Video*, Morristown, New Jersey, March 1990.

[9] J. Hsieh and M. Liu and J.C.L. Liu and H.C. Du and T.M. Ruwart. Performance of a Mass Storage System for Video-On-Demand. *Journal of Parallel and Distributed Computing: Special Issue on Multimedia Processing and Technology*, page , 1995.

[10] J.M. del Rosario and G. Fox. Constant Bit Rate Network Transmission of Variable Bit Rate Continuous Media in Video-On-Demand Servers. Technical Report SCCS-677, Northeast Parallel Architectures Center (NPAC), Syracuse University, Dec. 1994. Submitted to: Second IASTED/ISMM International Conference on Distributed Multimedia Systems and Applications.

[11] J.Y. Hui. Resourse Allocation for Broadband Networks. *IEEE Journal on Selected Areas in Communications*, 6(9):1598–1608, December 1988.

[12] M. Nomura and T. Fujii and N. Ohta. Basic characteristics of variable bit rate video coding in ATM environment. *IEEE J. on Select. Areas Commun.*, SAC-7:752–760, June 1989.

[13] P. Pancha and M. El Zarki. A Look at the MPEG video coding standard for variable bit rate video transmission. In *IEEE INFOCOM '92*, Florence, Italy, May 1992.

[14] P. Pancha and M. El Zarki. Bandwidth Allocation Schemes for Variable Bit Rate MPEG Sources in ATM Networks. Technical report, University of Pennsylvania, 1992.

[15] P. Pancha and M. El Zarki. Prioritized transmission of VBR video traffic. In *IEEE GLOBECOM '92*, pages 1135–1139, Orlando, Florida, December 1992.

[16] P. Sen and B. Maglaris and N. Rikli and D. Anastassiou. Models for packet switching of variable bit-rate video sources. *IEEE J. on Select. Areas Commun.*, SAC-7:865–869, June 1989.

[17] R.M. Rodriguez-Dagnino and M.R.K Khansari and A. Leon-Garcia. Prediction of Bit Rate Sequences of Encoded Video Signals. *IEEE Journal on Selected Areas in Communications*, (), 1991.

[18] S.S. Lam and S. Chow and D.K.Y. Yau. An Algorithm for Lossless Smoothing of MPEG Video. In *ACM SIGCOMM '94*, August 1994.

[19] T. Ott and T.V. Lakshman and A. Tabatabai. A Scheme for Smoothing Delay-Sensitive Traffic Offered to ATM Networks. In *IEEE INFOCOM '92*, pages 776–785, 1992.

[20] W. Verbiest and L. Pinnoo and B. Voeten. The Impact of the ATM Concept on Video Coding. *IEEE J. on Select. Areas Commun.*, SAC-6(9), December 1988.

10.2 Appendix 2: Hybrid wavelet-H263 video compression

This Appendix contains a report on the application of wavelet compression to video coding.

NPAC Research Report
Hybrid wavelet-H.263 video compression

Tomasz Major
Northeast Parallel Architectures Center
111 College Pl., Syracuse, New York 13244
toma@npac.syr.edu

October 1996

In 1995, ITU-T established the H.263 draft standard to be used for very low rate video coding at less than 64kb/s. H.263 is based on the blocked discrete cosine transform (DCT) for intra-frame coding, and block motion compensation for inter-frame coding, thus continuing the traditional approach to compression strategies known from JPEG, MPEG-1, MPEG-2, and H.261 standards. At the same time, new compression schemes mature and are being considered by ISO-MPEG as possible compression strategies in the MPEG-4 standard to come. Among them, wavelet-based compression has recently received considerable attention.

This report describes the idea of applying discrete wavelet transform (DWT) in intra-frame coding within the overall H.263 scheme. The hybrid codec has been designed, implemented, and tested on CIF- and QCIF-sized sequences. The codec produces reconstructed frames of the quality superior to standard H.263 at the same bit rate. The main drawback of using wavelets is high computational cost. The method seems to be promising because of additional features of wavelet schemes, interesting from the point of view of scalability, robustness, watermarking.

1 Introduction

Digital video compression is one of the key issues in video coding, enabling efficient interchange and distribution of visual information. One of the most recent video coding standards, ITU-T H.263, is the state of the art in video coding for very low bit rates. However, today's networks do not provide a guaranteed quality of service or sufficient bandwidth. Many CPUs commonly

used today cannot perform decoding operations at the desired frame rate. Areas that still need improvement include:

- robust video transmission,
- scalable video coding,
- even lower bit rates.

Similar requirements have been successfully met by a family of wavelet-based technologies in several government and commercial applications [3]. We belive that wavelet-based schemes have potential to prove useful in video coding too. Thus, pointing into the future, we have chosen relatively immature wavelets to enhance H.263 recommendation. The following paragraphs provide a rationale for this choice.

Wavelet-based compression has a number of important advantages over other approaches. One of the most complex problems in block based techniques is that the energy in the blocks may differ after quantization, causing a quilted appearance in the regenerated image. This is quite a severe drawback because of the properties of the human visual system: the eye is constructed to detect very fine variations in luminance intensity. Quilting is especially detrimental in video applications because the eye is an integrator, any effect that may be corrected and effectively invisible on a frame-by-frame basis can prove to be severe overall.

With wavelets, there is a very gradual degradation of the compressed image quality as the compression rate is increased. The entire image is transformed and compressed as a single data object rather than block by block, allowing for an uniform distribution of compression error across the entire image and at all scales.

The extension of wavelet-based image compression to wavelet-based video compression may be done by changing the 2-D wavelet transform to a 3-D wavelet transform. The wavelet transform structure allows for scalable frame rate. There is no motion compensation required because of efficient representation of transient phenomena by wavelets. Higher compression can be attained by using a deeper temporal transform, but the pipeline delay is unacceptable in many applications. One major advantage of using a spatial transform is that in the same way as the still-image geometric relationships are preserved, the 3-D wavelet transform-based video compression cannot produce motion artifacts.

One report [1] describes the use of multiresolution motion estimation in wavelet domain but these results have not as yet been independently confirmed [21].

The wavelet transform structure that creates a multiscale representation allowing scalable access lends itself to efficient channel coding by allowing allo-

cation of error correction bits to critical scales of data.

Although the positive aspects of using wavelets in video compression are potentially many, the scope of the present experiment is constrained to the application of wavelet still image compression to intra-frame coding. We take advantage of existing software implementing H.263, where we merely change intra-frame coding from blocked DCT to DWT. This software is described in the section 5. Experiments and their discussion follows in the section 7. Sections 2- 4 give an introduction to the technologies used: first H.263 is outlined, then DCT-based and wavelet-based compression are described as they apply to this work.

2 The H.263 video coding scheme

The H.263 video coding standard [4] is a descendant of the motion-compensated DCT methodology prevalent in several existing standards such as H.261, MPEG-1 [22], and MPEG-2. H.263 focuses on the very low bit-rate (below 64kb/s).

The basic configuration of the H.263 video source coding algorithm is based on the ITU-T recommendation H.261 and is a hybrid of inter-frame prediction to utilize temporal redundancy, and transform coding of the remaining signal to reduce spatial redundancy. The source coder can operate on five standardized picture formats: sub-QCIF (128×96), QCIF (176×144), CIF (352×288), 4CIF (704×576) and 16CIF (1408×1152).

The codec has half-pixel motion compensation capability. Variable length coding (VLC) is used to encode motion vectors. In addition to the basic video source coding algorithm, four negotiable coding options are included for improved performance:

- Unrestricted Motion Vectors: motion vectors are allowed to point outside the picture. The edge pixels are used as prediction for the non existing pixels. With this mode, a significant gain is achieved if there is a movement across the edges of the picture, especially for the smaller picture formats.
- Syntax-based Arithmetic Coding: arithmetic coding is used instead of variable length coding. The reconstructed frames are the same, but fewer bits are produced. Routines for arithmetic coding are listed explicitly in the standard as well as constant models to be used with them in pertinent contexts. The routines are the same as those presented in the pioneering work of Witten, Neal, and Cleary [5].
- Advanced Prediction: overlapped block motion compensation is used for the luminance part of P-pictures. Four 8x8 vectors instead of one 16x16 vector are used for some of the macroblocks in the picture. The encoder

has to decide which type of vectors to use. This results in less blocking artifacts giving a subjective gain in quality.

- PB-frames: A PB-frame consist of two pictures being coded as one unit: one P-picture which is predicted from the last decoded P-picture and one B-picture which is predicted from both the last decoded P-picture and the P-picture currently being decoded. With this coding option, the frame rate can be increased considerably without increasing the bit rate much.

All these options can be used together or separately except for the Advanced Prediction mode which requires the Unrestricted Motion Vector mode to be used at the same time. The video bit rate is variable and no constraints on it are given. Error handling should be provided by external means.

The pictures are coded as luminance and two color difference components: Y, Cb and Cr. These components are defined in CCIR Recommendation 601. There is 4 times less samples of each chrominance component as compared to the luminance component.

Each frame is divided into groups of blocks, a group of blocks comprises of k*16 lines, where k depends on the picture format. Each group of blocks is divided into macroblocks. A macroblock relates to 16 pixels by 16 lines of Y and the spatially corresponding 8 pixels by 8 lines of Cb and Cr.

The prediction is inter-frame and may be augmented by motion compensation. The coding mode is called "inter" when prediction is applied or "intra" when no prediction is used. This can be determined at the frame level or at the macroblock level in predicted frames. Both horizontal and vertical components of the motion vectors have integer or half integer values restricted to the range [-16,15.5].

Encoded blocks are DCT-transformed and resulting coefficients are subsequently quantized (see section 3). Finally, the variable length coding or syntax arithmetic coding (SAC) is applied. ITU-T did not specify either a unique forward or a unique inverse DCT algorithm in its proposed standard.

H.263 is a very attractive video coding scheme as far as the picture quality and generated bit rate are concerned. The improvements in H.263 compared to H.261 are mainly obtained by improvements to the motion compensation scheme [6]. Still, new improvements are being tested [2]. The method remains limited by the underlying block-based compression approach.

3 DCT technology

Since DCT methods are well documented [14], we limit ourselves to a very brief review.

3.1 DCT Transform

At the input to the discrete cosine transform, source image samples are grouped into 8×8 blocks, shifted from unsigned to signed integers, and input to the forward DCT. The following equations are the idealized mathematical definitions of the 8×8 DCT:

$$F(u,v) = \frac{1}{4}C(u)C(v)\left[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y) * \cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}\right]$$

$$f(x,y) = \frac{1}{4}\left[\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}\right]$$

where: $C(u)$, $C(v) = \frac{1}{\sqrt{2}}$ for u,$v = 0$;
$C(u)$, $C(v) = 1$ otherwise.

The DCT is related to the Discrete Fourier Transform. Some simple intuition for DCT-based compression can be obtained by viewing the forward DCT as a harmonic analyzer and the inverse DCT as a harmonic synthesizer. Each 8x8 block of source image samples is effectively a 64-point discrete signal which is a function of the two spatial dimensions x and y. The forward DCT takes such a signal as its input and decomposes it into 64 orthogonal basis signals. The DCT coefficient values can thus be regarded as the relative amount of the two-dimensional spatial frequencies contained in the 64-point input signal. The coefficient with zero frequency in both dimensions is called the DC coefficient and the remaining 63 coefficients are called the AC coefficients.

Many different algorithms for computation of the forward and inverse DCT have been devised [15]. No single algorithm is optimal for all implementations.

3.2 Quantization and zig-zag sequence

In principle, the DCT introduces no loss to the source image samples; it merely transforms them to a domain in which they can be more efficiently encoded. This section describes the quantization of DCT coefficients in H.263 standard.

Because sample values vary slowly from point to point across an image, the forward DCT processing step lays the foundation for achieving data compression by concentrating most of the signal in the lower spatial frequencies. For a typical 8x8 sample block from a typical source image, most of the spatial frequencies have zero or near-zero amplitude and need not be encoded. After output from the DCT coefficients are uniformly quantized and ordered into the

"zig-zag" sequence. This ordering helps to facilitate entropy coding by placing low-frequency coefficients (which are more likely to be nonzero) before high-frequency coefficients. Then, nonzero coefficients and runs of zeros are encoded. Most common cases are variable-length coded.

4 Wavelet technology

4.1 DWT Transform

The discrete wavelet transform used in this work is identical to a hierarchical subband system, where the subbands are logarithmically spaced in frequency and represent an octave-band decomposition. To begin the decomposition, the image is divided into four subbands and critically subsampled. Each coefficient represents a spatial area corresponding to approximately 2x2 area of the original image. The low frequencies represent a bandwidth approximately corresponding to $0 < |\omega| < \frac{\pi}{2}$, whereas the high frequencies represent the band $\frac{\pi}{2} < |\omega| < \pi$. The four subbands arise from separable application of vertical and horizontal filters as shown in Fig. 1.

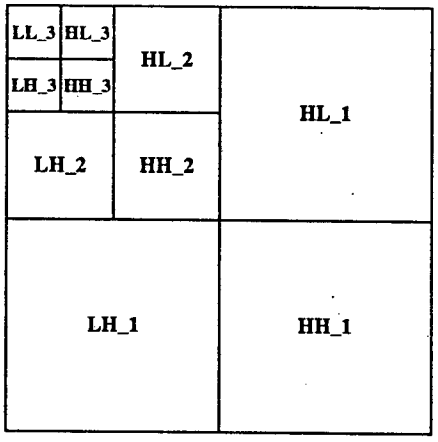


Figure 1: A three-scale wavelet decomposition

The subbands labeled LH_1, HL_1 and HH_1 represent the finest scale coefficients. To obtain the next coarser scale of wavelet coefficients, the subband LL_1 is further decomposed and critically sampled. The process continues until some final scale is reached. Note that for each coarser scale, the coefficients represents a larger spatial area of the image but a narrower band of frequencies. At each scale there are three subbands; the remaining lowest frequency subband

is a representation of the information at all coarser scales. The issues involved in the design of the filters have been discussed by many authors [19].

4.2 Zerotree quantization

Here we present the zerotree algorithm of Shapiro [9], which we use in this work. Nonetheless, other methods of quantization of wavelet coefficients have been devised [20].

In a hierarchical subband system, with the exception of highest frequency subbands, every coefficient at a given scale can be related to a set of coefficients at the next finer scale of similar orientation as shown on Fig. 2.

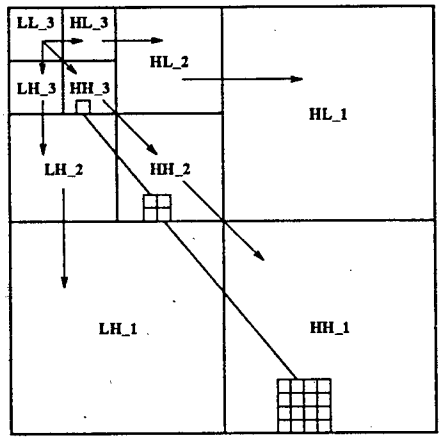


Figure 2: Parent-child dependencies of subbands

A wavelet coefficient x is said to be insignificant with respect to a given threshold T if $|x| < T$. The zerotree is based on the hypothesis that if a wavelet coefficient at a coarser scale is insignificant with respect to a given threshold T, than all coefficients of the same orientation in the same spatial location at finer scales are likely to be insignificant with respect to T.

A coefficient x is said to be an element of a zerotree for threshold T if it itself and all of its descendants are insignificant with respect to T. The positions of significants, assuming certain scanning order, can be represented in three-symbol alphabet:

- zerotree
- significant

- isolated zero

Significants are scanned in subsequent passes n with thresholds $T_n = \frac{T_{n-1}}{2}$. The values of coefficients are encoded bit by bit in so called refinement passes.

5 H.263 software

The source code of the H.263 codec was obtained from Buława [7], who improved the public domain software available from Lillevold [8]. The wavelet compression was implemented by the author. Integration of wavelet routines with H.263 code has been done by Buława and the author.

5.1 Lillevold's codec

Telenor R&D has contributed greatly to the methods and improvements used in H.263. Telenor's codec is fully compliant with H.263 Recommendation and has all negotiable coding options implemented.

Fast DCT routines follows "two dimensional" Chen-Wang algorithm [16]. They are implemented in 32-bit integer arithmetic (8 bit coefficients). IEEE 1180 reference (64-bit floating point, separable 8×1 direct matrix multiply) Inverse Discrete Cosine Transform may be optionally chosen.

Since the recommendation does not define a policy for determining whether a frame shall be inter- or intra-coded, only the first frame is intra-coded, every next picture can be encoded only as P or PB frame. This obviously reduces the generated bit rate but is very inconvenient when any data loss occurs during transmission in the network. Every P or PB frame is decoded on the basis of the previous frame and if any frame in the video sequence is lost, the rest of it can not be decoded properly.

The motion estimation is based on full search which is very CPU time consuming but gives a very good picture quality.

5.2 Buława's codec

The following enhancements were made by Buława [7]:

- interfaces to some useful routines have been created in order to facilitate future experiments;
- periodical intra-frames generation has been implemented. This allows for random access capability to the video sequence and recovery from lost or

corrupted part of stream while significantly increasing required bandwidth - about 40% when every 10th frame is intra-coded. The gap between two consecutive intra-frames in a video sequence is to be specified by a user;

- full motion vector search has been replaced (optionally) with logarithmic search in order to increase the encoding speed;
- the most CPU time consuming routines have been partially optimized.

Buława's coder runs on IRIX platforms. Encoding time has been reduced by more than factor of 3 in comparison with Lillevold's coder and varies from 3 to 15 frames per second for R10000/195MHz. Further improvements are possible by modifications of the source code, especially by elimination of time-consuming memory read/write operations, for and while loops optimization etc. [7].

The codec operates in the real-time domain which allows it to be deployed in collaboratory applications. This in turn, in conjunction with a very low bit rate and relatively good quality of encoded pictures, makes such a solution one of the most advanced in the current state of Web technology.

6 Wavelet software

Wavelet routines were written by the author and successfully used in stand-alone codec for still image compression [11].

6.1 Wavelet transform

The wavelet transform is performed separately in the vertical and horizontal direction, recursively on the low-pass subband of dyadic decomposition. The following filters are used:

- low-pass filter for decomposition:
  0.037829, -0.023849, -0.110624, 0.377403, (0.852699), 0.377403, -0.110624, -0.023849, 0.03782
- high-pass filter for decomposition:
  -0.064539, 0.040690, (0.418092), -0.788485, 0.418092, 0.040690, -0.064539
- low-pass filter for reconstruction:
  -0.064539, -0.040690, 0.418092, (0.788485), 0.418092, -0.040690, -0.064539
- high-pass filter for reconstruction:
  -0.037829, -0.023849, 0.110624, 0.377403, -0.852699, (0.377403), 0.110624, -0.023849,-0.037829

Bands edges are mirrored to allow the convolution of filter taps with extremal pixels. Since high pass decimation must be preceded by a one-sample delay, which means that the high- and low-pass kernels must be staggered with respect to each other by one pixel, the central filter taps are not always in the filter symmetry center. Hence, mirroring is done in different way for different filters:

- either towards the extreme pixels ;
- or towards the hypothetical image boundary just outside extreme pixels.

6.2 Quantization

Zerotree quantization based on the paper of Shapiro [9] is used. Some modifications to the original algorithm have been made:

- only one symbol is used for both positive and negative significant and sign is encoded separately;
- reordering of significants after each dominant pass does not improve the coder properties, and therefore this part of the original algorithm is not used;
- coding of the (n+1)'th dominant pass is done before of the coding of the n'th refinement pass

Zerotree quantization is embedded in the sense that the encoder can terminate encoding at any point thereby allowing a target rate to be met exactly. Also, given a bit stream, the decoder can cease decoding at any point in the bitstream and still produce exactly the same image that would have been encoded at the bit rate corresponding to the truncated bit stream.

6.3 Entropy coding

Lossless entropy coding is done by using an arithmetic coder similar to the one of Mark Nelson [10], but working much faster. The codec works with three statistical models: for zerotree symbols on the decomposition levels from 2 to the highest, for zerotree symbols on the 1st decomposition level, and for binary symbols increasing the coefficients accuracy. All of them are adaptive, 0-order models, initialized with 1s. Codec speed may be substantially improved if models are non-adaptive. This would require smart initialization based on training data, nonetheless coding efficiency would be reduced.

6.4 Color pictures compression

Color pictures are compressed as a concatenation of the three substreams: one for luminance and two for chrominances. The substreams are independent from each other and the spectral redundancy is exploited solely by the fact that chrominances are already a differential signal.

Proper balance between the amount of data for luminance and two chrominances is achieved through the notion of passes [17], inherent in the wavelet scheme. For a given image size, the same number of passes, rather than the compression ratio, results in similar quality of images. The number of passes for luminance and chrominances is a parameter to the wavelet codec.

Since the all of the information about color components are placed one after another this format is not embedded. The straightforward extension of this format may be easily implemented to accommodate for the embedded format by interleaving color components on a pass-by-pass basis.

7 Experiments

7.1 Testbed

Telenor's H.263 codec as improved by Bulawa has been further enhanced by the option of coding intra-frames using wavelets. The source code has been compiled in three versions:

- standard H.263 (**org**) - intra-frames encoded with DCT;
- standard H.263 with Syntax Arithmetic Coder option (**orgSAC**) - intra-frames encoded with DCT, coefficients coded with arithmetic coder;
- hybrid H.263 with wavelets (**wav**) - intra-frames encoded in wavelets.

The three experimental programs were tested on CIF and QCIF format images with intra-frame insertion every tenth frame. The codec which uses wavelet compression also has the capability of coding intra-frame with a specified quality defined in terms of the number of passes of the zerotree algorithm.

7.2 Luma-chroma balance

The human visual system is most sensitive to the luminace, which determines sharpness, shows artifacts and contours. Chrominance information, as compared to the the luminace information, can be significantly reduced without

noticeable loss of quality. We started experiments from determining the balance between luminace and chrominance components in wavelet compression. Subjective quality tests have been conducted in three categories, characterized by the difference of number of passes used for luminace and chrominance components: 1-, 2- and 3-pass difference. Resulting experiments will be denoted as **wavNM**, where N stands for number of passes for luminace, and M - number of passes for chrominances. The following experiments have been carried out:

- group 1: **wav87**, **wav76**, **wav65**;
- group 2: **wav86**, **wav75**, **wav64**;
- group 3: **wav85**, **wav74**, **wav63**.

It turned out that once a good subjective color rendition is obtained, further nominal increase in color quality does not improve the perception. Consequently, group 1 and 2 results in similar perceived quality while group 3 is unacceptable. Obviously, group 2 has been chosen for further experiments as it gives higher compression ratios then group 1. Two-pass difference distributes bytes among luma and chromas in proportions 80-84%:8-11%:7-10% on average for "akiyo" sequence.

7.3 Results for I-frames

The basic difficulty in carrying out tests within our testbed is the lack of scalability quality/compression ratio in **org** experiments. Although scalability mentioned in **wav** experiments is only stepwise, it allowed us to interpolate results in a very predictable manner as the wavelet technology we use scales very well [17] when applied to smarter stream formats.

Five experiments has been made in both CIF and QCIF format: **org**, **orgSAC**, **wav86**, **wav75**, **wav64**. The results are plotted on the Fig. 3.

It can be seen from comparison of **org** with **orgSAC** that using an arithmetic coder improves H.263 intra-frame coding efficiency by 11.3% for QCIF, and 11.9% for CIF. However with wavelets we achieve even higher compression and higher quality at the same time. Our wavelet-based codec outperforms the DCT based algorithm with SAC option engaged by 0.75dB for QCIF, whereas for CIF the difference is as large as 1.5dB. The price of this gain is high however. The interpolated decoding time for wavelet-based technology is about twice as large as the time of DCT based technology. Decoding times for MIPS R4400, 200MHz SGI workstation are given in the Table 7.3.

It must also be mentioned that H.263 may be used without arithmetic coding, at which time it operates 3 times faster with percent loss in compression lying

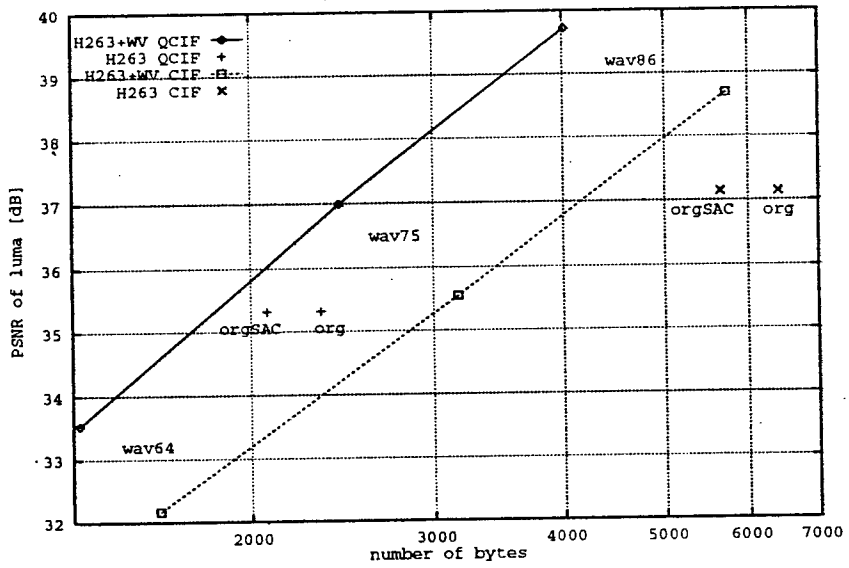


Figure 3: Average quality of the I-frame versus average number of bytes used to encode the I-frame for the sequence of "akiyo"

Table 1: Decoding time of I-frame on a 200MHz MIPS4400 SGI workstation

	org	orgSAC	wav86	wav75
QCIF	0.018	0.058	0.178	0.137
CIF	0.066	0.175	0.384	0.307

only in the teens. On the other hand, wavelets, as implemented for the purpose of this work, would lose about 50% in compression when deprived of the use of arithmetic coding. There are works [18] that claim the use of "analysis based coding" as a substitute for arithmetic coding without loss in quality/compression performance, but we do not know about the speed of these method.

7.4 Discussion on the frame size

In fact the CIF and QCIF sequences used in this experiment are very similar and do not differ much in the amount of detail but rather in their magnification. Accordingly, one would expect that they should be encoded with a similar number of bits. Results shows that wavelets are closer to this reckoning than DCT.

The explanation of the higher gain of wavelet methods over DCT methods for larger size images is the following: The DCT algorithm always operates on 8x8 blocks and subsequently overlooks larger scale correlations. DWT treats the image as a whole thus exploits these correlations. In CIF format, DWT performs one more decomposition than in QCIF and creates longer trees which can be better compressed by the zerotree algorithm. In other words, DCT uses 4 times more blocks to encode a magnified detail, while wavelets always use the same tree to describe a particular feature, independently of the magnification.

7.5 Quality of P-frames

In the experiments described in chapters 7.1- 7.4 we solely changed the compression algorithm of the I-frame. Now, we will describe the impact of the I-frame on the successive frames. We will also show how it is possible to controll P-frames' quality in order to balance the I- and P-frame qualities.

The figure 4 shows the average changes of quality in the succession of I- and P-frames. Each plot is made of the three sections:

- the I-frame;
- the P-frame immediately following the I-frame;
- all other P-frames.

It can be seen that for the original algorithm, **org[SAC]**, that the quality deteriorates after the I-frame. This fact is due to the replacement of the original liner search with the logarithmical search in Bulawa's codec [7].

In experiments in which wavelet-compression were used, the I-frames impact the quality of succesive frames: the better the quality of the I-frame, the better

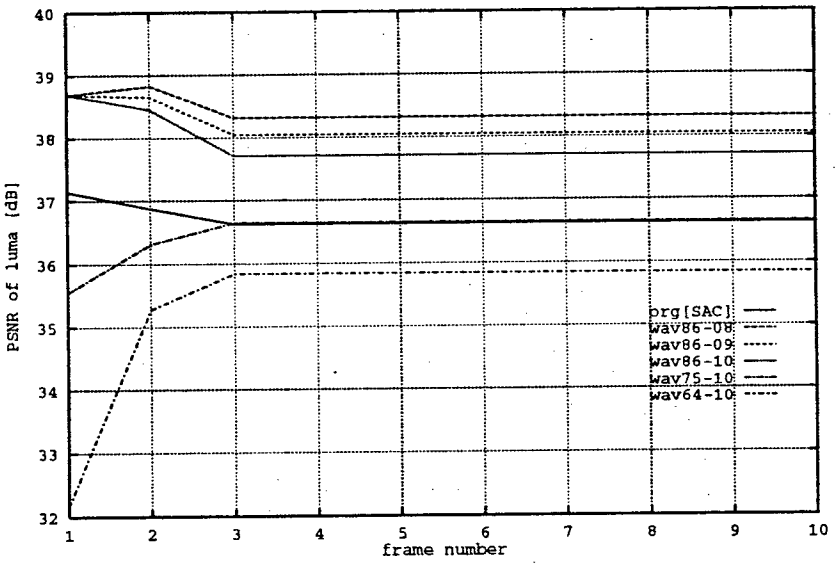


Figure 4: Average quality changes in IPP... succesion of 10 frames for the CIF sequence of "akiyo"

the quality of P-frames. Although **wavNM** encodes P-frames the same way as **org[SAC]** does, some blocks of P-frames are not encoded as they contain no motion. These blocks are replicated in successive P-frames with the quality of the I-frame. On the other hand, the blocks which are coded, are encoded with the quality with which they were also encoded in **org[SAC]**. Thus the quality of P-frames in **wavNM** experiments tends to converge to the level of the quality of P-frame from the **org[SAC]** experiment.

To reduce this tendency of P-frames, which is negative for I-frames compressed with better quality than I-frames in **org[SAC]** experiment, we change the quantization step of AC-coefficients in P-frames. The results for three quantization steps, equals to 8, 9, and 10, are shown in the picture 4. The corresponding experiments are denoted as **wav86-08**, **wav86-09** and **wav86-10** respectively. The experiments previously referred to as **wavNM** are eqivalent to **wavNM-10**.

Lowered quantization step reduces compression of P-frames significantly: from 20% for **wav86-09** to 38% for **wav86-08**. However, a P-frames itself uses as little as 5-10% of bytes needed to encode an I-frame, thus additional bytes for P-frames do not add relatively much to the total stream size. Exact number of bytes are given in the table 7.5.

Table 2: Average number of bytes used to encode I- and P-frame, and total number of bytes for 300-frames CIF sequence of "akiyo"

Experiment	org	orgSAC	wav86-10	wav86-09	wav86-08
I-frame	6409	5648	5719	5719	5719
P-frame	314	315	308	371	425
proportion [%]	4.9	9.1	5.4	6.5	7.4
total	274646	252358	254985	273475	290146

In **wav86-XX** experiments, for CIF-sized sequence of "akiyo", the quality of P-frames is worse than the quality of I-frames. For this, and for P-frames are more numerous than I-frames, the quality of P-frames can be safely thought as the overall quality of the sequence. Figure 5, therafter, shows the superiority of the **wav86-XX** codec over the **org[SAC]** codec in terms of quality/compression efficiency.

8 Conclusion

We decided to integrate H.263 encoder with wavelet compression scheme to obtain the very first hybrid video encoder. This novel approach marries the tra-

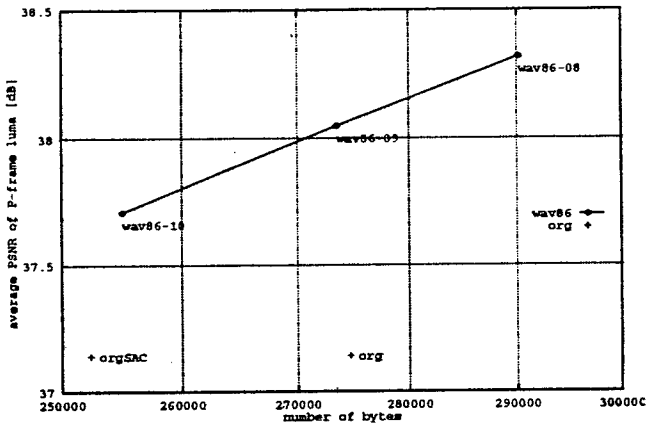


Figure 5: Average P-frame quality versus total number of encoded bytes for the 300-frames CIF sequence of "akiyo"

ditional approach with relatively immature technology. However, our solution increases quality and reduces bit rate significantly, especially for bigger frame size, where the gain is 1.5dB. The drawback of wavelet compression is its speed. Decoding time is prohibitive for real-time operation on most systems, though it may be acceptable on the fastest ones. This problem may be soon overcome because algorithms used for wavelet compression are getting faster due to enormous research effort all over the world. The advantages of using wavelet schemes, as proven by this work, lays a foundation for further experimentation with scalable video codecs, and possibly 3-D compression.

9 Acknowledgements

We are grateful to professor Czesław Jędrzejek for the idea of the hybrid codec. Janusz Buława is thanked for his pioneering implementation.

The research work presented in this document was funded by US Department of Defense grant, Rome Lab. Contract No. F30602-95-C-0273, PR No. C-5-2293/4.

References

[1] Sohail Zafar, Ya-Qin Zhang, Bijan Jabbari: *"Block-Classified Bidirectional Motion Compensation Scheme for Wavelet Decomposed Digital Video"*

[2] G. Bjontegaard, F. Azadegan, T. K. Tan, Gwang-hoon Park, Atul Puri: *"Core experiment on improved intra coding"*

[3] *"Gray Scale Fingerprints Image Compression Status Report"*, FBI Systems Technology Unit, Nov. 1991

[4] ITU-T Recommendation H.263: *"Video Coding for Low Bitrate Communication"* (Draft), December 1995

[5] Ian H. Witten, Radford M. Neal, John G. Cleary: *"Arithmetic coding for data compression"*, Communications of the ACM, June 1987, vol.30, no.6

[6] Bernd Girod, Khaled Ben Younes, Reinhard Bernstein, Peter Eisert, Niko Färber, Frank Hartung, Uwe Horn, Eckehard Steinbach, Klaus Stuhlmüller, Thomas Wiegand: *"Recent advances in video compression"*, to appear in proceedings ISCAS-96

[7] Janusz Buława: *"Integration of multimedia colaboratory environment with Web browser"* (Master Thesis), Sep 96

[8] Karl Olav Lillevold: *"TMN"* software (H.263 codec), http://www.nta.no/-brukere/DVC/

[9] Jerome M. Shapiro: *"Embedded image coding using zerotrees of wavelet coefficients"*, IEEE Transactions on Signal Processing, vol.41, no.12, Dec. 1993, pp.3445-3462

[10] Mark Nelson: *"The Data Compression Book"*, M&T Books, San Mateo, CA 94402

[11] Tomasz Major: *"Wavelet project description"*, http://deneb.npac.syr.edu:-32831/overview.html

[12] King N. Ngan, Douglas Chai, Andrew Millin: *"Very Low Bit Rate Video Coding Using H.263 Coder"*, IEEE Trans. on Circuits and Systems for Video Technology, Vol.6, no.3, June 1996

[13] Steven McCanne, Van Jacobsonvic: *"vic: A Flexible Framework for Packet Video"*, Lawrence Berkeley Laboratory

[14] Gregory K. Wallace: *"The JPEG Still Picture Compression Standard"*, submitted in Dec. 91 for publication in IEEE Transactions on Consumer Electronics

[15] K. R. Rao, P. Yip: *"Discrete Cosine Transform - Algorithms, Advantages, Applications"*, Academic Press, Inc. London, 1990

[16] Chen-Wang: (inverse two-dimensional DCT algorithm), cf. IEEE ASSP-32, pp. 803-816, Aug. 1984

[17] Tomasz Major: *"Passes"*, http://deneb.npac.syr.edu:32831/research/-passes/pass.[ps|gif]

[18] V. Ralph Algazi, Robert R. Estes, Jr.: *"Analysis based coding of image transform and subband coefficients"*, CIPIC, UC, Davis

[19] E. H. Adelson, E. Simoncelli, R. Hingorani: *"Orthogonal pyramid transform for image coding"*, SPIE vol.845 Visual Communications and Image Processing II (1987), pp.5-58

[20] Amir Said, William A. Pearlman: *"A new fast and efficient image codec based on set partitioning in hierarchical trees"*, IEEE Transaction on Circuits and Systems for Video Technology, vol.6, no.3, June 1996

[21] Herve Murret-Labarthe: (internship), EFP, Poznan, PL

[22] ISO/IEC MPEG, International standard ISO/IEC 11172: *"Information technology - Coding of mowing pictures and associated audio for digital storage media at up to about 1,5 Mbit/s"*, Dec. 1991

10.3 Appendix 3: A continuous media file system for multi-resolution data

10.3.1 Introduction

Currently, video server design is regarded as a system integration issue where diverse components are haphazardly plugged together. In reality, video servers are complex entities that must address the requirements of individual video streams as well as system-wide requirements. The storage subsystem constitutes a major component of any media server. Yet, today, file system support for continuous media is very limited.

The traditional approach to Quality of Service issues is network-centric. Adaptive applications (i.e., applications that change their behavior in response to changing network conditions) are being proposed. We observe, however, that the potential benefit of adaptivity is limited without architectural changes in the media. Consider, for instance, a video server that is capable of adjusting the bitrate of outgoing media streams by sending only a part of the video stream encoded by a multi-resolution coder. Using current file system architectures, the server will have to read the entire stream from the disk before discarding the information that is not being sent. In this section we present an idea of the video server architecture which provides end-to-end support for such an adaptive application. The proposed Continuous File System (CFS) also provides explicit support for isochronous data.

10.3.2 Traditional File Systems

Traditional file systems have not been designed to support continuous data streams. In the media server context, standard file systems suffer from the following problems:

- There is no support for isochronous data. Continuos media requires support for disk scheduling so that data is read in time. Thus there are hard limits posed on delivery of data.
- There is no support for varying quality of services. In a heavily loaded file server, all the streams being served see lower throughput and higher response times. For continuous media this may lead to annoying jitter at the client. Support for admission control is needed to avoid this.
- Continuos media places strenuous demands on the file system for data movement. MPEG-2 movies require on average 6 Mbits/sec per stream for commercial quality video. For a 1000 stream video server this translates to 750 Mbytes/sec. Traditional file servers have throughput well under 10 MBytes/sec.

10.3.3 The CFS File System

File systems with support for continuous media will be at the heart of the next generation of video servers for Interactive TV. A video server primarily consists of three components: a control component that responds to client requests, a communication component that moves data across the network connection between the client and the server, and a file system component that manages the storage and retrieval of data from blocks. CFS provides for an Applications Programming Interface (API) whereby any application can interact with the underlying CFS file system.

10.3.4 Terminology

Data are transferred from the disk in units called transfer blocks. The file system needs to be configured during setup for a given transfer block size. The size of the transfer block is dependent on the kind of files being served from the disk array. The transfer block size can be used to tune the file system for a given stream characteristic. There is, however, no limitation on streams with varying characteristics residing within the file system. If the file system is serving high-rate streams such as MPEG-2, then the transfer block size can be coarser compared to a lower rate stream, such H.263.

We define *retrieval block* as a logical data unit which is provided to the application on a periodic basis. Thus for video, this could be 1 second worth of video data. For a retrieval block to be fetched and delivered to the application, multiple transfer blocks might need to be read from the disk array below. This is managed by the file system and is transparent to the application.

We define *data layout* as the placement of transfer blocks on a disk array i.e., the methodology used to write a block of data onto a physical disk or disk array. Data layout deals with the methodology for writing a given retrieval block onto the disk array.

We further define *data placement* as the placement of multiple retrieval blocks onto the disk array. The data placement policy encapsulates the data layout policy for each block.

10.3.5 Multi-resolution Support

The file system supports multi-resolution data. Each stream can consist of any number of user defined resolutions. Resolutions are specified by mapping to corresponding bit rates, i.e., a higher resolution will map to a higher bit rate. The owner of the stream decides the rate and the resolution allocation on a per retrieval block or stream basis. For resolution allocation on a stream basis, all retrieval blocks within stream consist of resolution for specified bit rates.

The file system provides adaptivity of resolution allocation at retrieval block level. For variable bit rate coding schemes such as H.263 and MPEG-2, the rate from the encoder varies significantly over time. Scenes with relatively little motion can be represented with fewer bits without degrading

the picture quality. This is converse for scenes with large motion which require more bits to be represented. Wavelet video coders are inherently multi-resolution. Video coding applications can use the adaptivity at the retrieval block layer to map fewer resolutions to lower bit rate while using more resolutions for higher bit rates.

10.3.6 Data Layout Policy

The problem of data layout can be defined as follows: Given a block of data, what is the optimal way to write it onto the disk array? Much research has gone into data layout policy in general. Keeton and Katz have evaluated data layout policies on disk arrays for multi-resolution data. The data layout policy on the disk array determines the transfer block size from the disks. RAID involves striping a data block across multiple physical disks to improve the overall throughput of the disk subsystem. Hence, it has been argued that for continuos media data a similar approach should be taken to maximize the throughput from the disk subsystem.

For continuous media data such as video, faster throughput from the disk subsystem translates to faster response time from the file system. However, data is eventually played out to the screen in a deterministic manner. Thus, for media such as video, it is not the throughput but the overall number of supported streams that matters. For a RAID subsystem, retrieval blocks can be striped across multiple disks or stored on a single disk. Each retrieval block consists of all the user defined resolutions. 1Disk and the 3Disk striping by Keeton and Katz are illustrated in the following figures. They simulated various client request loads and retrieval block sizes. Performance for the 1Disk case was better for heavy loads and large retrieval block sizes. CFS implements the 1Disk striping methodology.

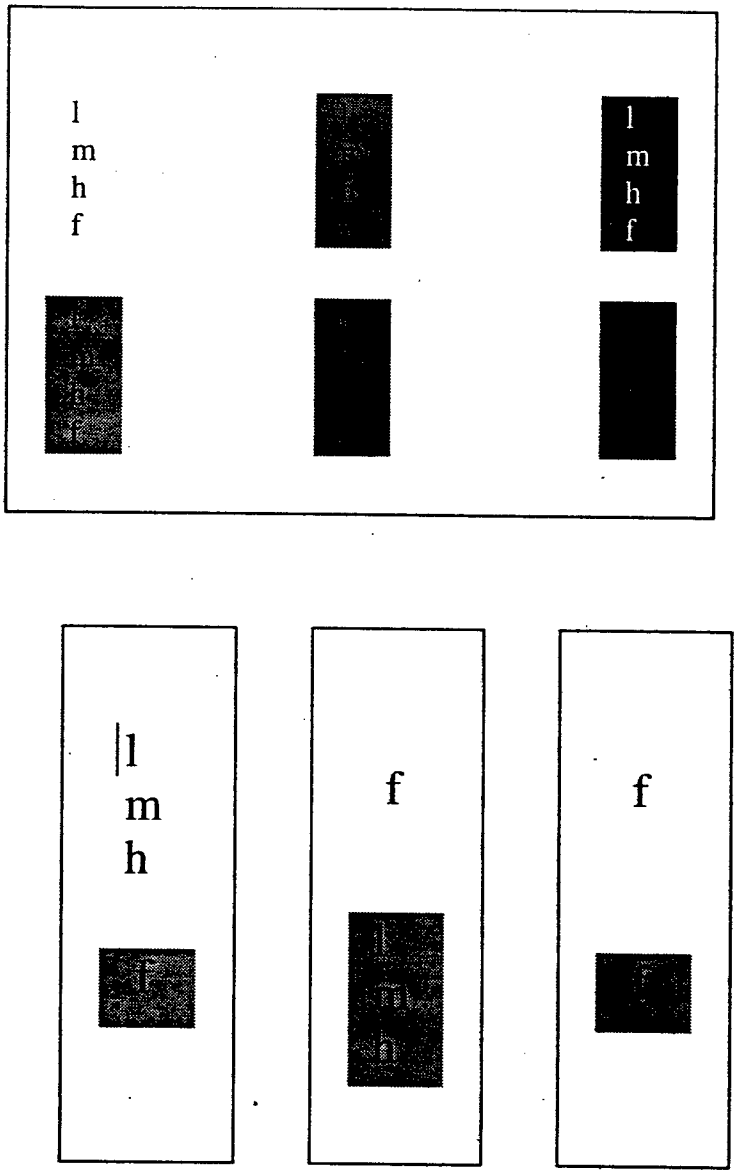


Figure 10.3.1. 1 disk striping (top) and 3 disk striping (bottom)

We define Parallelism P, as the number of disks taking part to fulfill a given client request, and Concurrency C, as the number of requests which can be serviced at a given time. Then for the 3Disk case we have a parallelism of 3 and concurrency of 1 as only a single request can be serviced at a time. Similarly, for the 1Disk case, the parallelism is 1 and concurrency 3. A higher concurrency value is needed to support the maximum number of stream from the disk subsystem.

10.3.7 Data Placement Policy

The data placement policy is based on the XFS file system from Silicon Graphics. In XFS, data is read/written to the disk in retrieval block size. Since CFS provides for variable rate data, the size of the retrieval block can vary over time for a given stream. In the XFS file system for the VoD mode, retrieval blocks for a given stream are placed in a round robin manner onto the physical disks in the array. The size of retrieval blocks on the XFS are constant for all streams on the VoD partition. The retrieval block size is decided when the partition is created. We feel that this is a limiting aspect to the XFS scheme. For variable rate data (such as MPEG-2) this poses problems for streams exceeding the retrieval block size. Furthermore, this scheme leads to a great deal of disk fragmentation where multiple bitrate streams exist on the same partition.

CFS reduces the disk fragmentation by splitting the retrieval block into multiple transfer blocks. The transfer blocks do not need to be arranged continuously on the disk. The transfer block size is defined when the CFS partition is set up. It can be dynamically configured to optimize for various bit rate streams being stored on the partition.

Retrieval blocks are placed in a cyclical manner on the disk array. A given disk will contain the data wholly for a retrieval block. This placement policy lets us map the scheduling policy for retrieval block fetches to the disk array. In addition, the cyclical placement algorithm provides an implicit load balancing between multiple disks. This however can not be guaranteed for variable rate data. For constant rate data, this leads to a well load-balanced disk subsystem.

10.3.8 The Directory Service

Metadata is stored for each stream existing on the CFS partition. The metadata can exist on the CFS partition, or could exist on a conventional partition such as NFS or NTFS. The metadata structure for each stream is shown in the figure below.

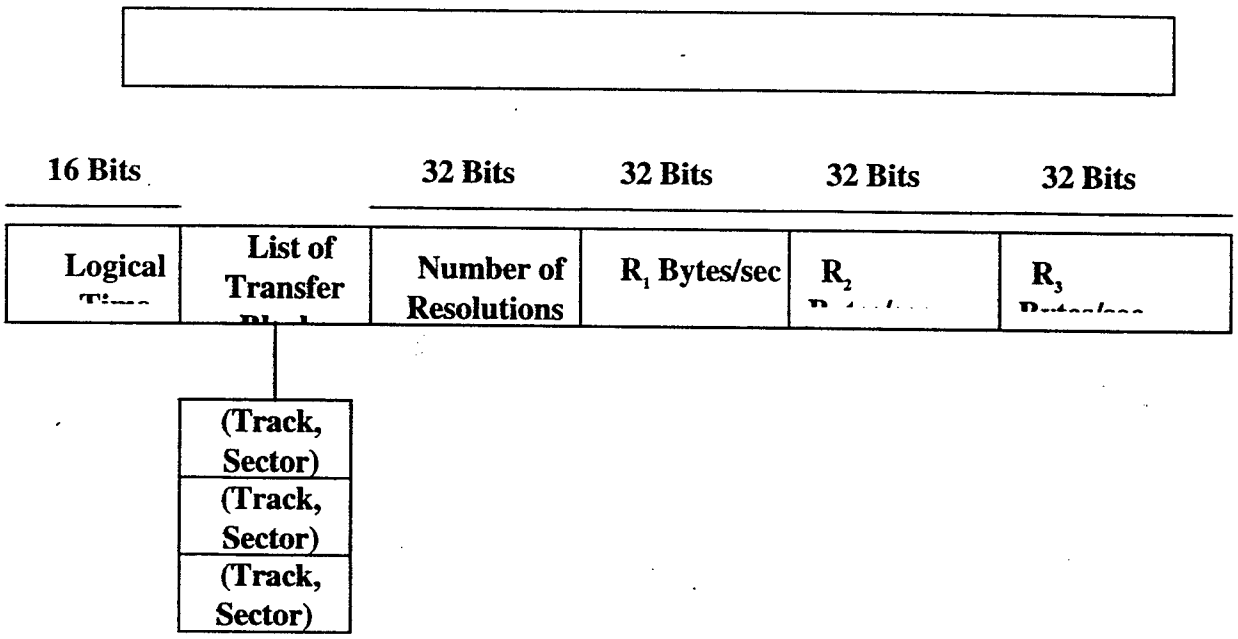


Figure 9.3.2. Metadata entry for a stream

Each metadata consists of the stream header. The header consists of core information such as the stream name, the access rights associated with the stream, the owner of the stream and the creation date. Following the header is a chunk of data associated with each retrieval time unit for the data called metachunks. For video this could be for every second of data. There are variable number of metachunks for each stream depending on the logical length of the stream. Each metachunk consists of the logical time unit this metachunk represents. It contains the number of resolutions of data for the current logical time along with the bit rates for each of the resolutions. In addition, a list of transfer blocks is maintained in a list. The transfer blocks are stored in the list in (track, sector) pairs. The transfer blocks are ordered such that sequentially reading the blocks will reconstruct the data. Metadata allows us to deterministically schedule the access to transfer blocks. Metadata is extensively used by the admission control policy described later to guarantee a certain resolution or data rate from the disk subsystem.

In addition to managing the metadata structure, the CFS file system exposes the Directory API. This API allows applications to view the streams existing on the system along with their access rights, creation date and owner. It also provides the ability to query the average bit rate for a given

resolution (since bitrates may vary over time per resolution). Additional information can be queried as the metadata stored for each stream is extensive.

10.3.9 Admission Control Policy

An admission policy is needed to determine if a new stream can be serviced at the required QoS parameters without affecting the service of other streams. CFS uses a benchmarking scheme to approximate the throughput for each of the disks in the disk array. The benchmarking scheme calculates the minimum number of transfer blocks which can be read from a given disk. This is used as a conservative limit to guarantee that we can deliver the requested throughput for all the streams.

A logical scheduler is used to schedule transfer of data at retrieval block level. The admission control is implemented at this level. For each of the disks in the array we define Min_i as

Min_i = *The minimum number of transfer blocks disk i can read.*

For each of the disks in the array, we need to construct an access graph. The access graph signifies the total number of transfer blocks read for a disk at a given time.

The access graph is constructed using the function $D_{t,\ i}$ for each of the disks in the array by varying the time t. The access graph with the overload function is shown in the figure below.

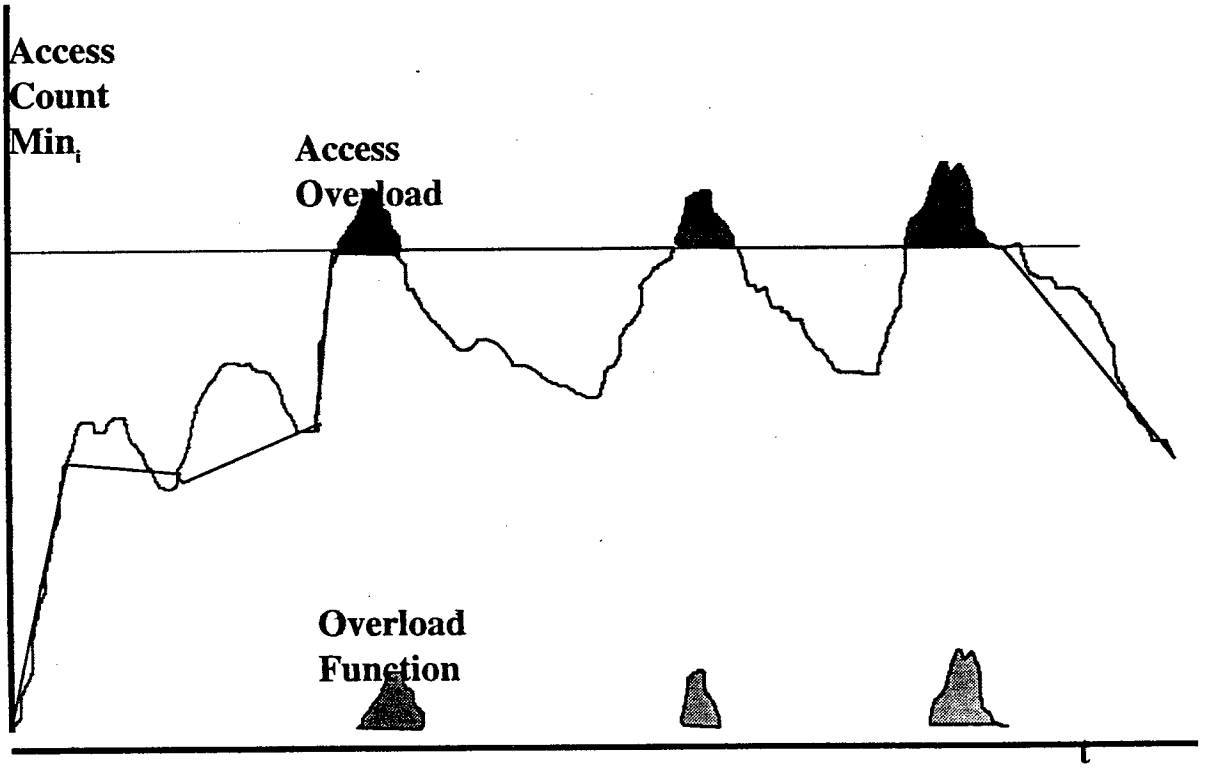


Fig. 9.3.3 Access Graph and Overload Function

The simplest admission criteria for admission of a new stream can be to look at the overload function assuming the stream is admitted into the system at a given resolution. A stream is denied admission if

$$OV = \sum_{i=0}^{i=S} \sum_{im=0}^{N} O(t,i) \neq 0$$

Here we assume that if the access function $D_{t,i}$ exceeds the minimum number of transfer blocks a disk can read we reject the request. For constant rate data, this scheme provides a reasonably good admission control policy. However, for variable rate data, it is likely that the access load on the disks will be in bursts with sudden demand for data. Hence the access graph will be peaky. This approach can unnecessarily reject a stream in this scenario where we exceed the threshold for a short duration. However this approach provides a simple and elegant solution for admission control. In order to maximize the number of streams the disk array can support, a more complex policy needs to be defined.

We define a new admission policy based on prefetching of transfer blocks at times when the access load is high. Blocks are prefetched when the access load on the disk is lower. We can thus aim to maximize the throughput from the disk subsystem by optimizing the fetch patterns of the blocks.

Prefetching of transfer blocks is done via an iterative algorithm. The number of blocks to be prefetched for time *t* are given by the overload function at time *t*. We maintain an access list at time *t*. The access list is a set consisting of the tuple

$AL_{t,i}$ = *(M, t, Start Block, End Block, Current Req.).*

The start block and the end block are extracted from the metadata. The current request parameter indicates whether the block fetches are for the current scheduling cycle or are being prefetched for later. The algorithm tries to minimize the prefetch interval for a stream. Blocks which are marked to be prefetched at time t are removed from the access list and added to a prefetch list at time *t-1*. The prefetch list consists of the tuple

$PL_{t-1,i}$ = *(M, t, Start Block, End Block).*

The prefetch list is added to the access list at time *t-1* marking the current request entry as prefetch. Blocks marked for access as current request at time *t* are given priority to move to the prefetch list at time *t-1*. All the transfer blocks can be moved for a given stream from the access list to the prefetch list or only some. The access list at time *t* is updated accordingly. This leads to the access lists being modified for *t* and *t-1*. The overload function is recalculated for the new stream with the new access lists. The overload function for time t, should be 0 as *O(t, i)* blocks are

moved from $AL_{t\,i}$ to $PL_{t-1,\,i}$. This iterative algorithm proceeds till the start of the stream is reached. The start of the stream is given by R_{im}. The admission policy for the new stream then is to admit if

or all i, we have $O(R_{im}, i) = 0$

Thus, the stream is admitted if the overload function is 0 at the start of stream playback. This scheme is more complex and intricate to interpret. However, the benefits over the simpler approach should be significant..

10.3.10 The Physical Scheduler

Once a stream has been admitted into the system, the physical scheduler uses the metadata for each stream to transfer blocks from the disk in an ordered manner. Many of the SCSI drivers in the market already optimize disk head movement. The physical scheduler is responsible for constructing the retrieval block from multiple transfer blocks. Hence it maintains a list of blocks currently read into a buffer. This problem is compounded with the transfer blocks being prefetched for some stream well before they are needed. The overall architecture of the CFS is shown below

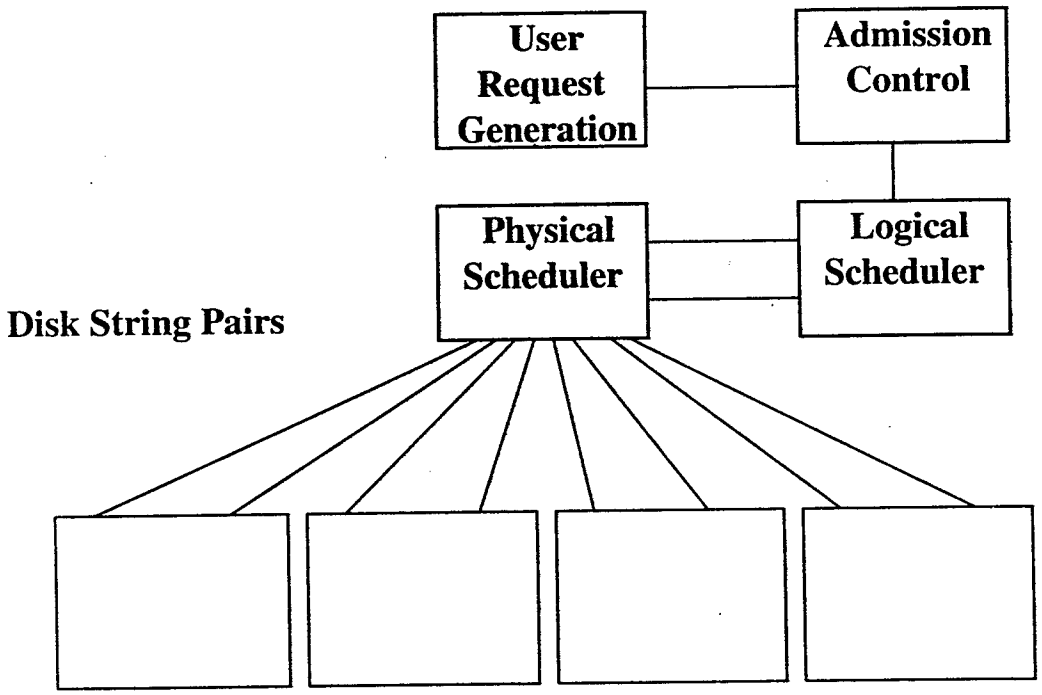


Fig. 9.3.4. Underlying architecture of the CFS

10.3.11 Interactive Support for Continuous Media

It is clear from the design that CFS is a viable design for non-interactive streaming of continuous media. Recently there has been a spurt of such applications which utilize non-interactive streaming data on the WWW. However, CFS can implement rudimentary interactivity. If the application wishes to stop reading data from the disk, the stream entry can be removed by the logical scheduler. This will result in the transfer blocks not being fetched by the physical scheduler. Once the stream resumes playback, CFS will attempt to admit the stream as a new stream. On success, the stream can continue as before. If the system has become loaded and admission is denied, the admission policy can iterate to determine the earliest time the stream can begin playback. Hence there is no guarantee that the stream can be reinstated but a best effort is made.

10.3.12 Quality of Service (QoS) Support

CFS provides for multiple quality of services. An application can dynamically request a newer quality of service from the disk subsystem but the request may be denied. Varying levels of QoS can be specified via using the resolution index of the stream i.e. Specify a given resolution to be played at. This results in variable amount of data being fetched per scheduling cycle if the data is variable over time.

An application may also specify a constant bit rate from the disk subsystem. This may lead to multiple resolution levels being fetched over time for the stream due to the variable nature of the data. A request for lowering the resolution to be served, or lowering the bit rate will always be accepted. The access graph needs to be recalculated for all the disks in the array. For a higher resolution request or increase in bit rates, the request may be denied and the stream can continue to be serviced at current QoS parameters.

10.3.13 Comparison of CFS and XFS

CFS is based on the XFS file system from SGI. However in many respects it aims to and does alleviate many of the shortcomings of the XFS file system. XFS provides for a constant retrieval block size. This is a constraint as one needs to know before hand the stream characteristics to be stored on the XFS partition. This is not a constraint with the CFS file system as it allows for dynamic retrieval block sizes. Hence varying bit rate streams can coexist on the same partition.

CFS explicitly supports multi-resolution data sets. The XFS was designed for a single resolution data set. Single resolution data set can be visualized as a multi-resolution data sets with a single resolution. Hence standard MPEG1 streams can be supported by CFS. It is envisioned in the near future that multi-resolution compression schemes for video will dominate over convention DCT based schemes as MPEG. Hence CFS is geared for the change.

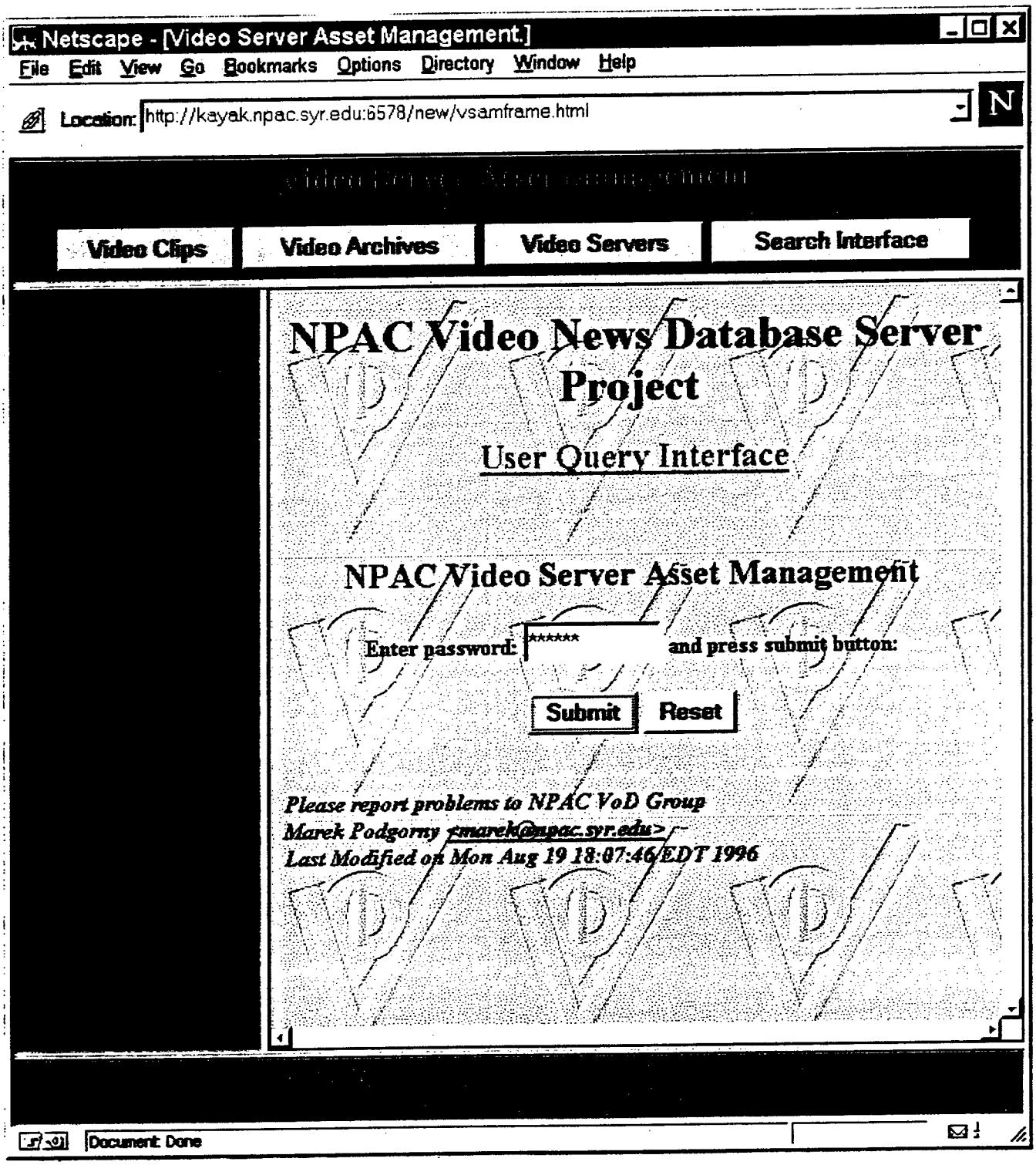
Increasingly with the integration of networking into application, the notion of varying QoS is growing. CFS allows for dynamic QoS support for a stream. Applications can use this to dynamically match the available bandwidth on the network. XFS does not support QoS parameters.

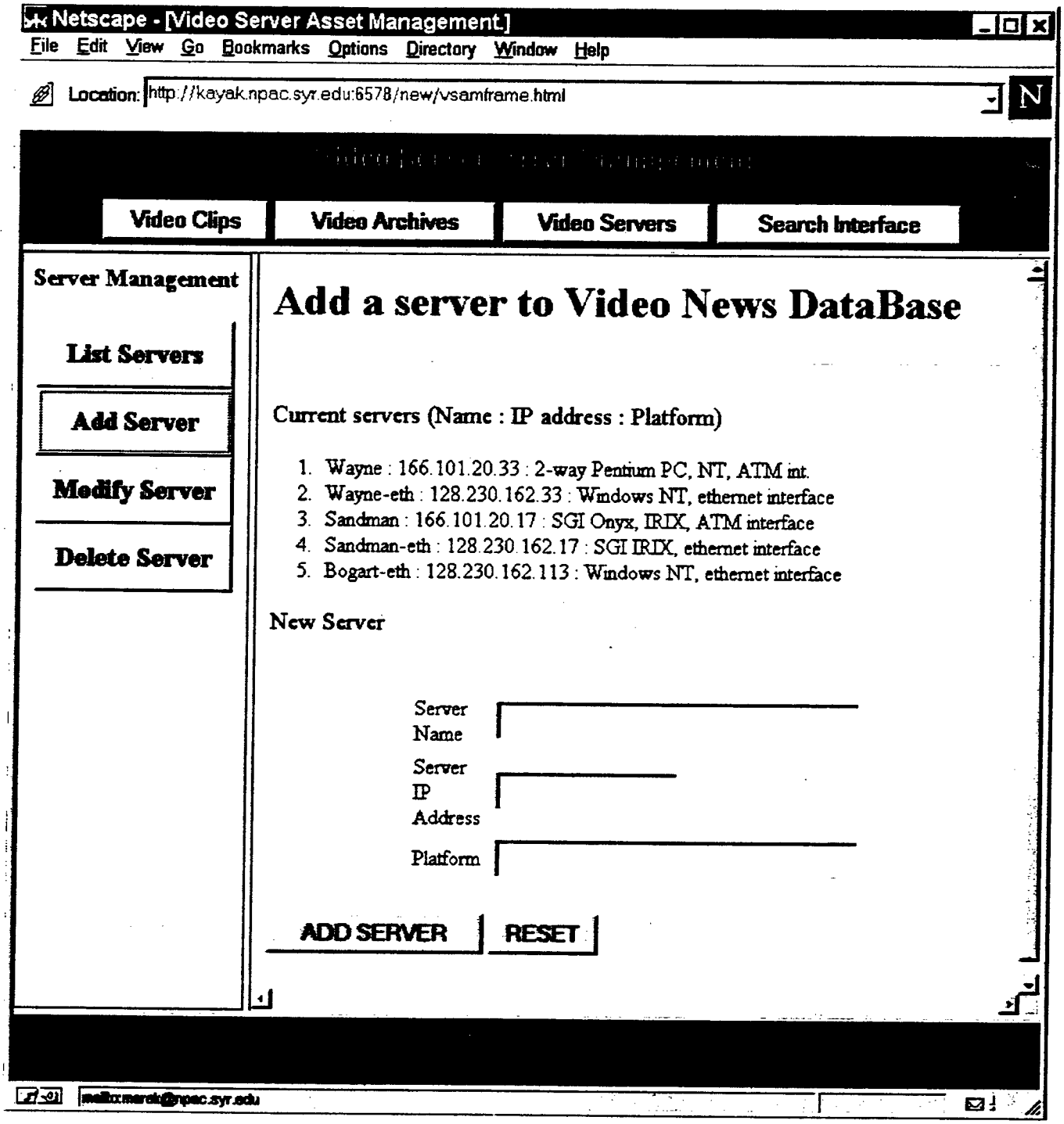
10.3.14 Conclusion

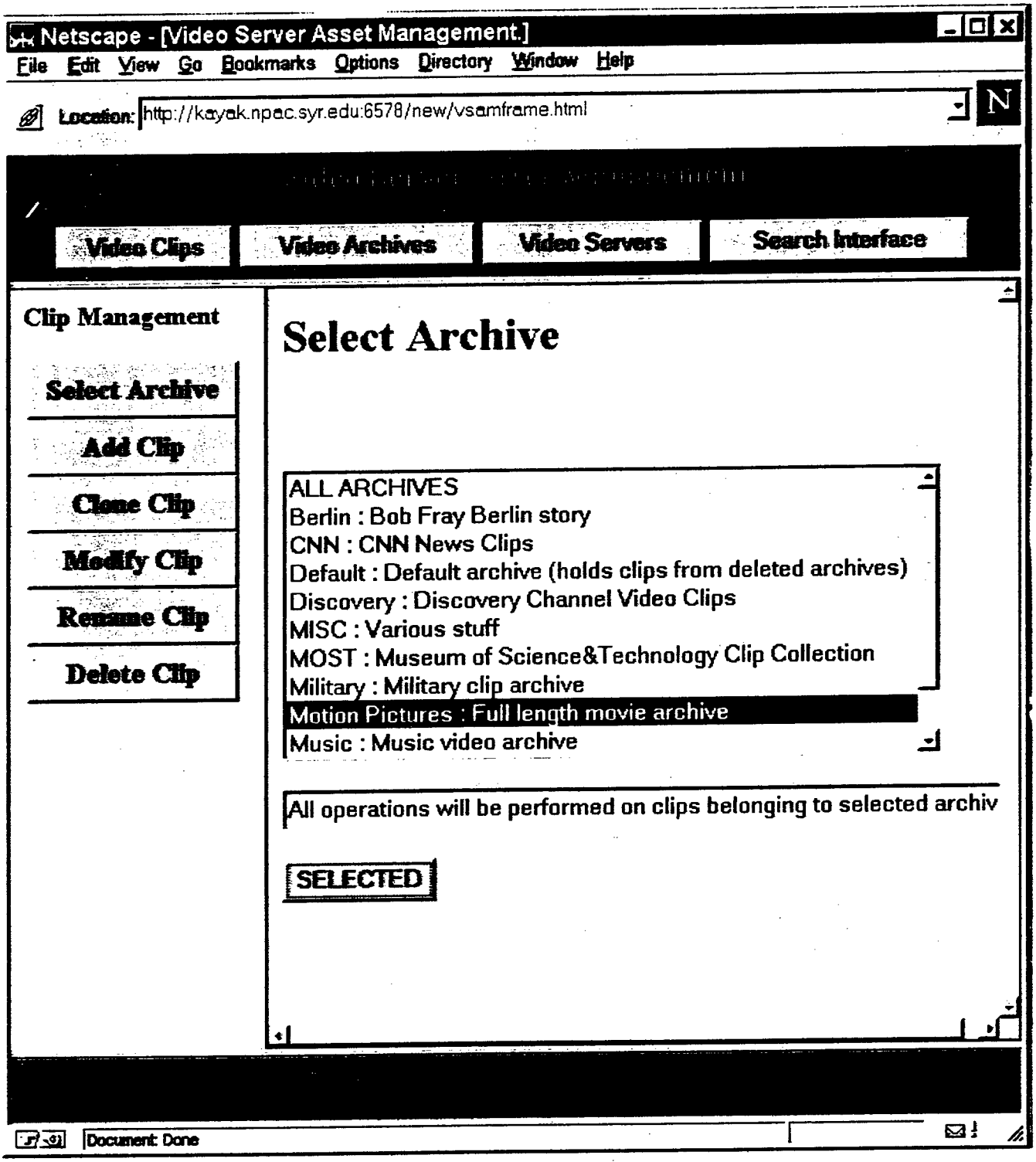
In this section, we have described a novel file system which provides explicit support for continuos media such as video. CFS design tries to address a host of problems conventional file systems such as NFS face. Its dynamic support for QoS, along with hierarchical data storage support and guaranteed rate QoS should make it useful for applications dealing with isochronous data with hard deadlines.

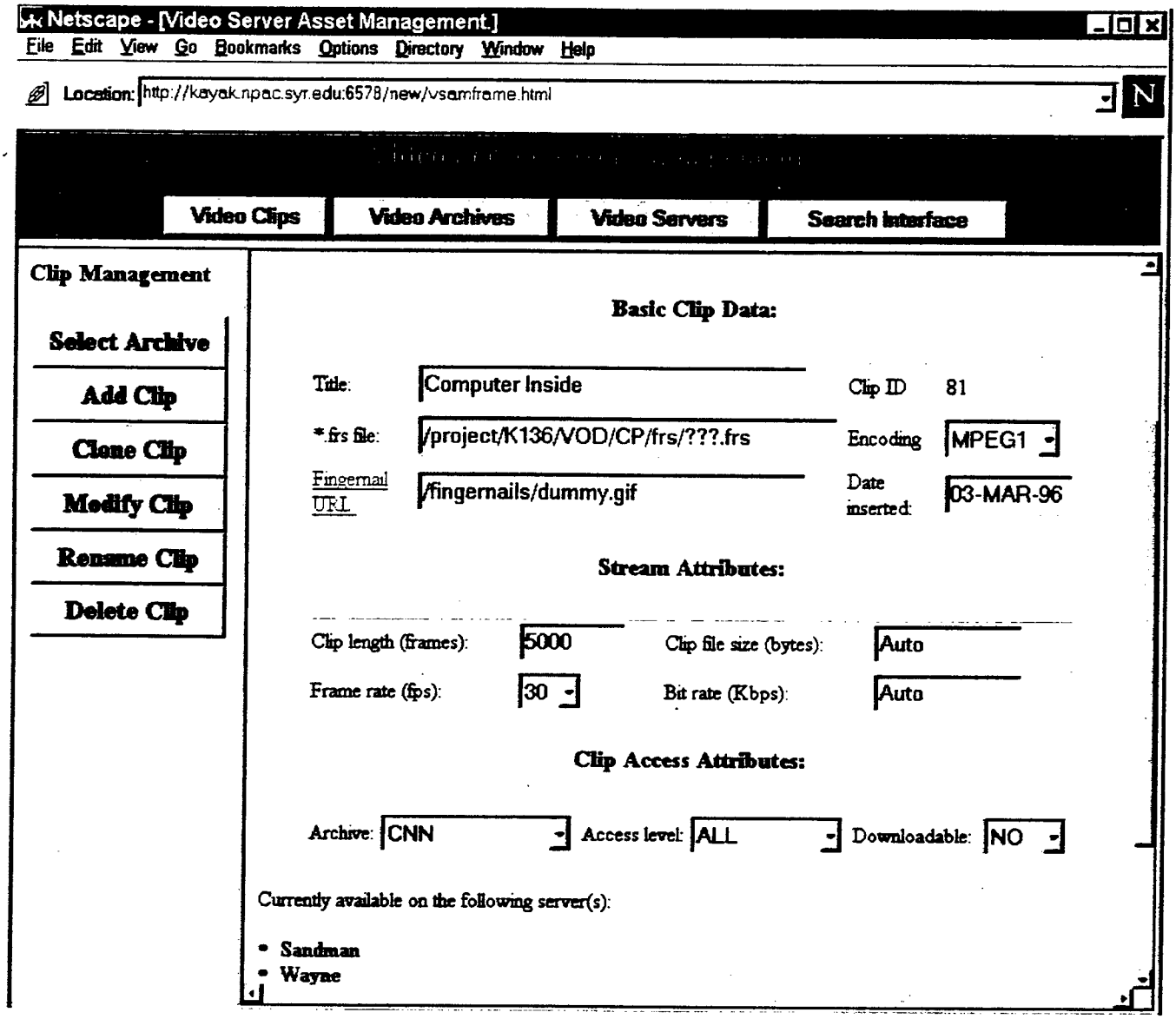
10.4 Appendix 4: Video Server Asset Management Screen Dumps

This Appendix contains screen dumps of the Video Server Asset Management interface.


Netscape - [Video Server Asset Management]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://kayak.npac.syr.edu:6578/new/vsamframe.html
Video Clips
Video Archives
Video Servers
Search Interface
NPAC Video News Database Server Project
User Query Interface
NPAC Video Server Asset Management
Enter password: ******
and press submit button:
Submit
Reset
Please report problems to NPAC VoD Group
Marek Podgorny <marek@npac.syr.edu>
Last Modified on Mon Aug 19 18:07:46 EDT 1996
Document: Done

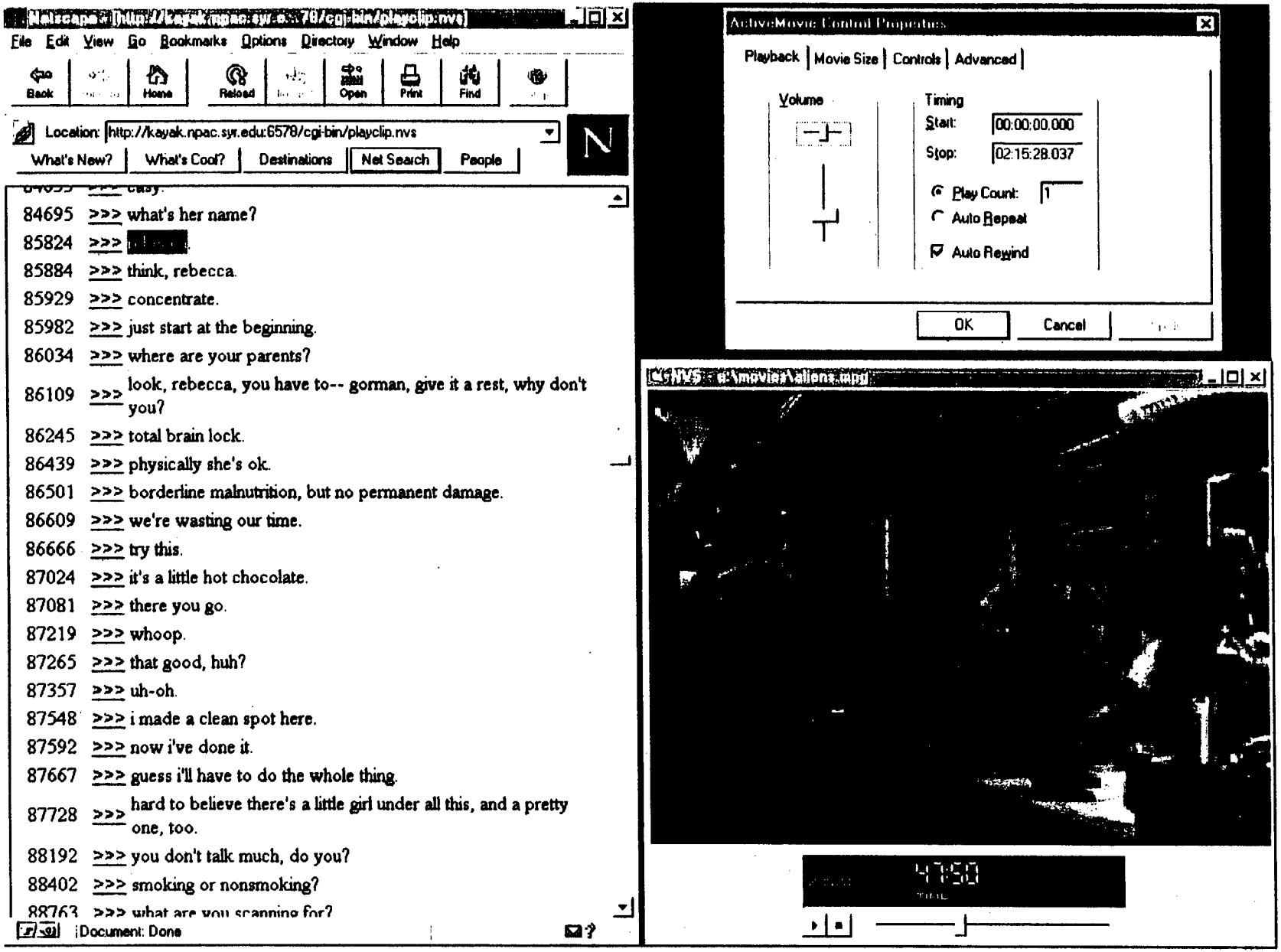

Netscape - [Video Server Asset Management]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://kayak.npac.syr.edu:6578/new/vsamframe.html
Video Clips
Video Archives
Video Servers
Search Interface
Server Management
List Servers
Add Server
Modify Server
Delete Server
Add a server to Video News DataBase
Current servers (Name : IP address : Platform)
1. Wayne : 166.101.20.33 : 2-way Pentium PC, NT, ATM int.
2. Wayne-eth : 128.230.162.33 : Windows NT, ethernet interface
3. Sandman : 166.101.20.17 : SGI Onyx, IRIX, ATM interface
4. Sandman-eth : 128.230.162.17 : SGI IRIX, ethernet interface
5. Bogart-eth : 128.230.162.113 : Windows NT, ethernet interface
New Server
Server Name
Server IP Address
Platform
ADD SERVER
RESET

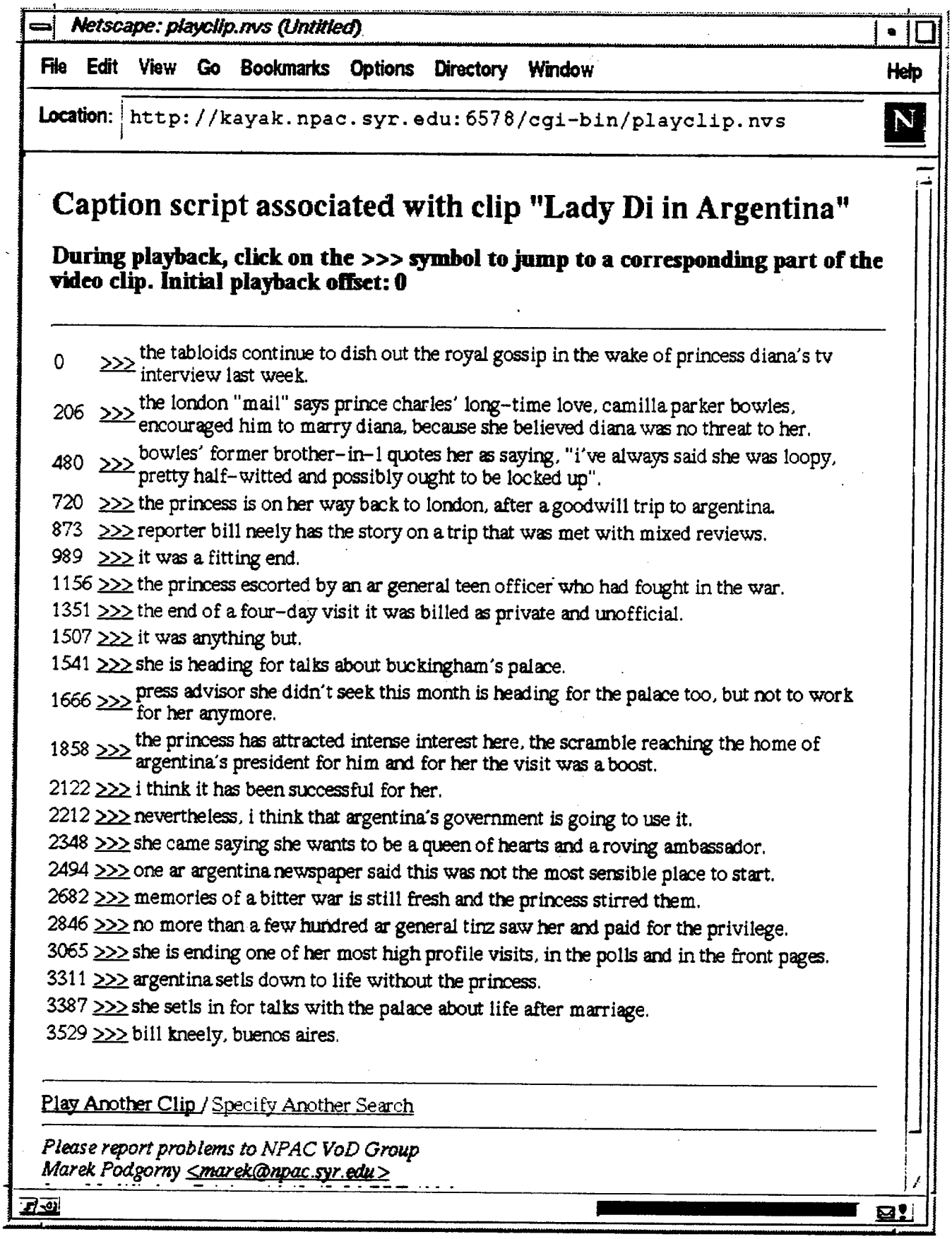

Netscape - [Video Server Asset Management.]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://kayak.npac.syr.edu:6578/new/vsamframe.html
Video Clips
Video Archives
Video Servers
Search Interface
Clip Management
Select Archive
Add Clip
Clone Clip
Modify Clip
Rename Clip
Delete Clip
Select Archive
ALL ARCHIVES
Berlin : Bob Fray Berlin story
CNN : CNN News Clips
Default : Default archive (holds clips from deleted archives)
Discovery : Discovery Channel Video Clips
MISC : Various stuff
MOST : Museum of Science&Technology Clip Collection
Military : Military clip archive
Motion Pictures : Full length movie archive
Music : Music video archive
All operations will be performed on clips belonging to selected archiv
SELECTED
Document: Done

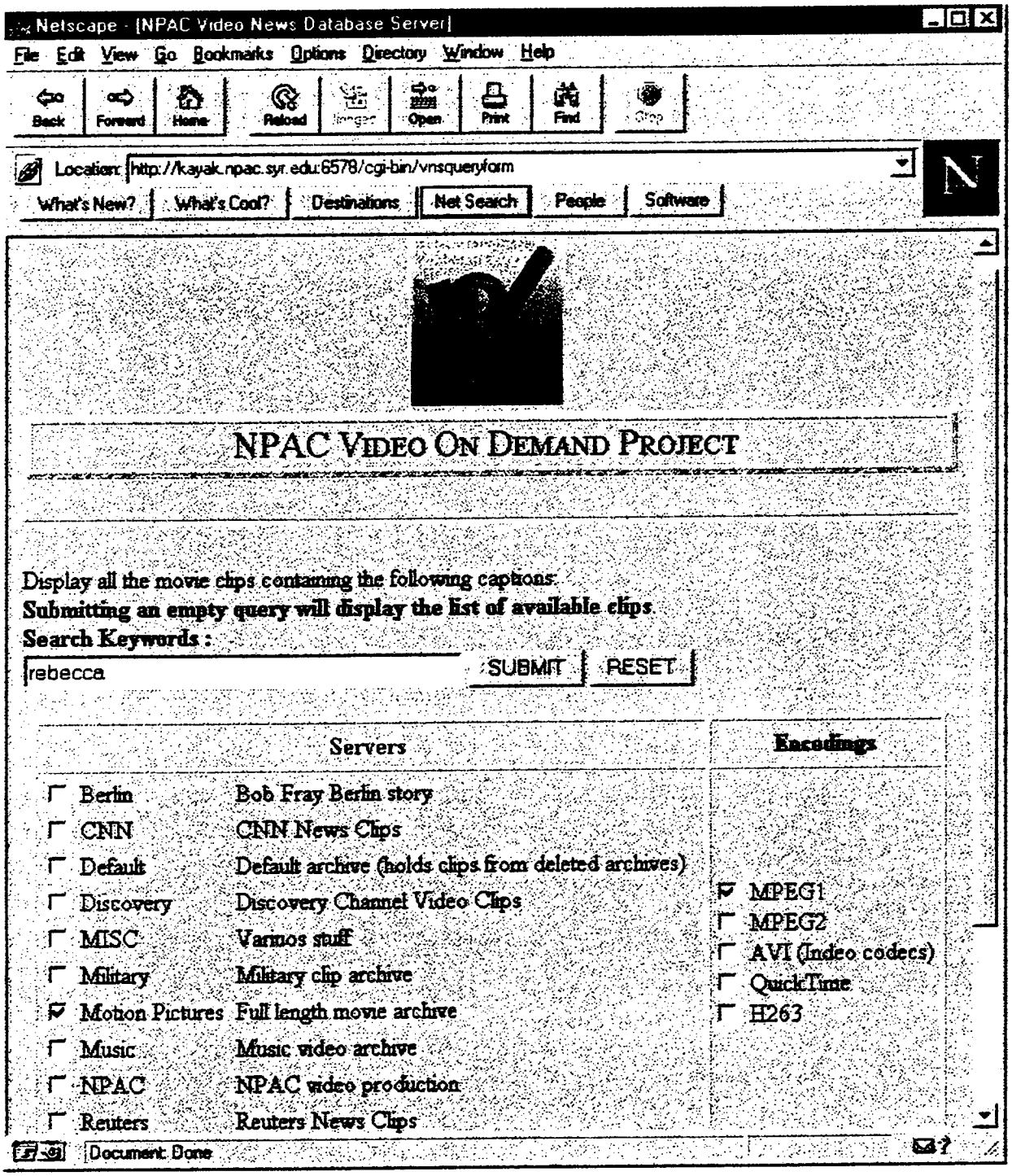

Netscape - [Video Server Asset Management.]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://kayak.npac.syr.edu:6578/new/vsamframe.html
Video Clips
Video Archives
Video Servers
Search Interface
Clip Management
Select Archive
Add Clip
Clone Clip
Modify Clip
Rename Clip
Delete Clip
Basic Clip Data:
Title: Computer Inside
Clip ID 81
*.frs file: /project/K136/VOD/CP/frs/???.frs
Encoding MPEG1
Fingernail URL /fingernails/dummy.gif
Date inserted: 03-MAR-96
Stream Attributes:
Clip length (frames): 5000
Clip file size (bytes): Auto
Frame rate (fps): 30
Bit rate (Kbps): Auto
Clip Access Attributes:
Archive: CNN
Access level: ALL
Downloadable: NO
Currently available on the following server(s):
Sandman
Wayne

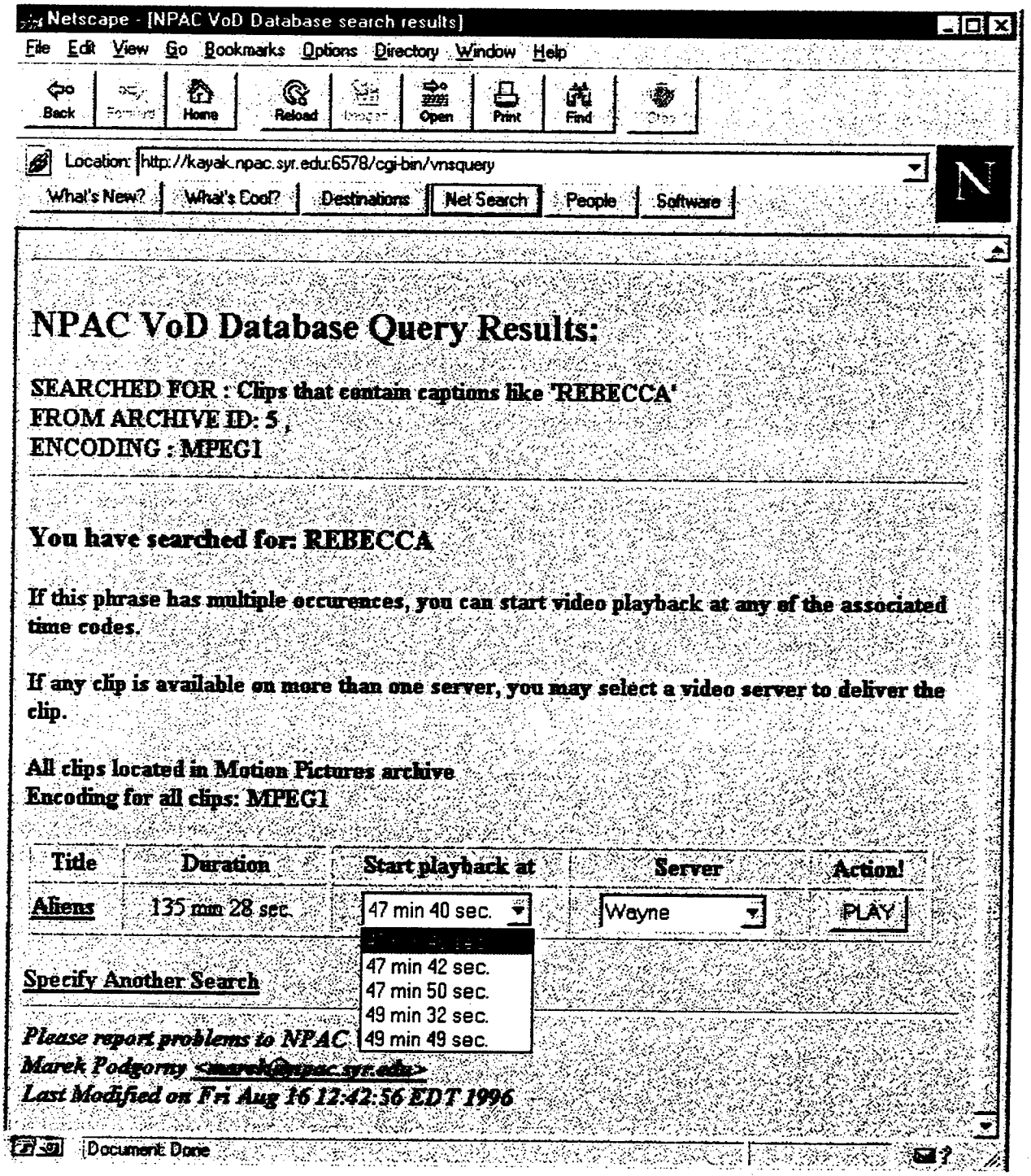
10.5 Appendix 5: Video Database Search Interface Screen Dumps

This Appendix contains screen dumps of the Video Database search interface.


File Edit View Go Bookmarks Options Directory Window Help
Back
Home
Reload
Open
Print
Find
Location: http://kayak.npac.syr.edu:6578/cgi-bin/playclip.nvs
What's New?
What's Cool?
Destinations
Net Search
People
84695 >>> what's her name?
85824 >>>
85884 >>> think, rebecca.
85929 >>> concentrate.
85982 >>> just start at the beginning.
86034 >>> where are your parents?
86109 >>> look, rebecca, you have to-- gorman, give it a rest, why don't you?
86245 >>> total brain lock.
86439 >>> physically she's ok.
86501 >>> borderline malnutrition, but no permanent damage.
86609 >>> we're wasting our time.
86666 >>> try this.
87024 >>> it's a little hot chocolate.
87081 >>> there you go.
87219 >>> whoop.
87265 >>> that good, huh?
87357 >>> uh-oh.
87548 >>> i made a clean spot here.
87592 >>> now i've done it.
87667 >>> guess i'll have to do the whole thing.
87728 >>> hard to believe there's a little girl under all this, and a pretty one, too.
88192 >>> you don't talk much, do you?
88402 >>> smoking or nonsmoking?
Document: Done
ActiveMovie Control Properties
Playback
Movie Size
Controls
Advanced
Volume
Timing
Start: 00:00:00.000
Stop: 02:15:28.037
Play Count: 1
Auto Repeat
Auto Rewind
OK
Cancel
NVS: e:\movies\aliens.mpg

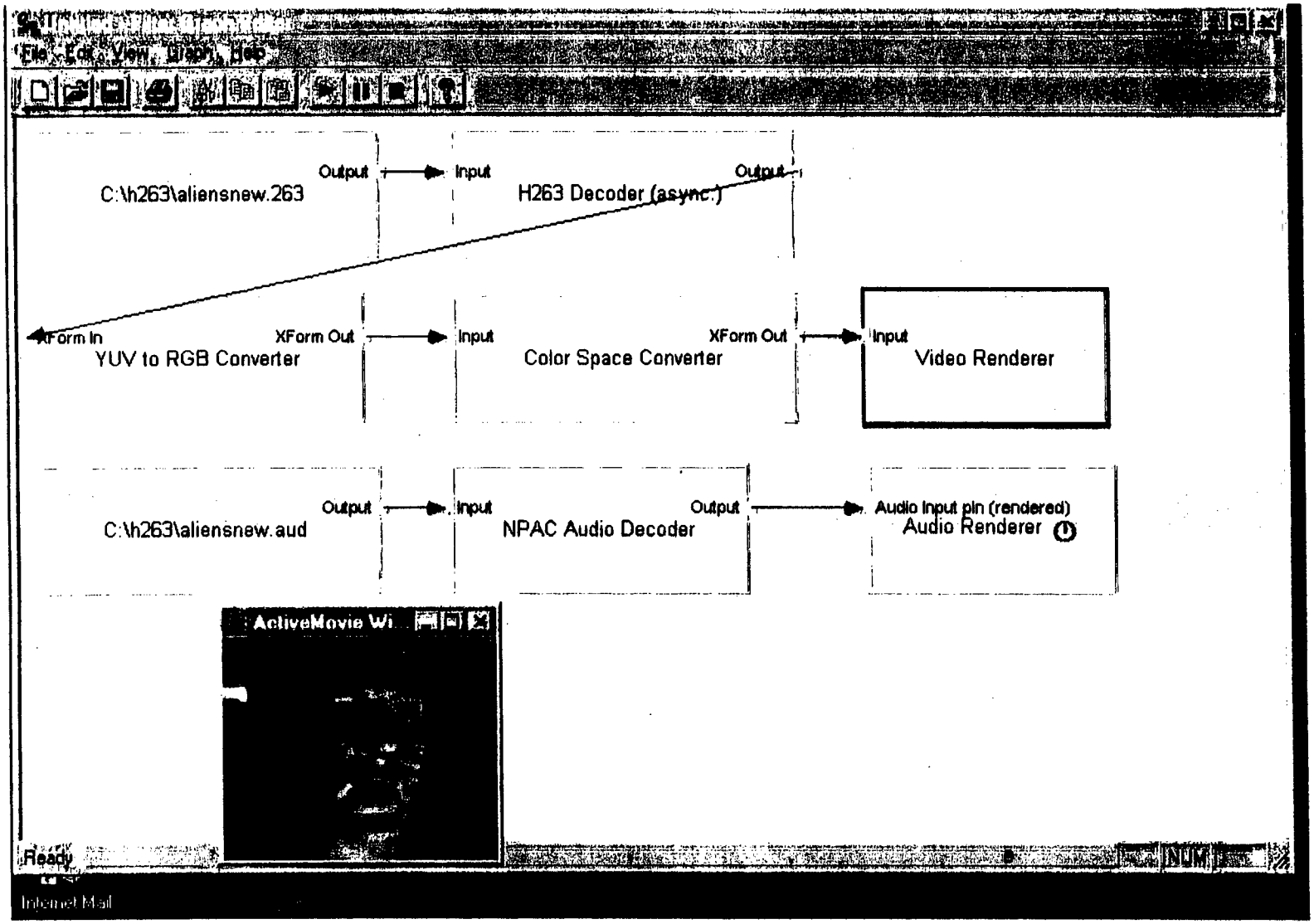

Netscape: playclip.nvs (Untitled)
File Edit View Go Bookmarks Options Directory Window Help
Location: http://kayak.npac.syr.edu:6578/cgi-bin/playclip.nvs
Caption script associated with clip "Lady Di in Argentina"
During playback, click on the >>> symbol to jump to a corresponding part of the video clip. Initial playback offset: 0
0 >>> the tabloids continue to dish out the royal gossip in the wake of princess diana's tv interview last week.
206 >>> the london "mail" says prince charles' long-time love, camilla parker bowles, encouraged him to marry diana, because she believed diana was no threat to her.
480 >>> bowles' former brother-in-l quotes her as saying, "i've always said she was loopy, pretty half-witted and possibly ought to be locked up".
720 >>> the princess is on her way back to london, after a goodwill trip to argentina.
873 >>> reporter bill neely has the story on a trip that was met with mixed reviews.
989 >>> it was a fitting end.
1156 >>> the princess escorted by an ar general teen officer who had fought in the war.
1351 >>> the end of a four-day visit it was billed as private and unofficial.
1507 >>> it was anything but.
1541 >>> she is heading for talks about buckingham's palace.
1666 >>> press advisor she didn't seek this month is heading for the palace too, but not to work for her anymore.
1858 >>> the princess has attracted intense interest here, the scramble reaching the home of argentina's president for him and for her the visit was a boost.
2122 >>> i think it has been successful for her.
2212 >>> nevertheless, i think that argentina's government is going to use it.
2348 >>> she came saying she wants to be a queen of hearts and a roving ambassador.
2494 >>> one ar argentina newspaper said this was not the most sensible place to start.
2682 >>> memories of a bitter war is still fresh and the princess stirred them.
2846 >>> no more than a few hundred ar general tinz saw her and paid for the privilege.
3065 >>> she is ending one of her most high profile visits, in the polls and in the front pages.
3311 >>> argentina setls down to life without the princess.
3387 >>> she setls in for talks with the palace about life after marriage.
3529 >>> bill kneely, buenos aires.
Play Another Clip / Specify Another Search
Please report problems to NPAC VoD Group
Marek Podgorny <marek@npac.syr.edu>


Netscape - [NPAC Video News Database Server]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://kayak.npac.syr.edu:6578/cgi-bin/vnsqueryform
What's New? What's Cool? Destinations Net Search People Software
NPAC VIDEO ON DEMAND PROJECT
Display all the movie clips containing the following captions:
Submitting an empty query will display the list of available clips.
Search Keywords :
rebecca
SUBMIT
RESET
Servers
Encodings
Berlin Bob Fray Berlin story
CNN CNN News Clips
Default Default archive (holds clips from deleted archives)
Discovery Discovery Channel Video Clips
MISC Various stuff
Military Military clip archive
Motion Pictures Full length movie archive
Music Music video archive
NPAC NPAC video production
Reuters Reuters News Clips
MPEG1
MPEG2
AVI (Indeo codecs)
QuickTime
H263
Document Done


Netscape - [NPAC VoD Database search results]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://kayak.npac.syr.edu:6578/cgi-bin/vnsquery
What's New? What's Cool? Destinations Net Search People Software
NPAC VoD Database Query Results:
SEARCHED FOR : Clips that contain captions like 'REBECCA'
FROM ARCHIVE ID: 5 ,
ENCODING : MPEG1
You have searched for: REBECCA
If this phrase has multiple occurences, you can start video playback at any of the associated time codes.
If any clip is available on more than one server, you may select a video server to deliver the clip.
All clips located in Motion Pictures archive
Encoding for all clips: MPEG1
Title
Duration
Start playback at
Server
Action!
Aliens
135 min 28 sec.
47 min 40 sec.
Wayne
PLAY
47 min 42 sec.
47 min 50 sec.
49 min 32 sec.
49 min 49 sec.
Specify Another Search
Please report problems to NPAC
Marek Podgorny <marek@npac.syr.edu>
Last Modified on Fri Aug 16 12:42:56 EDT 1996
Document Done

10.6 Appendix 6: Video Clients Screen Dumps

This Appendix contains screen dumps showing video clients in operation.


File Edit View Graph Help
Output
Input
C:\h263\aliensnew.263
H263 Decoder (async.)
Output
XForm In
XForm Out
Input
YUV to RGB Converter
XForm Out
Input
Color Space Converter
Input
Video Renderer
Output
Input
C:\h263\aliensnew.aud
NPAC Audio Decoder
Output
Audio Input pin (rendered)
Audio Renderer
ActiveMovie Wi
Ready
NUM
Internet Mail

Netscape - [Video Server Asset Management.]

File Edit View Go Bookmarks Options Directory Window Help

Back Home Reload Open Print Find

Location: http://kayak.npac.syr.edu:6578/new/vsamframe.html

What's New! What's Cool! Handbook Net Search Net Directory Software

Video Server Asset Management

Video Clips | Video Archives | Video Servers | Search Interface

Full Screen

NPAC VoD Database C

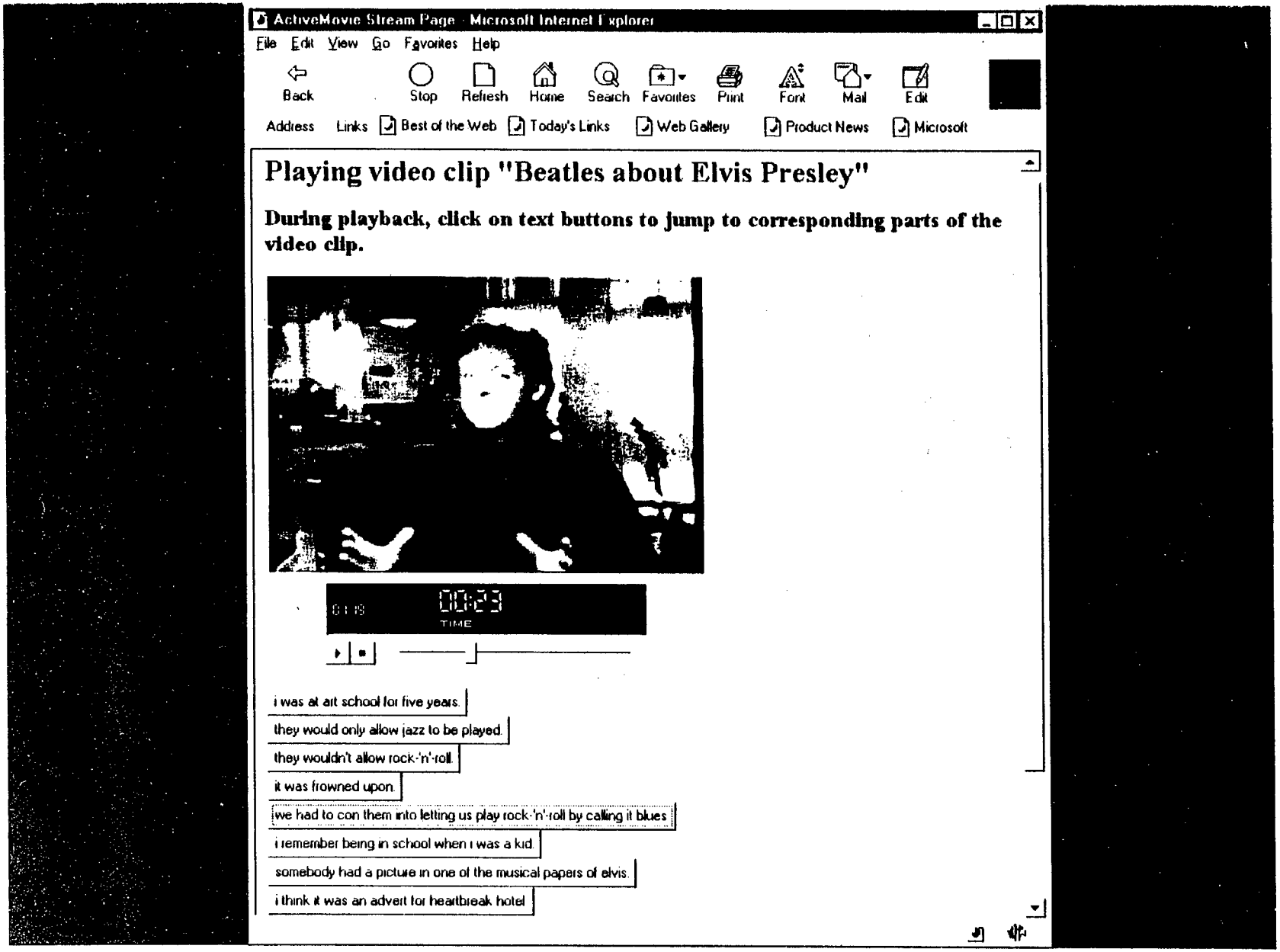
SEARCHED FOR : All clips
FROM ARCHIVE ID: 2 ,
ENCODING : All types

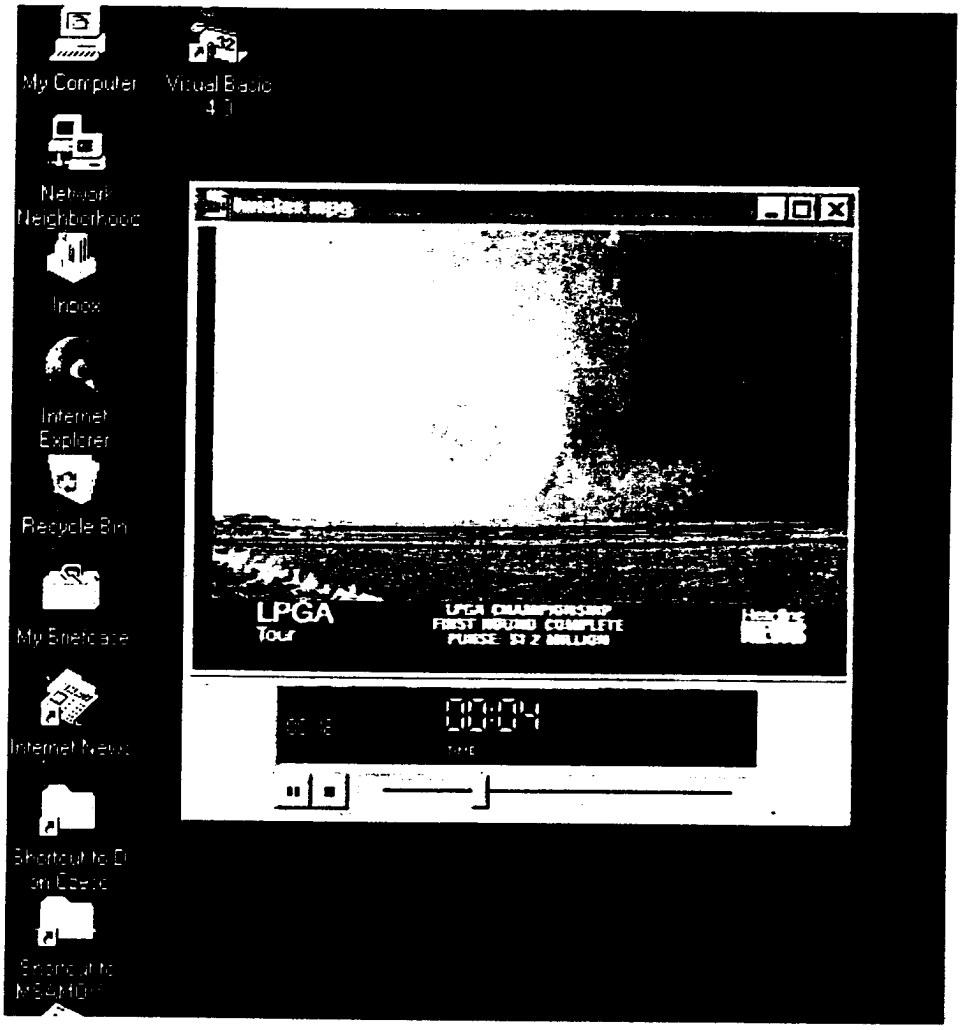
If any clip is available on more than deliver the clip.

All clips located in CNN archive



Title	Duration	playback at	Codec	Server	Action!
Comment on current situation in Bosnia	2 min 22 sec.	the beginning	MPEG1	Wayne	PLAY
Computer Inside	2 min 46 sec.	the beginning	MPEG1	Wayne	PLAY


ActiveMovie Stream Page - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Stop Refresh Home Search Favorites Print Font Mail Edit
Address Links Best of the Web Today's Links Web Gallery Product News Microsoft
Playing video clip "Beatles about Elvis Presley"
During playback, click on text buttons to jump to corresponding parts of the video clip.
TIME
i was at art school for five years.
they would only allow jazz to be played.
they wouldn't allow rock-'n'-roll.
it was frowned upon.
we had to con them into letting us play rock-'n'-roll by calling it blues
i remember being in school when i was a kid.
somebody had a picture in one of the musical papers of elvis.
i think it was an advert for heartbreak hotel

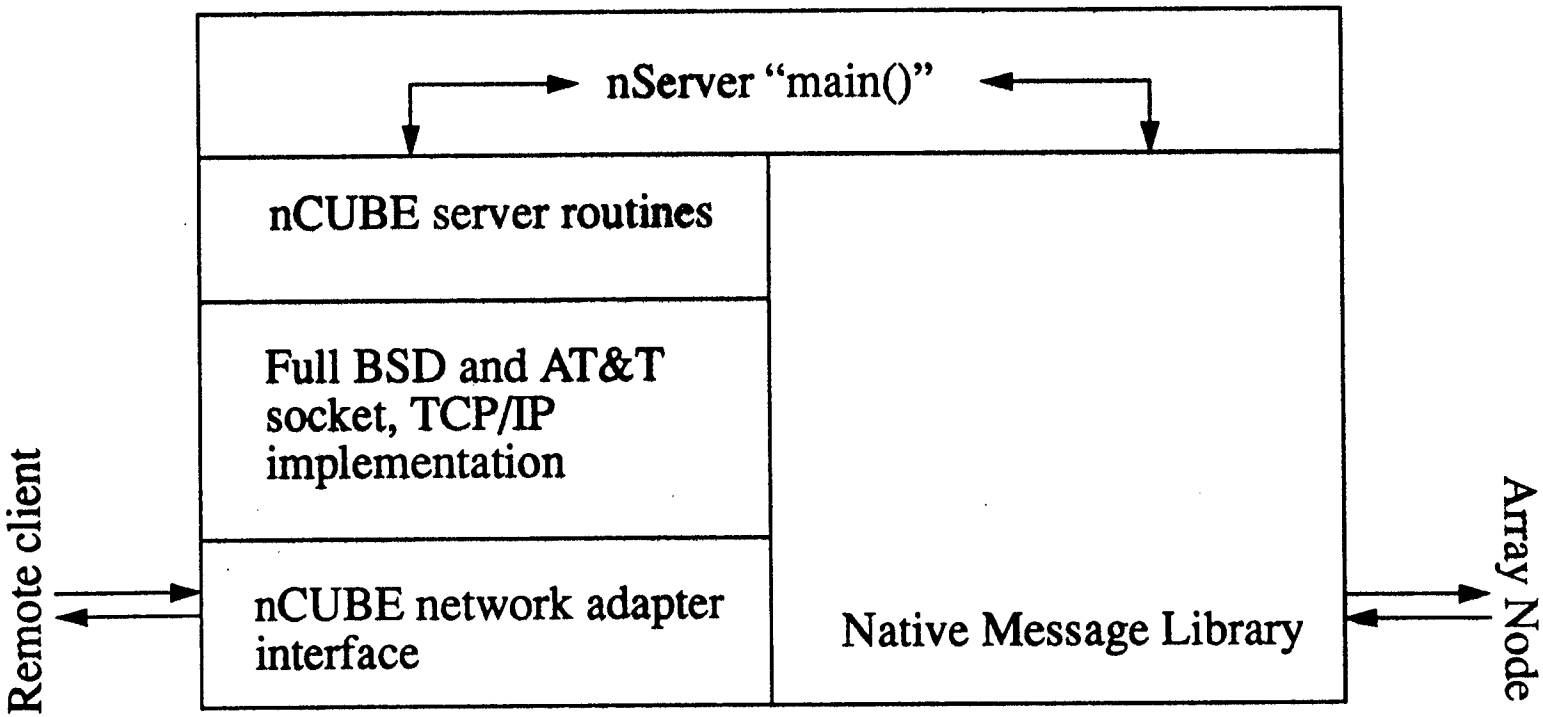

My Computer
Visual Basic 4 32
Network Neighborhood
Inbox
Internet Explorer
Recycle Bin
My Briefcase
Internet News
Shortcut to D on Eagle
Shortcut to MSDN10
LPGA
Tour

10.7 Appendix 7: Details of the nCUBE2 OS network layer architecture

This Appendix provides details of the nCUBE2 OS network layer implementation.

nCUBE NETWORK ARCHITECTURE

I/O Node:



nCUBE NETWORK ARCHITECTURE

I/O Node:

- nServer "main()" -- responds to requests from nCUBE array processors. Calls appropriate service routine after calling SvrPortGet().

- SvrPortGet() -- verifies existence of network port (i.e., socket) structure associated with the array node request message. If ok, calls service routine (e.g., SvrWriteReq()).

- SvrWriteReq() -- services write requests from an array processor. To send message out to network, calls BSD socket implementation routines (i.e., sosend()).

- sosend() -- from this function on, full Berkeley and AT&T TCP/IP implementation is used. Network adapter interface layer is provided at lowest level.

nCUBE NETWORK ARCHITECTURE

Array Node: "Write" path

- Application calls standard Socket interface.
- Connection setup is accomplised using a combination of implementations. Most important functions are nCUBE implemented.

  e.g.,

  nCUBE library : socket(), listen(), write(), etc.

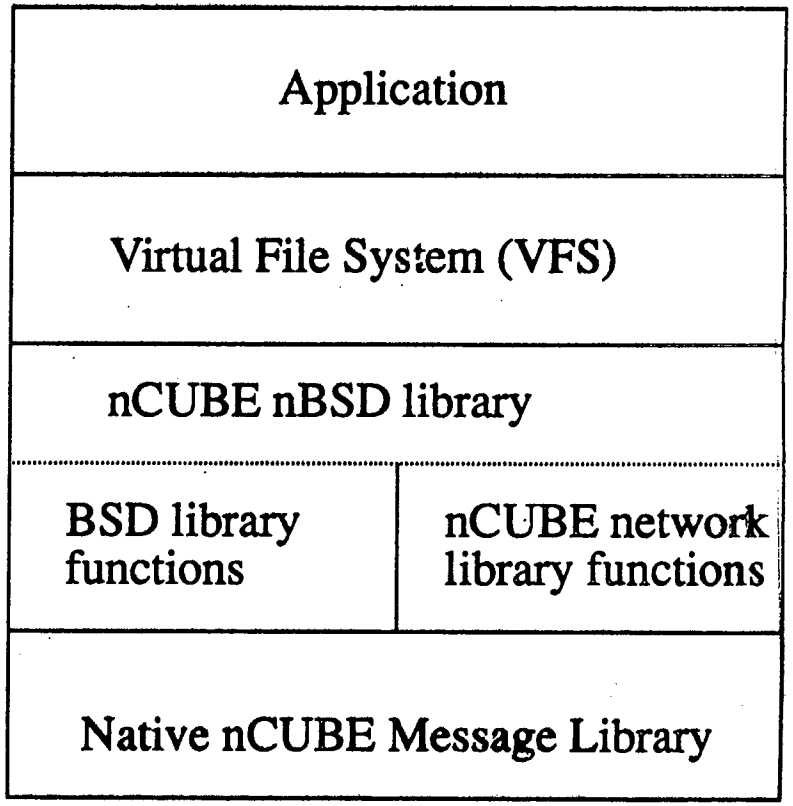
  BSD library: gethostbyname(), inet_addr(), etc.

- "Write()" call gets caught by VFS and arguments get marshalled. Entry into nCUBE nBSD library is at "write_nbsd()".

nCUBE NETWORK ARCHITECTURE

Array Node:

- Application uses standard Socket and Unix TCP/IP interface.
- VFS layer provides uniform interface to large variety of file system and device libraries.
- nCUBE library composed of subset of BSD network library + nCUBE implementation of complement of BSD network library.
- Native nCUBE message library contains nread(), nwrite(), etc. implementations.



nCUBE NETWORK ARCHITECTURE

Array Node: "Write" path overheads

- VFS layer latency is not exactly known (unavaliable). However, this layer is notoriously slow. Involves data copies, sometimes disassembly (fortunately, it seems no disassembly takes place for at least up to 32K).

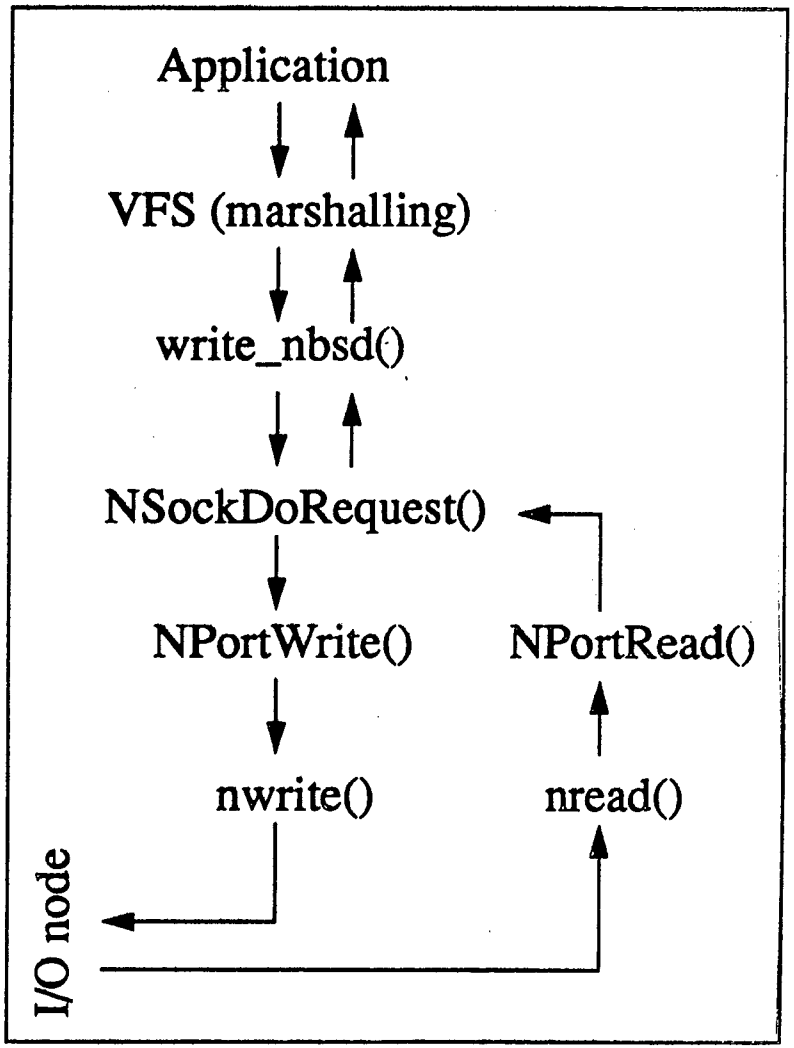
- write_nbsd() -- this routine copies marshalled data into nCUBE comm buffer prior to calling NSockDoRequest().

- Possibly most significant source of delay is the reliable messaging incorporated into the protocol; every write involves round trip message latency prior to returning.

nCUBE NETWORK ARCHITECTURE

Array Node: "Write" path

- write_nbsd() -- entry into nCUBE nBSD library.
- NSockDoRequest() -- sends message to nBSD server and gets a response.
- NPort*() -- simulated port based IPC facility using nCX message types. A port address contains a node no., proc. address, message types, and a sequence no.
- Note: NSockDoRequest() implements a reliable messaging protocol internal to nCUBE.



nCUBE NETWORK ARCHITECTURE

Hurdles and stumbling blocks:

- Profiling and debugging is difficult -- nCUBE debuggers don't work with network routines. Timer routines have redefinition conflict between standard C library and network library.

- I/O node server is multithreaded, with LOTS of state. Alterations to code structure design is made very complicated by this.

- Full network implementation on the I/O node makes it more likely to be the bottleneck for large numbers of client array nodes.

nCUBE NETWORK ARCHITECTURE

Experiments and results:

- Initial build was made difficult by lack of documentation and cryptic "implicit" linking of network library with the -DNBSD compiler directive.

- Further build problems originated from a parsing error by the compiler (when -DNBSD_VOD was recognized as -DNBSD).

- Bypass of copy at write_nbsd() routine -- Data was properly transferred down to the native nCUBE messaging layer. Messages were properly received at I/O node and sent out to remote client.
  Problem: after ~6 messages were sent, process faulted with broken pipe error; or, messages would be sent, but corrupted.

- Bypass of reliable communication protocol within nCUBE -- The idea here was to later replace this by some Go-Back-N protocol. Messages were continuously sent by array node. I/O node received the messages properly. I/O node sends first message to client correctly, then finds a state error upon receiving the second. It has not been determined which state variable is responsible; lack of a working debugger and the complex nature of state structures makes this time consuming.

10.8 Appendix 8: Slides of the final project presentation

Task 1: Develop and Evaluate Video Server

- **We have designed and implemented a complete, working, end-to-end video on demand system including:**
  - video digitization and encoding from multiple sources to multiple video codecs
  - automatic video indexing system via closed caption
  - random access to video material based on metadata search
  - video retrieval using multiple access methods
  - video server monitoring and asset management layers
- **System highlights:**
  - modular design ensures system flexibility and extensibility
  - design supports both on-demand and broadcast mode
  - Web technology provides GUI and middleware
  - multiplatform, both server and client

Task 2: Develop and Evaluate Video Network Service

- **Network video client technology.**
  - Evaluated most of the available video client technologies
  - Implemented codec-independent video client for PC platforms and multiple other client architectures
- **Evaluated ATM support for multimedia services**
- **Evaluated QoS support for packet-switched networks**
- **Evaluated various topologies and delivery options**
- **Developed new concepts for handling VBR multimedia traffic**
- **Designed and implemented ATM testbed to investigate service scalability issues**

Task 3: Evaluate capabilities of HPCC platforms for VoD services

- **Evaluated three different platforms for video server implementation**
  - nCUBE2 multiprocessor
  - SGI symmetric processor with shared memory
  - cluster of Windows NT workstations
- **Implemented video servers on Windows NT and SGI platforms**
- **Evaluated and benchmarked different multithreaded video server architectures**
- **Designed scalable I/O system for continuous media supporting the notion of the end-to-end scalable server design**
- **Provided blueprint for a scalable, distributed, dynamically reconfigurable VoD system design**

Task 4: Evaluate and Implement user interface technologies for VoD

- **Developed the concept of an integrated video delivery system with indexed, random access**
  - implemented video asset management system using Web browser as a front-end, Web technology as middleware, and relational database system as a backend
  - implemented user interface to the VoD service based on text search
  - implemented a variety of video client interactive interfaces

Task 5: Upgrade existing VoD demonstrations to run in WAN ATM environment

- **All development tasks 1-4 were constantly contributing to the evolving set of VoD demonstrations.**
  - Installed and configured ATM WAN capability
  - Analyzed and tuned ATM WAN performance to support VoD bandwidth requirements.

Task 6: Evaluate and Demo the WAN ATM network delivery of VoD

- **Video in Demand demonstrations over WAN ATM links have been presented at the following locations:**
  - Supercomputing '95, December '95, San Diego
    - **running over NYNET and vBNS backbone**
  - HPDC-4, August '95, Pentagon City
    - **running over NYNET, WilTel, and Bell Atlantic links**
  - Interop, March '95, Atlanta
    - **running over NYNET and WilTel links**
  - HPDC-5, August '96, Syracuse
    - **running over NYNET**
  - Rome Laboratory, April '96
    - **running over NYNET**

Project Goals vs. Project Results

- **Have we delivered?**
- Initial expectations:
  - a demonstrable, end-to-end VoD system should have been built
  - no actual software delivery was expected
  - the system should have been tested in a context of VoD applications
  - research was supposed to concentrate on ATM networking issues
- What really happened:
  - system designed and implemented
  - deliverable software package
  - minimal application testing
  - extensive ATM/network evaluation but minimal creative research
  - research concentrated of system issues

Driving Forces of the Project

- Technological stagnation of ATM
- Explosive theoretical development of multimedia -related protocols for IP networks, acceptance of IETF standards
- Free-for-all in digital video and network multimedia delivery technology
- Digital convergence: Telcos, CATV, and Internet becoming ONE
- Explosion of Web technology as a universal multimedia delivery vehicle
- Necessity to provide inexpensive solutions for schools

ATM Technology for Multimedia Delivery

- **During last two years, there was no significant progress in COTS ATM technology**
  - very slow progress of ATM forum in setting up ATM standards
    - **LANE 1.0 accepted May '95.**
  - almost zero progress of ATM vendors in providing working implementation of basic ATM protocols beyond absolute essentials
    - **no working LANE in devices until end of the project period**
      - ***installed every available upgrade on FORE hardware***
      - ***tested CISCO and Cabletron hardware***
      - ***totally unreliable information from vendors' sales force***
- **A number of standards accepted in October '96 - faster progress?**

ATM Technology for Multimedia

- **LAN Emulation Update**
  - provides support for connectivity between ATM- attached and Ethernet/Token Ring attached hosts; supports ELANs. LANE is layer 2 protocol, equivalent to bridging.
  - LANE does not offer much support for multimedia apps:
    - **No direct support for FDDI - a router needed to translate FDDI frames to 802.3/5 frames**
    - **No QoS support at all. Discussion in progress**
    - **No support for any traffic type but UBR (Unspecified Bit Rate)**
    - **No sharing of VCs by multiple flows**
- **LANE opens the way to test scalable ATM testbeds, but it is far from sufficient support for multimedia applications.**

ATM Technology for Multimedia

- **A number of critical ATM standards is still unsettled. The most important :**
  - Signaling: only UNI version 4.0 will support ABR and QoS
  - Service aspects and applications: audio/video multimedia services
  - LANE: FDDI support
  - MPOA: support for QoS (protocols like RSVP)
  - Traffic Management: ABR, QoS parameter settings
  - Network Management: performance monitoring
  - Physical Layer: support for DS3
  - Residential Broadband: last mile. NOTE: Fiber-to-the-Curbl
- **Within a year, ATM will probably start providing the long expected support for multimedia applications**

IP Networks and Multimedia

- **Integrated Services Model evolved to the point of a set of accepted IETF standards.**
- **Main established components:**
  - Multicast routing (Distance Vector, PIM)
  - Reservation Protocol (RSVP)
  - Real Time Protocol (RTP)
- **Emerging components (working drafts)**
  - Real Time Streaming Protocol (RTSP) - Netscape/ Progressive Networks
  - Simple Conference Management Protocol/Session Control Protocol (SCMP/SCP)

Digital Video and Video Network Delivery

- **Explosive development of technology in a field with no leaders, quality standards, or even clearly defined goals!**
  - process driven by attempts to gain early market share for hardly useful products
  - lack of standards, proprietary solutions still seen as essential for market carving
  - incoherent visions proposed by different industry sectors
  - very uneven research background: from significant for integrated network services to hardly existent for video server architectures
  - multi-platform development practically not feasible as lack of standards makes the process prohibitively expensive

Digital Video and Video Network Delivery

- **Development frameworks**
  - Microsoft Video for Windows, later replaced by Active Movie architecture
    - **ActiveMovie is probably the first comprehensive network multimedia delivery framework**
    - **NetShow**
  - SGI Digital Media Library, development and run-time
    - **MediaBase**
  - Apple QuickTime
  - Avid-supported OMF
  - Sigma OpenMPEG
  - Oracle Media Objects
  - Dozens of others....

Basic research issues addressed in the project

- **Variable Bit Rate vs. Constant Bit Rate transmission issues: buffer space vs. bandwidth optimization**
- **Wavelet compression for video**
  - motion compensation for wavelets
  - hybrid wavelet/DCT schemes
  - scalable coders
- **I/O support for scalable coders: Continuous Media File System**

VoD Services Implementation

- **We have designed, implemented, and integrated a modular Video on Demand system supporting all of the following aspects:**
  - digital video creation (digitization, encoding, reencoding)
  - asset management (metadata, indexing, backend database)
  - network delivery (video servers, video clients, overall system architecture)
  - we have implemented multi-platform support
  - modular design and the overall architecture provides foundation for a future distributed server

Contents Creation Service

- **Integrated software and hardware for real-time video encoding facility**
  - supported video source: VHS, S-VHS, and Betacam tapes, satellite link
    - **VTRs with full computer control, batch encoding available**
    - **Satellite link: fiber from Newhouse II to NPAC, dedicated satellite dish receiving from Galaxy IV, tunable satellite receiver on antenna site, remotely controlled by modem via software. Broadcast quality video with 2 audio channels.**
  - supported digital video formats:
    - **MPEG1 system or elementary stream, Intel Indeo4 (AVI)**
      - ***real-time encoding in hardware (OptiBase, Intel)***
    - **MPEG1 in MPEG2 transport stream (postprocessing)**
    - **H.263 + GSM or ADPCM audio**
      - ***off-line software reencoding from high-quality MPEG1 stream***
  - support for computer screen movies

Contents Creation Service

- **Integrated closed caption encoder**
  - using external hardware VBI decoder with serial output port
  - caption parser software implemented in Visual Basic
    - **parses the CC stream and breaks it into phrases**
    - **receives relative timestamps from the video source**
    - **runs in background concurrently to the video encoder**
    - **stores caption in a structured file for database loading**
    - **capable of monitoring multiple CC channels**
- **Database insert automatization layer**
  - Goal: support automatic video acquisition and DB loading
    - **ScriptMaker for automated control of GUI-based applications**
    - **UNIX-based daemon for scheduled, automatic data acquisition, satellite transponder control, data movements, database loading.**
    - **multicomponent, extremely heterogenous system integrating 3rd party monolithic applications**

Video Server Asset Management

- Main design principle: strict separation of video data and all other data (metadata)
  - video/audio data stored only in file systems with (possible) real-time support
  - metadata stored only in relational database
- VSAM: simple interface to all video repository management tools
  - implemented using Web front-end (HTML, Java, Javascript), CGI with Perl/oraperl, and Oracle RDBMS
  - supports multiple servers and archives, designed to support distributed server with dynamic, load-based resource re-allocation
  - interactively supports data loading and data management procedures
  - designed to ensure heterogenous database consistency

Metadata Concept

- Any data item characterizing or otherwise related to a video clip is considered metadata
  - technical clip parameters
    - encoding, framerate, bitrate, length, file size, format, frame to file offset lookup table, VBR pattern if any .....
      - *automatically created by encoding procedures and exported for DB load*
  - clip attributes
    - source, author, copyright, creation date, insert date, credits...
  - clip statistics
    - availability on servers, address on servers, clip usage patterns...
    - # of copies, downloadability, access scope, charge per viewing...
  - clip associated data
    - title, annotations, captions with timestamps
      - *annotations: enable video-driven slide shows, can include text, image, URL*
- Metadata is organized into a relational schema and is fully searchable

Video Database Indexing Support

- User search capabilities based on metadata contents
  - search attributes:
    - archive, encoding type, date....
  - contents search of:
    - titles
    - annotations
    - closed captioning
- Each search retrieves both the video and the associated information item/object
- Closed caption search supports fully interactive, random access to audio/video streams

Video on Demand Architecture - System Software and Hardware Components

- Web browser: system GUI for most functions
  - platform independent, requires Java support
- HTTP server: serves applets, supports CGI to the RDBMS
  - platform independent, needs Perl/oraperl and SQL*Net
- Oracle RDBMS: commercial product, metadata backend
- Video server(s):
  - cluster of audio/video data pumps under common management
  - implemented for SGI IRIX and Windows NT platforms
- Video clients:
  - play video, support VCR mode, interact with the browser
  - a number of different implementations for UNIX and PC platforms

Video on Demand Architecture - Data Flow Model

- Select video clip attributes and/or a keyword.
  - input sent to CGI script. CGI script retrieves a "hit list" from the database server and formats next stage form, returns to browser
- Select clip to play and, optionally, stream entry point
  - input sent to CGI script. CGI script retrieves metadata needed by video client and formats the MIME metafile. Metafile is sent to the browser. The system might decide which video server to engage.
- Browser acts accordingly to the type of video client (helper application, plug-in, ActiveX control, Java applet)
  - textual data is displayed in browser. Time-stamped textual data forms a random access interface to stream entry point.
  - metafile is passed to the video client. Client parses it and retrieves clip data. Client establishes virtual connection with the video server.
- Video client receives the video stream
  - random access only involves traffic between video client and video server

Video on Demand Architecture - Data Flow Model cont.

- Browser may request access to another stream entry point
  - A simplified metafile is sent from the DB server, all clients are re-entrant
- Slideshow architecture
  - Slideshow data pointers are transferred to the video client in metafile. Client drives the browser using CCI mechanism
- Data consistency measures
  - Video server also acts as an information collection agent. Data about video archive contents can be verified. Support for distributed server function

U.S. GOVERNMENT PRINTING OFFICE: 1998-610-130-81008

Video on Demand Architecture

- **Features of the proposed architecture**
  - Web-based: web infrastructure was used as middleware whenever possible.
  - Modular: different system functions are encapsulated in well defined modules. The interfaces are formalized. Any module can be exchanged at any time.
  - Portable: we have strived to implement platform-independent architecture. This succeeded to a degree.
  - Scalable: each element of the system has a scale-up path. Distributed video server removes the most serious bottleneck.
  - Flexible: various applications may be developed on top of the video delivery system.
  - Extensible: additional functionality can be easily added
    - *adaptivity, QoS support, multicast functionality*

Video on Demand - Video Servers

- **Multiple generations of video servers have been implemented. Experiments:**
  - **process structure (different process/thread models)**
  - **interprocess communication methods**
  - **asynchronous disk I/O models**
  - **asynchronous network I/O**
  - **implementation on different operating systems**
    - *nCUBE2, SGI SMP under IRIX, Windows NT 3.5 -> 4.0*
- **All implementations share the same protocol**
- **Underlying design principle: DoD (Data on Demand)**
  - **server is data-independent. Stream's only characteristics are average bitrate and, optionally, B(t) function.**
- **Support for concurrency, random access, rate control**

Video on Demand - Video Clients

- **Goal: a codec independent, portable video client**
  - Elusive target. Most recent implementation for 32bit Wintel platforms comes close.
  - Experiments with different ways of video client integration with web browsers
    - **modified helper application (16bit Windows, SGI)**
    - **Netscape plug-in (SGI H263 client)**
    - **Netscape server-less plug-in (SGI H263)**
    - **ActiveX control based on ActiveMovie**
    - **Java applet (!)**
  - Our video client solution cover practically all technologies later found in commercial environments
    - ***some solutions still not commercially available***