A Video Mail Distribution System for Networked Personal Computers

Fraser Street and Alfred C. Weaver

Computer Networks Laboratory
Department of Computer Science
University of Virginia, Charlottesville VA 22903
[fraser|weaver]@virginia.edu

Abstract

The Computer Networks Laboratory at the University of Virginia has used the Xpress Transfer Protocol (XTP) to construct the Xpress File System (XFS), a local area network peer-to-peer file sharing system for MS-DOS based IBM Personal Computers. XFS is usable for a variety of purposes, most notably the real time delivery of multimedia data streams. Using specialized hardware and software from Fluent Machines Inc., we have shown that XFS is capable of maintaining sufficient file system to file system throughput to support the real time delivery of the 30 frames per second audio/video data stream generated by the Fluent system. This paper provides a general discussion of the architecture and operation of XFS, and an overview of the Fluent system that we use to generate and process motion video data streams. We also describe our plans to extend the video mail system into a video conferencing system using XTP.

1. Introduction

Science Applications International Corporation (McLean, VA) is a value-added reseller of multimedia computer products produced by Fluent Machines Inc. (Framingham, MA). SAIC approached the Computer Networks Lab at the University of Virginia with a requirement to develop a video mail system whereby the synchronized voice and video stream emitted by Fluent's *Fluency* [FLU92] multimedia system could be electronically mailed from user to user on a local area network. We have designed and implemented this video mail system using a peer-to-peer file sharing tool which we call XFS, the Xpress File System. XFS gives users transparent client-server access to an MS-DOS disk file system on a remote PC connected via an Ethernet or FDDI local network (see Figure 1). XFS uses the Xpress Transfer Protocol [STR92] to provide a reliable transport service between the client and server host systems. We have optimized the performance of XFS to insure that it provides the file access bandwidth required by our video mail application.

Section one of this paper provides the motivation for the XFS video mail system. Section two contains a general introduction to the hardware and software components of the *Fluency* multimedia system. Sections three and four discuss XFS and the MS-DOS device driver that is the core of XFS. In section five we describe the role of XTP as the communication protocol embedded in XFS. Section six summarizes the performance of XFS, and section seven explores the requirements for a PC based teleconferencing system.

2. Overview of *Fluency*

2.1. Hardware and software environment

Fluency is an elaborate package of hardware and software for MS-DOS based PC systems. The hardware component of *Fluency* is the VSA-1000, a set of two ISA-compatible 16-bit boards (known as the *processor* and *controller*) that are inserted into the expansion slots of an 80486 class PC. The processor board consists of an Intel i960 CPU, the C-Cube MicroSystems CS550 JPEG compression engine, and a Motorola DSP chip for audio compression. The controller board, which holds 2 MB of video RAM, interfaces with the system VGA card to render the motion video on the VGA monitor screen.

The software component of the *Fluency* system is called *FluentStreams* (a set of Microsoft Windows applications and programming tools). The primary application program is the "VideoPad Editor" which, through a Windows user interface, provides functionality similar to that of a home VCR. Through VideoPad the user can record into a disk file a JPEG [WAL91] compressed representation of any NTSC audio/video signal, including those produced by standard laserdisks and camcorders. VideoPad

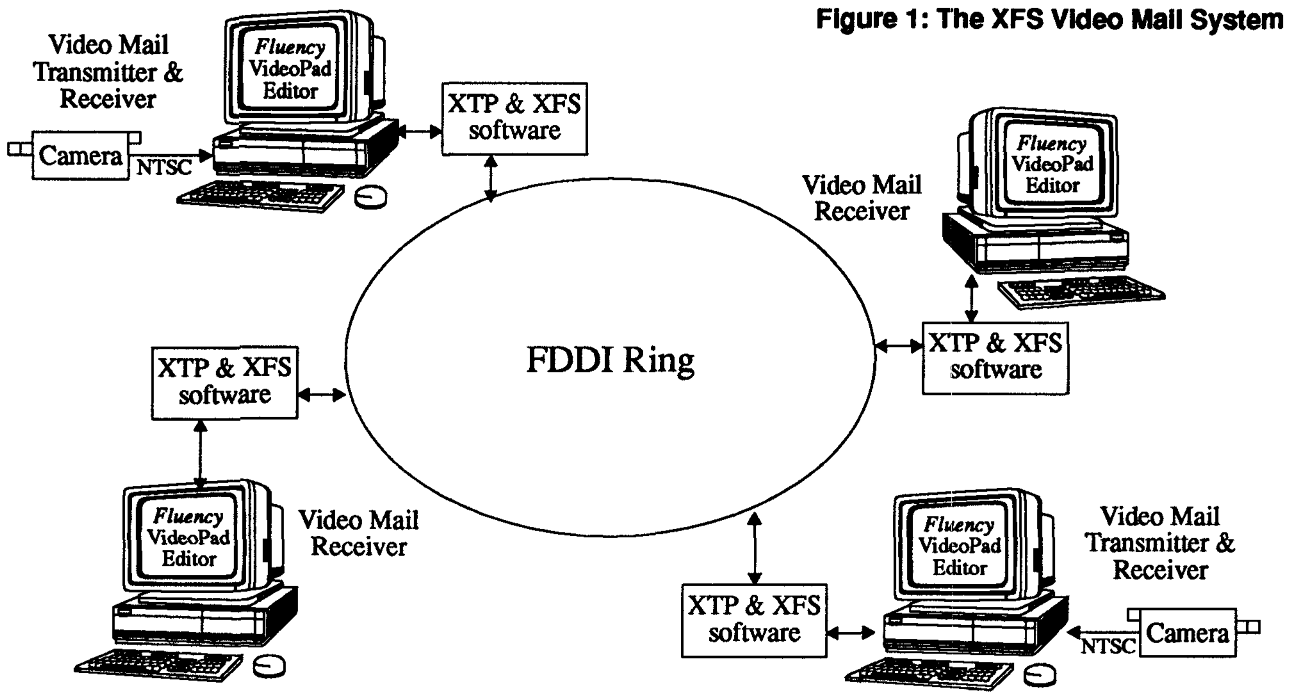

Figure 1: The XFS Video Mail System

also allows the user to play back and perform frame-by-frame editing of recorded streams.

The primary programming tool in the *FluentStreams* package is the Digital Video Object (DVO) Application Programming Interface (API). The DVO API library provides a 'C' language interface that gives the Windows application programmer control over the creation and operation of player objects. For example, the DVO API library provides calls to open, close, and edit streams, and to control system parameters (such as bit rate) prior to recording a stream. The DVO API provides exactly the sort of interface that a programmer would need to develop an application like the VideoPad Editor. Fluent provides no low level programmer interface to the compressed audio/video stream output by the VSA-1000 hardware.

2.2. Operational characteristics

The *Fluency* system provides a robust platform for the recording, editing, and playback of video streams to or from a conventional MS-DOS file system partition on a locally attached disk drive. During the recording process the *Fluency* hardware functions as an audio/video frame grabber, performing analog-to-digital conversion of the input NTSC signal at its full 30 frames per second rate. After A/D conversion, the *Fluency* system uses the C-Cube JPEG engine to compress each video frame in real time. The associated audio component of the input signal is passed to the Motorola DSP for compression, and the compressed audio and video digital data streams are integrated and written to the disk as a single file with the .FMI extension. The VideoPad Editor can be used to play back any FMI file. During play back, the full motion video appears in a 320×240 window on the VGA screen, and the accompanying synchronized audio signal can be played through external speakers.

Although the exact specification of the FMI file format is proprietary to Fluent, it is based on the Intel AVSS format. AVSS is a member of the Intel Digital Video Interactive (DVI) family of standards. Within the DVI family, the AVSS file format is primarily used for storing Real Time Video (RTV) and Production Level Video (PLV) on both CD-ROM and disk systems. AVSS files are based on the notion of Streams, for which Intel defines two main types: Compressed Image and Audio. A hierarchy of data structures within an AVSS file allows for browsing, random access to frames, and editing of interleaved streams. *Fluency* exploits the high degree of structure that AVSS imposes in order to achieve frame-by-frame random access within an FMI file, which is critical to both the DVO API library and the VideoPad Editor. There is no facility within the FluentStreams software to manipulate a video stream stored in any form other than an FMI file. The only API provided by Fluent is the DVO library, which has a user interface and functionality specifically

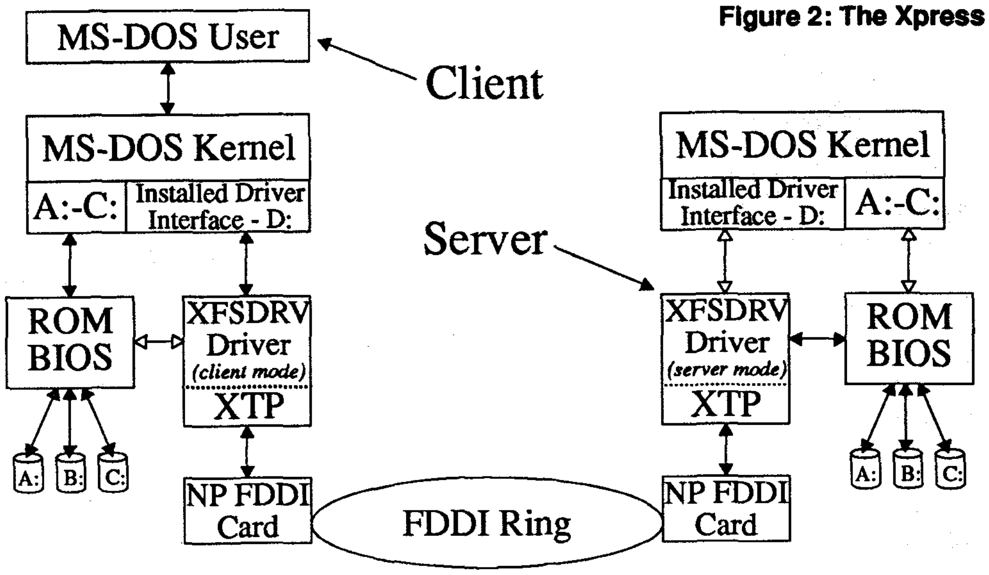

Figure 2: The Xpress File System

designed for manipulating audio/video data as it is held in FMI files.

3. Xpress File System

3.1. XFS functionality

The Xpress File System is a general purpose tool that enables peer-to-peer client-server access between a PC and an MS-DOS disk file system physically located on a remote machine (see Figure 2). Although the primary use of XFS has been in our video mail prototype system, it is a general tool that can be used for client-server access to any remote MS-DOS file system. To the human user, as well as MS-DOS and Windows application programs, a remote MS-DOS file system mounted with XFS is functionally indistinguishable from a locally attached disk drive. From the user's perspective, XFS extends the array of available disk drives from the physically attached floppy and hard disks to include a single remote disk accessed over the network through XFS. Disk systems of any size, including 3.5" and 5.25" floppies, and hard disk partitions greater than 32 MB can be mounted by XFS.

The XFS system is composed of two separate pieces of software: an MS-DOS installable block device driver (XFSDRV), and an application program (XFSCTL) that provides an I/O Control interface to the XFSDRV driver. The dynamic state of the XFS client-server system is controlled by the user through the XFSCTL program. The user issues XFSCTL commands like *mount* and *unmount* which are interpreted by the XFSCTL program and passed using the standard DOS I/O Control mechanism to the XFSDRV driver.

XFS provides a session layer communication service to the MS-DOS operating system on the client machine. The session layer connection is opened with the *mount* command, which causes XFS to establish a logical connection between the host system and the remote disk device. Once mounted, communication occurs between the client and the remote server instances of XFSDRV on behalf of MS-DOS on the client machine. The user is free to perform disk file access at will on the XFS mounted device. This access can occur from the MS-DOS system prompt using simple commands like `dir` or `cd`, or it can be achieved within an application program by using standard library calls such as `read()` and `write()`; the remote, networked nature of the XFS device is completely invisible to the user. Finally, the *unmount* command is used to close the session layer connection, terminating MS-DOS's access to the disk device on the server. At this time the client machine reverts to server mode, and is available to any other client in the network for mounting. This establishes the symmetric peer-to-peer nature of XFS, since all the machines in the network are equally capable of functioning as either a client or a server in an XFS connection.

3.2. XFS and *Fluency* as a video mail system

The ability to access a remote disk device using XFS provides the basis for the "video mail" service that was the goal of this project. We use the *Fluency* Video-Pad Editor in conjunction with XFS to record and play back FMI streams to and from remote disk drives. Before starting Windows, the user issues the XFS *mount* command from the MS-DOS prompt to open a connection with one of the XFS servers in the network. Then, after starting Windows and the VideoPad Editor, the user can select the XFS disk device from the menu of available target drives that is presented by VideoPad. Once the XFS device is selected, the VideoPad Editor will have completely transparent access through XFS to the remote file system; VideoPad can read or write FMI video streams to or from the remote disk device at greater than 2.0 Mbits/sec, which is sufficient throughput to maintain the 30 fps rate supported by the *Fluency* system.

4. The XFSDRV device driver

4.1. XFSDRV

The core of the Xpress File System is the XFSDRV device driver [LAI92]. MS-DOS Version 5 supports user-supplied installable character and block device drivers [MIC91]. Microsoft documents an interface of more than 20 commands, called device service requests (DSR), that MS-DOS uses to communicate with installed drivers. MS-DOS makes use of seven of these commands when accessing drivers for disks with removable media (e.g., floppy drives). As a removable media device, XFSDRV supports these seven commands.

Consider a modern PC system with three physical disk drives, named A, B, and C. A and B are usually floppy disk drives, and C is usually the hard disk drive. Drivers for these physical devices are provided with MS-DOS in one of the system files, IO.SYS, which is automatically loaded when MS-DOS boots [DUN88]. As a user-installed driver, XFSDRV is loaded at boot time by an entry in one of the system configuration files, CONFIG.SYS. As it is being loaded, MS-DOS examines some attribute information in the executable code of XFSDRV and recognizes, among other things, that the driver will control a removable media disk device. The operating system then assigns the next available block device identifier, in this case D, to the new device. From this point on MS-DOS can interact with the D drive in fundamentally the same way it interacts with A and B. Therefore, as long as XFSDRV faithfully emulates the behavior of a standard floppy disk device, MS-DOS will not be able to distinguish any difference between the physically attached floppy disk devices of the host system and the D floppy disk device emulated by XFSDRV.

XFSDRV has purposely been designed so that it appears to MS-DOS to be a removable rather than fixed disk device. Switching diskettes in a floppy drive is analogous to mounting a new server disk on the D device using XFS. MS-DOS defines a device service request called "mediacheck" which it uses to determine whether the media in a floppy drive has been changed by the user. Each time the user goes through the XFS *mount* process, XFSDRV notes this as a change in its "media". Because the driver has defined itself as a removable media device, MS-DOS precedes every new attempt to access the D device with a mediacheck request. On the first mediacheck after a new mount, XFSDRV will respond by reporting that the media in the D drive has changed. MS-DOS then initiates a series of device service requests on XFSDRV to determine the operating parameters of the disk that is now in the drive (capacity in sectors, location of directory entries, etc.). It is this mechanism that enables users to mount successively disks of various sizes and formats using the single D device. Because XFSDRV always reports that the media in drive D has changed after a mount, MS-DOS is alerted to take the steps necessary in order to reorient itself to the new disk configuration.

Every user level file operation is decomposed by MS-DOS into one or more device service requests. Each DSR is serviced atomically by a device driver. For example, issuing an `fopen("A:\myfile.txt", "r")` library call from a 'C' application will result in DSR's to determine whether the A drive has a diskette in it, (and if so whether the diskette has been changed since the last access), followed by a series of disk sector reads to load the disk directory in order to locate a particular file. When a file operation is directed to a physically attached disk drive, as in this case, each DSR is passed by MS-DOS to the IO.SYS resident driver that controls the target device. The driver resolves the requests using the ROM BIOS of the PC, which provides the lowest level programmer interface to the hardware of the machine.

When the target of the file operation is the XFS device, as in `fopen("D:\remote.dat", "r")`, the device service requests generated for the operation are passed to XFSDRV, the driver that controls the D drive. XFSDRV, now functioning as the client, packetizes each DSR into an individual XFS transaction and uses XTP to send them one at a time to the server. After each transaction the client blocks and waits for a response from the server. As it receives each response the client passes back to MS-DOS the completion status of the DSR and any other information required by the particular request. Control is then returned from XFSDRV to MS-DOS.

4.2. XFSDRV thread of control

It is important to note that the server responds asynchronously to XFS transactions from the client. At any time when it is not processing a transaction, the single thread of control in the server PC remains under the control of MS-DOS. However, because of potential file system conflicts between the activity of a local user and the "background" processing of XFS transactions by XFSDRV, a local user of the server PC should not attempt to write to the disk that has been mounted by the XFS client. In other respects the server PC is able to concurrently function as an XFS server and a normal, single-user MS-DOS computer.

The processing cycle described above indicates that there are two paths that the single thread of control in the PC can take into the code of the XFSDRV driver:

1) Control can pass from the top of the system (i.e. from MS-DOS) down into the driver as the result of a device service request, or

2) Control can pass from the bottom of the system (i.e. from the hardware interrupt controller) up into the XTP engine embedded in the driver as the result of a hardware interrupt generated by the arrival of a MAC frame at the network interface.

Path two allows us to generate the illusion of multi-tasking between MS-DOS and XFS on the server PC.

Our XTP User Interface provides for event-driven call-backs to the user application. XFSDRV exploits this feature in order to prevent blocking within the server as it waits to receive an XFS transaction from a client. At initialization time, XFSDRV registers the address of its main transaction processing function as the call-back for the arrival of a complete XTP message. Through path two the XTP engine receives control as each MAC frame arrives from the network interface hardware. When it recognizes that an entire message has arrived, the XTP engine transfers control to the call-back address. At this point control has entered the higher level XFS transaction processing code within XFSDRV. This code performs the processing necessary to carry out the XFS transaction specified by the contents of the message received from the client. After this processing is complete the server generates a response message which is transmitted to the client, and control in the server PC is returned to MS-DOS.

5. XTP in XFS

5.1. XTP services

As a minimum, XFS requires that its underlying communications service provide a duplex channel that guarantees reliable, end-to-end delivery of messages between the client and server systems. Because each XFS transaction directly results in the low level manipulation of a disk file system, error free delivery of the transaction data by the communications service is critical. XTP, TCP, and ISO TP4 can all effectively provide this type of end-to-end reliable service. However, the implementation of the communications service within an MS-DOS device driver, combined with the design goals of XFS and the operation of the MS-DOS file system, places additional demands on the transport protocol within XFS. We have employed the unique rate and burst control facilities of XTP to meet these demands.

5.2. Rate and burst control

XTP's rate and burst control can be used to reduce the problem of buffer overruns, which can be significant when the source traffic is bursty and the amount of buffer space available at the receiver is limited, both of which are true in XFS. Because the XTP connection opened by XFS is fully reliable, XTP's error recovery mechanisms in the receiver recognize that data is missing after a buffer overrun and automatically generate a request for retransmission of the lost data. These retransmissions substantially reduce the aggregate throughput of the connection. The likelihood of buffer overruns increases when the processor speed of the transmitter exceeds that of the receiver. This is becoming a significant issue for MS-DOS systems, where MIPS ratings across the range of Intel processors, from the original Intel 8086 to the recently announced Pentium, span at least two orders of magnitude.

We have observed the potential utility of rate and burst control in XFS when operating between a 20 MHz 80386 and a 33 MHz 80486 system over Ethernet with 8 MAC buffers. In this case, the 80486 server system floods the slower 80386 client, resulting in dropped packets and requests for retransmission by the client. With no rate and burst control, the aggregate file system to file system throughput of this connection is 896 Kbits/sec. By enabling and tuning rate and burst control for this particular connection, we can achieve a throughput of 1.96 Mbits/sec. This matches the peak performance of a version of XFS operating between these hosts that generates no retransmissions, but uses 16 MAC buffers (i.e., an additional 12,000 bytes of buffer space in XFSDRV). When operating between matched 80486/33 systems the XFS receiver is fast enough to prevent buffer overruns, so rate and burst control are not required on this connection.

The value of the rate and burst parameters for an XFS connection can be set by the user through the XFSCTL *throttle_client* and *throttle_server* commands. In addition, default values for rate and burst can be defined as part of the XFS peer name table, so that when a connection is opened, the client and server set their values of rate and burst based on the identity of their peer in the XFS connection.

5.3. The MS-DOS "kernelization" of XTP

Because it is an integral part of a device driver, the implementation of XTP used in XFSDRV does not have access to operating system services that would normally be available to a user-level application. The version of XTP that is integrated into XFSDRV has been modified to remove all calls for MS-DOS system services. For instance, one the most critical system services used in our implementation of XTP is dynamic memory allocation. In our new driver implementation of XTP, all calls for dynamic memory services are handled by a custom internal memory manager that performs allocations from a statically defined segment of memory reserved within the executable image of the driver. The whole process of integrating XTP into the XFSDRV device driver can be viewed as an MS-DOS "kernelization" of the XTP software protocol engine.

6. Performance

The primary design goal of XFS was to provide the *Fluency* system with transparent access to a remote file system using XTP, FDDI, and Intel 80486 PC's. To provide this transparent access, XFS must be capable of reading and writing the FMI byte stream to or from the remote device at a rate sufficient to support the full motion multimedia data stream produced by *Fluency*. The VideoPad Editor provides a range of user-selectable bit rates for recorded streams, ranging between 384 Kbits/sec and 2.0 Mbits/sec. We have measured the throughput of XFS while reading and writing 6 MB files, the equivalent of approximately 24 seconds of FMI motion video recorded at 2.0 Mbits/sec. As shown in Table 1, XFS can support the 2.0 Mbits/sec rate over FDDI, and can provide read access at this rate over Ethernet.

Client Access ->	READ	WRITE
Ethernet	2.32	1.62
FDDI	4.86	2.52

Table 1. XFS Throughput (Mbits/sec)

The theoretical peak throughput of XFS is limited by the hard disk device of the PC system. The disks used for these measurements have a sustained throughput of 7.6 Mbits/sec, so XFS can read the remote device over FDDI at 63% of the disks effective rate. XFS writes from the client to the server involve three data copies whereas reads require two; this additional data copy explains the lower throughput measurements for write accesses.

7. Future work: video teleconferencing

The successful application of XFS to the real-time transfer of Fluent FMI files between remote hosts demonstrates that our software implementation of XTP is capable of providing a transport level service that satisfies the throughput and latency constraints imposed by a full motion multimedia byte stream. We intend to apply the experience gained during the XFS/Fluency video mail project by designing and implementing a PC based video teleconferencing system using XTP and specialized frame grabber and video compression hardware. We have identified three functional requirements that a compression system considered for use in a teleconferencing application must meet. Such a system must allow the application software to:

1) control the data rate of the compressed bit stream processed by the hardware,

2) directly read and write the raw compressed bit stream to and from the compression engine, and

3) quickly switch the operating "direction" of the JPEG hardware between compression and decompression.

After surveying the marketplace, two PC video compression systems were identified that met these requirements: the Intel *ActionMedia II* (marketed by IBM) and the *Visionary* system from Rapid Technology Inc. (Williamsville, NY). *Visionary* was selected, primarily because the application development toolkit provided by the vendor includes support for both Microsoft Windows and MS-DOS. Like the Fluency VSA-1000, *Visionary* consists of two 16 bit ISA bus boards that perform A/D conversion of NTSC video signals and JPEG compression of each digital video frame.

Rapid Technology provides a full-featured 'C' language API that allows the programmer to control the operation of the *Visionary* board set. It includes functions that control the dimensions of the digital frame passed to the compression engine and the coding and quantization tables used within the JPEG algorithm. This interface enables the developer to trade lower video quality for reduced data rate. This control is critical in a networked teleconferencing application, where the rate that data is emitted by the compression system must be tuned so as not to overwhelm the underlying communication system.

Unlike the MPEG [LEG91] and H.261 [TUR93] standards, JPEG does not specify any interframe (temporal) compression; each frame of video can be compressed, packetized, transmitted, and decompressed with no dependence on neighbor frames. The *Visionary* interface provides calls to initialize the compression and decompression of individual frames, so the operation of the *Visionary* hardware can be activated from the underlying communications application based on the presence of an incoming compressed frame or available buffer space for an outgoing frame. From the user level, data is transferred to and from the JPEG engine via a 16-bit register that is I/O-mapped onto the ISA bus. Therefore, the rate at which frames of compressed video can be transferred between the communications application and the compression hardware is limited only by the bandwidth of the 8 MHz 16 bit ISA bus.

We believe that this hardware/software system provides the necessary hardware and software functionality such that, when integrated with the reliable low latency transport service provided by XTP, we can achieve a 10 to 12 fps duplex teleconference service over an FDDI network.

8. Summary

In this project we developed a video mail distribution system based on personal computers, MS-DOS, FDDI, and the Xpress Transfer Protocol. Our Xpress File System provides a general purpose file sharing capability which we use to support multimedia capture, transmission, storage, and retrieval. The throughput and latency characteristics of XFS are such that a human observer is unable to distinguish whether the multimedia bit stream is being replayed from the PC's local hard disk or from a remote disk which is being accessed via XFS using XTP and FDDI.

Acknowledgment

The authors gratefully acknowledge the technical cooperation of Dr. Kevin Vest at Science Applications International Corporation (McLean, VA), and the financial sponsorship of both SAIC and Virginia's Center for Innovative Technology (Herndon, VA). We appreciate the technical support provided by Dr. Robert Simoncic and Mr. James McNabb, both of whom are Research Scientists in the Computer Networks Laboratory, and the additional help of Mr. Bert Dempsey.

References

[DUN88] R. Duncan, General Editor, *The MS-DOS Encyclopedia*, Microsoft Press, 1988.

[FLU92] Fluency Product Manual, Fluent Machines Inc., Framingham, MA, 1992.

[LAI92] R.S. Lai, *Writing MS-DOS Device Drivers*, Addison-Wesley, 1992.

[LEG91] MPEG: A Video Compression Standard for Multimedia Applications, Communications of the ACM, April 1991.

[MIC91] *Microsoft MS-DOS Programmer's Reference*, Microsoft Corporation, 1991.

[STR92] W.T. Strayer, B.J. Dempsey, A.C. Weaver, *XTP: The Xpress Transfer Protocol*, Addison-Wesley, 1992.

[TUR92] T. Turletti, "H.261 Software CODEC for Videoconferencing Over the Internet", Report 1834, Institut National de Recherche en Informatique et en Automatique, France, January 1993.

[WAL91] G.K. Wallace, *The JPEG Still Picture Compression Standard*, Communications of the ACM, April 1991.