A Video Conferencing Testbed on ATM: Design, Implementation and Optimizations

*D. Saha** *D. Kandlur*† *T. Barzilai*† *Z. Shae*† *M. Willebeek-LeMair*†

Abstract

This paper describes our experiences with the design and implementation of a very high-end video conferencing testbed on an ATM network. Our system is built on an IBM RISC System/6000 equipped with prototype hardware for video and audio capture and compression, and an IBM 100 Mb/s ATM adapter. In our early experiments we used UDP/IP running over ATM Adaptation Layer 5 (AAL5) for data transfer between peers. Our initial experiences with the system indicated that the overall system performance did not match our expectations even though most of the video, audio, and network processing was performed in hardware. A thorough profiling of the system revealed that the protocol processing and data handling overheads in the end-host are responsible for the poor video/audio quality. Based on these observations, we have proposed and implemented changes to the protocol data path that can significantly improve the performance of the system. Although we discuss our solution in the context of a video conferencing application, our approach is general and can be applied to many other applications. It is particularly useful for applications that are required to handle large volumes of time-critical data, such as multimedia servers.

1 Introduction

The emergence of ATM networks has paved the way for a new generation of distributed multi-media applications, such as video conferencing, video on demand, tele-medicine, etc. These applications require hundreds of megabits of time-critical data to be moved between devices and processes. The protocol processing architectures of current operating systems were designed, however, for low-speed connectionless networks and non-time-critical data oriented applications. Therefore, it is not surprising that they fail to meet the challenges of a new generation of networks and applications. In order to realize the potential benefits that ATM offers, such as high bandwidth and quality of service support, the network interfaces of the current generation of operating systems must be enhanced. In this paper we describe our experience in optimizing the protocol data path at the end-host in the context of a video conferencing system running on an ATM network.

Our experiments are based on the Multimedia Multiparty Teleconferencing (MMT) System, a prototype desktop collaboration system developed at the IBM T. J. Watson Research Center [6]. We have developed a prototype adapter to perform video and audio capture, compression and decompression in hardware [3]. This adapter can support compression/decompression of up to 30 frames per second in real-time. The MMT desktop conferencing system has been developed on IBM PS/2 machines [5] and is deployed in the Aurora Gigabit Testbed. Our objective was to migrate this system to an IBM RS/6000 and provide a full motion, high resolution, video conferencing system running over ATM links. We started with a base system that used UDP/IP running over ATM adaptation layer five (AAL5) for data transfer between the peers. However, despite the hardware support for video processing, this base system did not meet our performance expectations.

In order to pin-point the bottleneck we analyzed the data paths in the end systems. In the configuration described above, on the transmitting side, the compressed video and audio data generated by MMT is first read into an application buffer and then sent to the destination via UDP sockets. Similarly, on the receive side, data from the source is read from the socket buffer into a user buffer and then delivered to MMT for decompression and playback. In this architecture, video/audio data is copied multiple times before being transmitted on the network. At the receiving host it is again copied several times after being received from the network interface. By profiling the system, we observed that this data copying has a larger impact on the system performance.

With the view of improving system performance, we then explored a new applications interface especially suited for an ATM environment. ATM provides

*Department of Computer Science, University of Maryland, College Park, MD 20742. The author is supported in part by NSF grant number CCR-9318933 and IBM equipment grant.

†IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, USA.

specific mechanisms for connection management, flow control, congestion avoidance, and segmentation and reassembly. With these mechanisms in place, similar functionality at higher-level protocols such as UDP/IP becomes redundant. ATM also provides application-to-application virtual connections. This makes it possible to set up a device-to-device (MMT adapters in peers) direct data path bypassing the protocol stacks. In doing so, we not only avoid data copying and redundant processing, but eliminate data copies due to the crossing of domain boundaries (User to Kernel space and vice versa). In the rest of this paper we describe details of the data path optimizations and changes required in the operating system interfaces to support these optimizations. Although we describe our work in the context of the video conferencing system, our approach can be applied to other applications, in particular, those which require high bandwidth and timely delivery of data.

This paper is organized as follows. In Section 2 we describe the hardware and software architecture of the video conferencing system. In Section 3 we profile the baseline system which uses the traditional network interfaces. Section 4 is devoted to the details of data path optimizations, wherein we also discuss the changes in the operating system interface required to support these optimizations. We conclude in Section 5.

2 System Description

The system platform for our experiments was the IBM RISC System/6000 (RS/6000) equipped with an IBM Turboways 100 ATM network interface, and the MMT prototype adapter. The ATM adapter implements the ATM adaptation layer 5 (AAL5), which is responsible for all segmentation and reassembly of datagrams and detection of transmission errors and dropped cells. The MMT adapter performs video and audio capture and compression/decompression. The MMT adapter is also equipped with an on-board Ethernet interface, but that is not used in this study. We use AIX 3.2.5 [1] for our experiments. In the following sections we describe the hardware and software architecture of the system.

2.1 Hardware Architecture

Like most modern workstations, in the RS/6000 system the CPU and the main memory are connected through the system bus. The I/O adapters, including the ATM adapter and the MMT adapter sit on the Micro Channel I/O bus. Data transfer from the I/O adapters to the main memory and vice versa goes through the I/O controller (IOCC). Data transfer on the Micro Channel can be in strides of 8, 16 or 32 bits with a cycle time as low as 20ns. In the following sections we will describe the architecture of the MMT and ATM adapters.

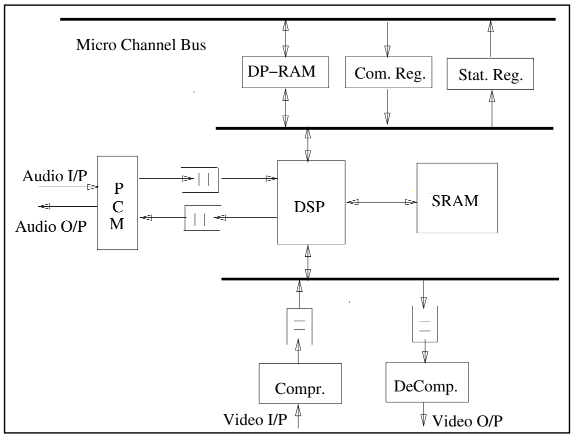

Figure 1: MMT Adapter.

2.1.1 MMT Subsystem

The MMT adapter (figure 1) consists of a video/audio capture (VAC) subsystem and a compression/decompression (CODEC) subsystem. The VAC subsystem accepts NTSC video and microphone or line audio input and generates YUV422 digital video and PCM/ADPCM audio and vice versa. The CODEC subsystem accepts YUV422 digital video at the video input for compression, and generates YUV422 digital video at the video output after decompression. It supports full-duplex real-time video compression and decompression at frame rates up to 30 frames/second. The CODEC subsystem accepts 64Kbit/sec PCM and 32Kbit/sec ADPCM digital audio at the audio input, and generates the same format of the serial audio stream at the audio output. The whole system is controlled by a dedicated DSP[2] processor. The DSP has access to a 256 Kbyte SRAM and a 16 Kbyte dual port memory (DP-RAM). The SRAM is used for smoothing and mixing of video and audio streams. The dual port memory is used to communicate with the system.

The adapter currently supports the ISO Motion-JPEG [12] standard. The video processing unit consists of two motion JPEG engines, one for compression and the other for decompression. They can support

[1]AIX is an UNIX like Operating System.

[2]DSP stands for Digital Signal Processor.

different frame rates and resolutions. The video input is fed through the video frame rate control logic to the compression engine. The data rate of the compressed data stream can be controlled by programming the quantizer to as low as 128 Kbits/sec and as high as 10 Mbits/sec. The CODEC is capable of mixing several video streams [11] in the compressed domain and presenting these in multiple video windows. The CODEC converts a PCM/ADPCM audio data stream into 8 bit parallel data for transmission. The on-board DSP processor uses two buffer areas in the SRAM to smooth out the traffic jitter and perform audio mixing. To guarantee continuous playback, the first audio packet received from a given source is held for a predefined delay period. This delay period can be adjusted to compensate for possible network jitter. A detailed description of the MMT hardware can be found in [8].

2.1.2 Turboways ATM Interface

The IBM ATM adapter [4] is responsible for performing the AAL5 functionalities. It features a dedicated i960 processor and a specialized chipset to handle AAL5 segmentation and reassembly in hardware. It handles four different types of traffic – signalling and local management traffic, high priority or delay sensitive traffic, low priority or delay insensitive traffic, and best effort traffic. There are three parameters associated with each traffic type – peak cell rate, sustainable cell rate, and burst size. Each virtual connection can be two way or one way. If one way traffic is specified, then direction must be specified, transmit-only or receive-only. Receive only traffic does not need any bandwidth reserved. For transmit only or two way traffic there is need to allocate bandwidth. The adapter can support up to 1024 connections.

The segmentation and reassembly (SARA) engine consists of a segmentation unit (SARA-S) and a reassembly unit (SARA-R). SARA-S has eight rate queues. Each rate queue has an associated peak rate of transmission. Thus there can be up to eight peak rates. The peak rates can be programmed in a set of rate queue registers. The sustainable cell rate and burst length parameters for each connection are stored in a VC table.

2.2 Software Framework

In this section we briefly describe the software architecture of the video conferencing system.

2.2.1 Application Structure

The conference management application consists of several components: the Conference Interface (CI), the Multimedia Conference Server (MMS), the Conference Control Unit (CCU), the Video/Audio Support Unit (VASU), and the shared workspace support. The CI provides the graphical user interface for the application. There is only one active instance of MMS in a workstation and it keeps track of all the users who may coexist in the workstation. The MMS creates a CCU responsible for managing a conference. Since a user may participate in multiple conferences simultaneously, the MMS maintains information regarding all the active instances of the CCU. The CCU is the heart of the conference management system. The primary function of the CCU is to setup and maintain a conference. This includes preliminary invitation to a group and accumulating acceptance, setting up network connectivity for control as well as media communication, and modifying network connectivity based on changes in the mode of participation of the participants, network condition, etc. The VASU handles the video/audio devices and controls the flow of multimedia data through the system. Hence, in this paper, our discussion is focused on the VASU and its related components – the video/audio device and the network device.

2.2.2 Video-Audio Device Control

The VASU controls the MMT through the MMT device driver. The MMT device driver interface follows the well-known open-close-read-write-ioctl UNIX paradigm. We have implemented several device specific ioctl calls. These include registering user processes with the device so that asynchronous call-backs can be made to the appropriate process when data is ready in the device, or when the device is ready to accept data. There are ioctl interface to register asynchronous exception handlers. The read, write, open and close interfaces are standard.

2.2.3 ATM Device Control

The ATM device software consists of several sub-layers in which the lowest layer interfaces with the ATM adapter and the highest layer interfaces to the AIX network subsystem. Currently, the standard interface for the ATM device is the IP network interface and the device supports the classical IP over ATM model [9]. In addition to this IP interface, the ATM device driver also provides a low-level UNIX device interface. Hence, the VASU can access the ATM network using either one of these two mechanisms. While using the low-level interface, the device is opened via the open system call. Before data can be sent or received (write and read respectively), virtual channels have to be established. This is done via ioctl calls. There are ioctl interfaces for opening and closing connections. In opening a VC its traffic characteristics (peak cell rate, sustainable cell rate, burst length) and other connection parameters (simplex/duplex, service priority, AAL type) are specified. In addition, there are ioctl calls for resetting the sender and receiver entities for a certain VC. Note that a VC can be opened from an application and application-to-application data transfers can be performed.

3 Profiling the Base System

In our experiments we started with a base system in which the VASU is connected to its peer through a standard network interface. In this configuration, the stations are connected directly to each other without any intervening ATM switch. This setup is sufficient since our focus is on identifying and eliminating performance bottlenecks in the end hosts.

On the transmitting side, audio and video data is captured, digitized and compressed by the MMT. Compressed data is packetized by the DSP and an interrupt is sent to the driver indicating that data is ready to be read. The driver, in turn, sends a signal to the VASU. The VASU, upon receipt of the signal, reads the data and sends it over the UDP/IP socket connection to its peer. Likewise, on the receive side the VASU receives data on the UDP socket. Once data is received from the network interface, the VASU writes it into the MMT buffer using the write system call provided by the MMT driver.

In the following we present the latency of different portions of the transmit and receive data paths. The measurements have been taken using the real-time clock which is an integral part of the RS/6000 architecture. This clock can be accessed by any process by reading two 32-bit clock registers with microsecond granularity. We use a two-instruction assembly language routine to read the clock registers with negligible overhead.

3.1 Transmit Processing

The transmit data path is illustrated in Fig. 2. The latency on the transmitting side can be divided into two major components — i) copying data from the MMT buffer to the user buffer in the main memory and ii) transferring data from the user buffer in the main memory out to the network. The first part, reading from the MMT buffer, can be further sub-divided into a fixed setup overhead and a variable part which depends on the volume of the data transfer. The second part is a function of the protocol processing overheads, and the cost of multiple data copies.

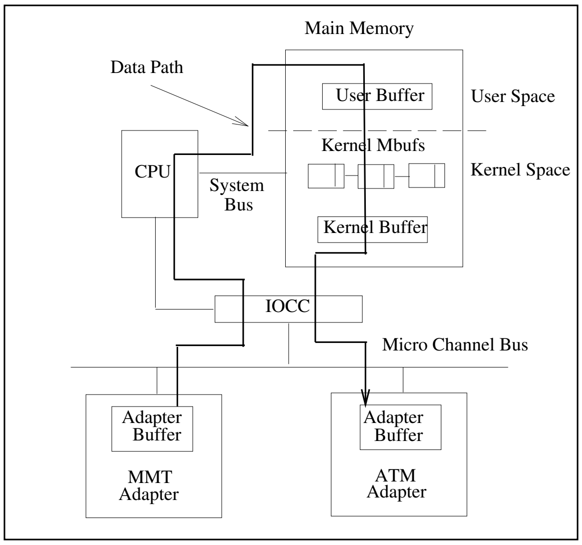

Figure 2: Transmit Data Path.

Figure 3 shows a plot of these measurements. These measurements have been taken on an RS/6000 Model 530H with 32 MB memory and a 33 MHz processor. The Maximum Transfer Unit (MTU) for the ATM interface has been set to 9180 bytes, which is the default for classical IP over ATM, so packets are not fragmented in the IP layer. MMT generates variable size packets and its MTU size is an adjustable parameter. For most of the videoconferencing experiments described in this paper the MMT MTU size was set to the largest possible size – 3220 bytes. As shown in figure 2, the data is copied several times on its way out to the network. First, data is copied from the user buffer to a chain of mbufs in the kernel space as part of the socket-UDP/IP processing. It then passes through UDP and IP protocol stacks and makes its way into the ATM device driver. Here, it is copied into a contiguous memory buffer (Kbuf) in preparation for transfer to the ATM adapter. Finally, it is copied into a buffer on the ATM adapter (Cbuf) using the DMA facilities of the adapter.

As apparent from the table, the data copying cost is quite different depending on the source and the destination. When data is copied from main memory to main memory, as in the Mbuf to Kbuf copy, the copying cost is small. However, the copy cost is large when copying is done from adapter buffers to main memory and vice versa. The reason for this disparity is that in the second case data has to pass through the I/O controller which turns out to be the system bottleneck. Figure 3 also shows an unusual dip in the socket-UDP/IP processing costs when the packet size changes from 512 bytes to 1 Kbyte. This anomaly can be attributed to the strategy for allocation and use of

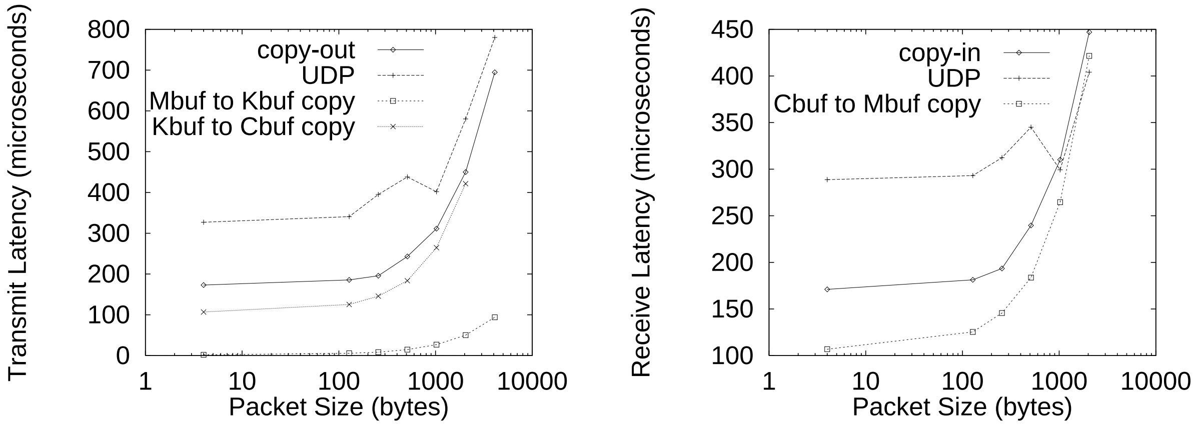

Figure 3: Transmit and Receive Side Latencies (in microseconds).

Mbufs. For small packet sizes a chain of Mbufs is used, whereas a cluster Mbuf is used for larger packets [10].

3.2 Receive Processing

The receive side data path is similar to the transmit data path in reverse. Once a complete packet is received it is DMAed into a chain of Mbufs in the kernel space in main memory. It is then copied into the user buffer of the VASU. Finally, it is copied from the user buffer into the buffer on the MMT card. Figure 3 show the processing costs on the receive side for different packet sizes. The latencies measured here are very similar to the latencies on the send side and there is, once again, a discontinuity in the latency for the socket receive processing from a packet size of 512 bytes to 1 Kbyte.

It is apparent from the description of the data path that there is room for improvement. The low-level ATM driver interface presents one possibility, and the VASU could use this interface to send/receive data. However, a closer examination reveals that this approach would only eliminate the UDP/IP processing and would not affect the bulk of the processing overheads. The number of data copies and the user-kernel domain switches would remain unchanged when this low-level interface is used. In the next section we address this issue of optimization of the data path in detail.

4 Data Path Optimization

In the last section we have seen that the data path from the device where data is generated/consumed (MMT in this case) to the network interface (ATM in this case) is very long, and can result in low throughput. In this section we examine alternative data paths and propose changes in the protocol processing architecture to enable the applications access to the optimized data paths.

There are different ways to move data from the MMT adapter to the ATM adapter. Four possible alternatives are shown in figure 4. The first choice is to have a direct hardware link between the MMT and ATM adapters. The second alternative is to copy data over the I/O bus (micro channel in this case) from the MMT buffer to the ATM buffer. A third alternative is to copy data through the main memory. In each of these three cases the data transfer takes place within the kernel and there is no domain boundary crossing. The fourth alternative is to move data through the application – as described in the last section. All of these alternatives have their advantages and disadvantages.

1. *Direct Hardware Link Transfer.* While this approach is very efficient it is also very inflexible. Data transfers between the CODEC (MMT) and the network adapter may occur without interference by the main system. This is advantages since existing applications may run concurrently without being affected by the audio and video data transfers. However, it adds complexity to the network interface which needs to identify incoming data to determine whether it should be passed to the system or across the direct interface to the CODEC. Similarly, either the network interface or the CODEC must add appropriate packet headers to outgoing data. Finally, this approach requires that the CODEC and communication card have a direct hardware connection between them and limits the solution to those communication and CODEC adapter cards which are equiped with the direct interface.

2. *Card-to-Card Bus Transfer.* The second approach is the next best in terms of performance. It al-

lows data to be transferred between the network interface and the CODEC adapter directly across the I/O bus. However, to do so, packet headers must either be processed in the network interface adapter and CODEC, or passed up to the system for processing to identify the data packets and transfer them accordingly. This ties the network interface and CODEC drivers together very closely. It also requires sufficient buffering on both adapters to be successful.

3. *Main Memory Data Transfer.* Moving data through the main memory, as proposed in this approach, eliminates the problem of buffer management on the adapters. However, it reduces performance due to the extra data copies it requires.

4. *Application Data Transfer.* The fourth approach is based on the traditional UNIX architecture for distributed applications, and is used in the baseline system described in the last section.

Depending on the application, any one of these data paths may be preferred. If performance is the primary concern, the first or second approach may be desired. On the other hand, the third or fourth approach may be preferrable in many instances where portability and composability take priority. It is therefore important for the operating system to provide interfaces which enable the application to choose the appropriate data transfer approach. An elegant solution to this problem is to provide a network interface that permits application-specific protocol processing, since it can accommodate the diverse needs of different applications.

The current protocol processing architecture at the end-hosts provides a common interface to all applications. With ATM providing application-to-application virtual connections it is possible to design interfaces where the common protocol stack is extremely shallow and most of the processing is done in an application specific manner. We have used this specific feature in optimizing the data path of the video conferencing application.

4.1 Application Specific Interface

One of the important functions of network and transport level protocols is the multiplexing and demultiplexing of different sessions onto a single link-level connection. This multiplexing/demultiplexing works well in the case of traditional data connections since they do not have explicit service requirements. However, in the case of new multimedia applications,

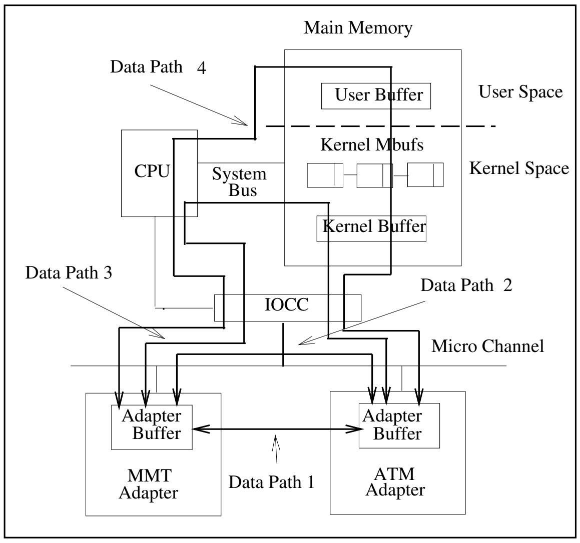

Figure 4: Alternative Data Paths.

each application may have its own requirements for delay, jitter, bandwidth, etc. In an ATM environment, the network provides the ability to setup connections (VCs) with specific QoS parameters. In this environment the network interface must provide the application with (a) facilities to request the setup of connections with QoS parameters, and (b) an efficient data path to the network. These goals can be achieved by decoupling the connection setup from the data transfer path. For the connection setup phase, we have designed extensions to the socket interface which enable applications to specify service parameters for connections. These extensions will be discussed in a forthcoming paper.

For the data transfer path, the traditional functions of multiplexing and demultiplexing are redundant when the VC is established for a specific application. The ATM link layer can use the VC to route incoming packets directly to the application. In this scenario the ATM interface is responsible only for the AAL5 processing, which includes segmentation and reassembly. Further processing of the data is done by application specific asynchronous handlers.

After the virtual connection has been setup, the application registers three asynchronous handlers with the ATM link layer for this VC. The link layer then uses these asynchronous handlers to direct data traffic on the VC to the application. This mechanism is illustrated below in a sample pseudo-code fragment.

Here, the receive function is called when the ATM driver has data for the user. The transmit function is called to indicate that sending data is now possible. The status function is called in response to a previously issued ioctl command. The ATM driver

```
int rc;
int atmfd;      /* ATM device file pointer */
struct ATMVCEXT {
int vchandle;  /* Virtual circuit handle  */
 int *rx_fn();
 int *tx_fn();
 int *sx_fn();
     } vcext;   /* VC handlers */


  vcext.rx_fn = rxhandler; /* Receive handler */
  vcext.tx_fn = txhandler; /* Transmit handler*/
  vcext.sx_fn = sxhandler; /* Status handler  */

    rc = ioctl(atmfd, ATMSETHANDLERS, &vcext);
```

also exports the address of a write function through an ioctl interface. This function accepts the buffer address and buffer length parameters and copies data directly into the buffer on the ATM adapter from either main memory or from the buffer on the CODEC adapter. A handle is also provided to this function to transmit data on the appropriate virtual channel.

4.2 Device to Device Transfer

With the modified interface described in the previous section it is possible to optimize the data path in the end-host. However, it still requires the crossing of domain boundaries and hence data copying. In order to avoid domain boundary crossing we could have taken one of several alternatives. In order to avoid domain boundary crossing we selected a simple approach — doing an in-kernel data transfer. We separated the data and control path for the video conferencing application. While control was still trusted with the application, we moved the data path inside the kernel [1, 2]. The MMT device driver was modified to communicate directly with the ATM driver. The MMT driver can open an ATM VC inside the kernel and data transfer can happen from the MMT driver directly to the ATM driver without going through the user buffer[3]. This data path eliminates two data copies. In the following we explain these optimizations in detail.

The MMT driver opens a virtual channel by using the **open()** system call for the ATM device. While opening the device the MMT driver registers the **status_handler()** and the **receive_handler()** with the ATM driver. These handlers are called whenever a status message has to be passed from the ATM driver to the MMT driver or whenever data is ready for MMT in the ATM card buffer. The MMT driver also obtains the address of the write function.

Now, in the send path, when the MMT card has data to send, it calls the write function of the ATM driver instead of copying it into the user buffer. The write function accepts the address of the memory buffer on the MMT card and the length of the buffer. It then DMAs the data directly into the ATM dual port buffer from where it is transmitted onto the network.

On the receive side, whenever an (AAL5) packet is completely received at the ATM interface it DMAs it to an mbuf chain in the main memory. It then calls the **receive_handler()** function registered by the MMT driver during the opening process of the device. This function copies data from the mbuf chain to the MMT buffer.

4.3 Optimized Data Path Latency

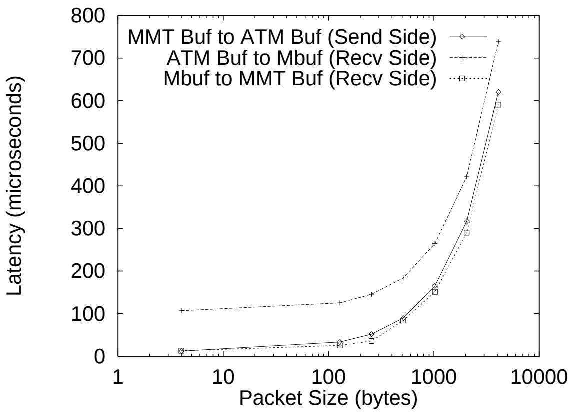

Figure 5: Latency for optimized data path (in microseconds).

The resulting latency for the optimized data path is shown in figure 5 for various data sizes. For comparison, figure 6 plots the latency for the base system and the optimized system. The latency of the send path involves the copying of data from the MMT dualport buffer to the buffer on the ATM card. The latency on the receive side involves the copying of data from the ATM card buffer to an mbuf chain in main memory and copying data from the mbuf chain to the dualport buffer on the MMT card. As such, there is room for further optimization on the receive side. We could potentially transfer data directly from the ATM adapter

[3] AIX supports a kernel interface that can be used to open files much like in an user application. It also provides facilities to create kernel processes which can be used, among other things, in a device driver.

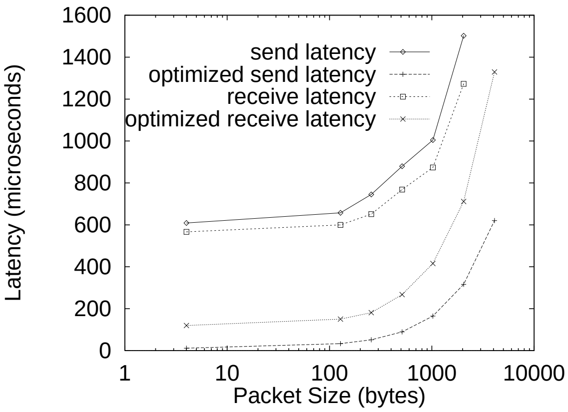

Figure 6: Comparison of latency for base and optimized data paths (in microseconds).

buffer to the buffer on the MMT card. This, however, is a little more complicated than it appears. If we do not use the mbuf chain in main memory we will have to tie up a buffer on the ATM card until it is copied to the MMT buffer. This might create buffer overflow problems. We are currently pursuing this optimization further.

5 Concluding Remarks

We have described our experience in the design and implementation of a very high-end video conferencing system on an ATM network. In our experiment we observed that, although ATM provides sufficient bandwidth, the operating system interfaces to the network are not optimized for high quality interactive conferencing. In general, traditional network interfaces have been designed for non-real-time data transfers and fail to exploit the benefits offered by ATM. In order to effectively support applications which require large volumes of time-critical data movement, the data paths in the end-hosts must be optimized to reduce the number of data copies to a minimum. The tradeoffs between various alternative data transfer approaches were presented along with results for an optimized path. The optimized path yielded a substantial improvement over the traditional data transfer approach.

The work presented in this paper addresses one aspect of the overall communication performance, namely that of data movement within the host. Other aspects in the communication process such as connection setup, virtual channel QoS support, etc., must also be considered when evaluating the overall performance. These issues are currently being explored.

References

[1] Calls and Subroutine Reference: Base Operating System, Vol. 1. *AIX Version 3 for RISC System/6000.*

[2] Technical Reference: Kernel and Subsystems. *AIX Version 3.2 for RISC System/6000.*

[3] MMTplus Functional Description. *IBM Internal Document*, May 1993.

[4] ATM Turboways 100 Adapter. *IBM Internal Document*, May 1994.

[5] M-S. Chen, Z-Y. Shae, D. D. Kandlur, T. Barzilai, and H. Vin. A multimedia desktop collaboration system. Research Report RC 17951, IBM Research, T. J. Watson Research Center, Yorktown Heights, NY 10598, May 1992.

[6] M-S. Chen, Z-Y. Shae, D. D. Kandlur, T. Barzilai, and H. M. Vin. A multimedia desktop collaboration system. In *Proceedings GLOBECOM 92*. IEEE, December 1992.

[7] M-S. Chen, H. Vin, and T. Barzilai. Designing a distributed collaborative environment. In *Proceedings GLOBECOM*. IEEE, December 1992.

[8] D. D. Kandlur, M. Willebeek-Lemair, T. Barzilai, Z-Y. Shae, and J-T. Liu. Supporting timing requirements of digital video and audio in desktop conferencing systems. Research Report RC 19479, IBM, March 1994.

[9] M. Laubach. Classical IP and ARP over ATM. Internet RFC–1577, January 1994.

[10] S. J. Leffler, M. K. McKusick, M. J. Karels, and J. S. Quarterman. *The Design and Implementation of the 4.3BSD Operating System.* Addison-Wesley, 1990.

[11] Z-Y. Shae and M-S. Chen. Mixing and playback of JPEG compressed packet videos. In *Proceedings GLOBECOM 92*. IEEE, December 1992.

[12] G. K. Wallace. The JPEG Still Picture Compression Standard. *Communications of the ACM*, 34, April 1991.