Protocol Architecture for Multimedia Applications over ATM Networks

Dilip D. Kandlur[*] *Debanjan Saha*[†] *M. Willebeek-LeMair*[*]

Abstract

At the data-link layer, ATM offers a number of features, such as high-bandwidth and per-connection quality of service (QoS) guarantees, making it particularly attractive to multimedia applications. Unfortunately, many of these features are not visible to applications because of the inadequacies of existing higher-level protocol architectures. Although a considerable effort is underway to tune these protocols for ATM networks, we believe that a new ATM specific protocol stack is essential to effectively exploit all the benefits of ATM. In this paper we describe the semantics of such a protocol stack, and discuss its advantages over traditional protocol architectures from the perspective of distributed multimedia applications. The performance impact of the new protocol architecture is experimentally demonstrated on a video conferencing testbed built around IBM RS/6000s equipped with prototype hardware for video/audio processing, and connected via ATM links.

[*]IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, USA.

[†]Department of Computer Science, University of Maryland, College Park, MD 20742.

1 Introduction

At the data-link layer, ATM offers a number of features, such as high-bandwidth and per-connection quality of service (QoS) guarantees, making it particularly attractive to multimedia applications. Unfortunately, higher layer protocols such as TCP/IP fail to extend these benefits to their applications. Although a considerable effort is underway to tune these existing protocols for ATM networks [5, 8], we believe there is considerable advantage to be gained by a new ATM specific protocol stack to effectively exploit all the benefits of ATM. In this paper we briefly describe the semantics of such a protocol stack and present experimental evidence of its impact on distributed multimedia applications.

Our design blends in the new protocol stack into the existing protocol structure by employing the widely used "socket" interface. To alleviate the problem of domain boundary crossing our design separates control and data flows. This design allows an application to control the data flow while delegating the data path to a more efficient device-to-device level. Application transparent data paths are often very useful in establishing device-to-device in-kernel tranfers in many multimedia applications where devices (e.g. camera and display), rather than processes, are the real end points of communication. The protocol processing ovehead is minimized by limiting the functionality of the common case protocol stack, which is explicitly designed to exploit the specific features of ATM networking. It does not duplicate any functionality provided by the underlying ATM and Adaptation layers. To enhance performance, connection identifier based demultiplexing is used to eliminate logical demultiplexing. We use connection specific handlers to complement the basic functionality provided by the common case protocol stack. These handlers can be registered at the time of connection set up, and can be effectively used to customize protocol processing on a per-connection basis. Connection specific processing provides multimedia applications with data channels customizable to satisfy their diverse service requirements.

The rest of the paper is organized as follows. In section 2 we describe the semantics of the protocol stack. In section 3 we present performance results to demonstrate the impact of the new protocol architecture on a video conferencing testbed. We conclude with section 4.

2 Protocol Architecture

Our network environment consists of hosts (end-stations) connected to a large ATM network. The network supports the setup of switched virtual connections and permanent virtual connections between end-stations. Our objective is to enable the development of new multimedia applications, while preserving existing data applications. Hence, it is necessary to support multiple protocol families on top of the ATM link interface. The traditional protocol families such as TCP/IP support packet data communication between applications that are not necessarily aware of the underlying ATM network. On the other hand, the native ATM protocol family supports the creation of application driven ATM virtual connections (VCs) to carry data with quality of service requirements. Moreover, the virtual connections may carry different types of traffic such as AAL1 circuit emulation, AAL5 sequenced packet, etc.

In this section we define different components of the protocol architecture. Our objectives

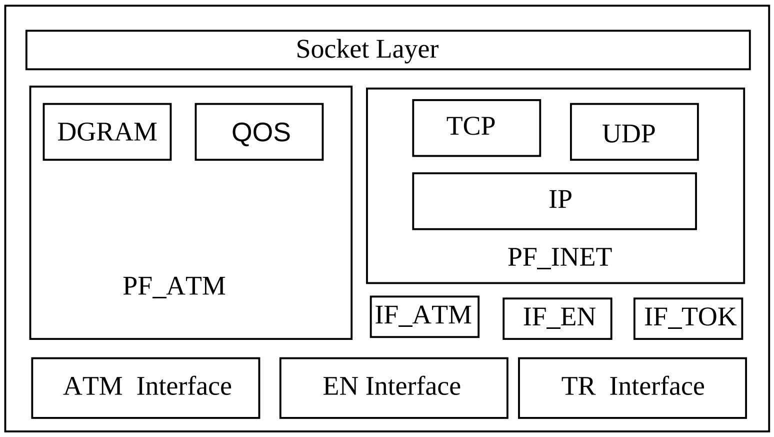

Figure 1: Protocol support for QoS.

are as following:

- Preserve compatibility with the current socket based application interface so that existing applications can run unchanged. Provide extensions to the interface for specification and negotiation of quality of service parameters.
- Separate control and data paths. This is particularly useful for data path optimizations in many continuous media applications.

Our approach is to blend native ATM support into the existing protocol structure as shown in Fig. 1. This is achieved by employing the widely used "socket" interface and defining a new protocol family specific to ATM. The figure shows two separate protocol paths: (a) the native ATM family, and (b) the IP protocol family.

Applications have transparent access to the ATM network using the PF_INET protocol family. The mapping of IP to the ATM data-link is handled by the ATM network interface (IF_ATM) using the IP over ATM protocol [3]. Applications also have direct access to ATM services using the ATM protocol family (PF_ATM). In the following we explain the application interface using the PF_ATM protocol family. The PF_ATM family uses ATM addresses for identifying endpoints for connections.

Addressing. One of the obstacles in the connection setup phase is the acquisition of ATM addresses, since it is unreasonable to expect applications and users to deal with 20-byte ATM NSAP (or E.164) addresses. We propose to exploit the Internet Domain Name Service (DNS) to provide naming support for applications. The DNS has been extended to support NSAP Resource Records [4] so that DNS servers can store NSAPs and client DNS resolvers can obtain the NSAPs by issuing queries based on the host name. Communication between the resolver and the DNS server will proceed as usual using the normal TCP/IP protocol stack.

Control. In keeping with the architecture of ATM, we have decoupled the control and data paths. In the traditional socket interface, control and data are handled on the same socket.

Control parameters are usually passed using `ioctl` or `setsockopt` function calls. However, these calls do not accommodate the upward flow of information from the protocol module to the application. More recently [7], the message structure used in the `sendmsg` and `recvmsg` function calls has been modified to include control fields, and these calls can be used to pass control information between the application and the protocol module. However, this multiplexing of control and data makes it difficult to construct applications in which the control information is handled by a different thread. For example, in a tele-conferencing application, the control part may be handled by the conference control entity, while the data is generated and consumed by a separate thread or by a hardware device. Therefore, we create a control socket that is distinct from the socket which handles the data flow.

The control socket is connected to the QOS protocol module in the ATM protocol family. This socket may be used to control one or more data sockets and it permits a bidirectional flow of information between the application and the QOS module. The QOS module can post messages to the application on the control socket reflecting changes in the connection state or in the network environment.

Datagram Service. Constant or variable bit rate multimedia traffic may be supported using a datagram service over AAL5. This service is well-suited for real-time multimedia applications since these applications are typically error resilient and employ forward error correction techniques. In the rest of this paper, we will focus on the datagram service for the ATM protocol family.

The ATM datagram socket differs from the datagram socket of the IP protocol family. An IP datagram socket may be used to send datagrams to multiple destinations, and also receive packets from multiple destinations based on the port number. On the other hand, our ATM datagram socket is closely associated with the semantics of an ATM virtual connection. Hence, a point-to-point connection would be bidirectional whereas a point-to-multipoint connection would be unidirectional. By associating a socket with a single virtual connection, it is possible to redirect the traffic on that socket to a device.

The following example illustrates the setup of an ATM datagram connection.

Source Station: The source establishes a data socket and binds it to a local port using standard system calls. This local port is used as an application identifier for the ATM connection setup signaling. As in TCP or UDP, the local port may be well-known (published) or assigned dynamically by the operating system.

```
data_sock = socket(AF_ATM,SOCK_DGRAM,0);
bind(data_sock,<local-atm-address,local-port>);
```

At this point, it uses the control socket to setup an ATM SVC to the destination. The information required for this setup are the endstation addresses, the application end-points, and the flow specification for the forward and reverse connections. The local end-point information is obtained by passing the data socket as a parameter.

```
ctl_sock = socket(AF_ATM,SOCK_QOS,0);
sendmsg(ctl_sock,<SETFLOWSPEC,data_sock,flow-spec,flow-spec-len>);
sendmsg(ctl_sock,<SETUP,data_sock,<dest-atm-address,dst-port>>);
```

Here, the `sendmsg` call is used to pass the control information structure since it differentiates between control and data fields [7]. Alternatively, the information could also be passed using `write` or `send`. The QOS module maintains connection state information for each connection that has been opened on the control socket. Once the connection is established, the QOS module registers the virtual connection handle in the data socket and informs the application by sending it a status message.

For a point to multipoint call, once a virtual connection has been established, new endpoints can be added or dropped by sending additional messages on the control socket.

```
sendmsg(ctl_sock, <ADDPARTY,data_sock,<new-dest-addr,new-dst-port>>);
sendmsg(ctl_sock, <DROPPARTY,data_sock,<dest-addr,dst-port>>);
```

Destination Station: At the destination, the application creates a listen socket and binds it to the local application port number. It also creates the control socket, places a request to listen for incoming call setup requests and waits for messages on the control socket.

```
listen_sock = socket(AF_ATM,SOCK_DGRAM,0);
bind(listen_sock,<local-atm-address,local-port>);

ctl_sock = socket(AF_ATM,SOCK_QOS,0);
sendmsg(ctl_sock, <SETRCVHANDLE,listen_sock>);
```

When an incoming call setup request is received, the QOS module posts the setup message to the appropriate application on the control socket. The application can accept the call using `accept` and at this point a new data socket is created for the incoming connection. The `accept` operation would involve the QOS module and result in the generation of an appropriate ATM call-accept message.

```
data_sock = accept(listen_sock, &remote-endpoint);
```

On the other hand, the application can reject the call by instructing the QOS module via the control socket.

```
sendmsg(ctl_sock, <REJECTSVC, data_sock>);
```

2.1 Implementation Architecture

Based on the design described above, we will briefly describe the control and data flows and the architecture for the sharing of the ATM data link between different protocols. Fig. 2 shows the protocol architecture in the context of the AIX [1] operating system.

[1]AIX is an UNIX like operating system developed by IBM.

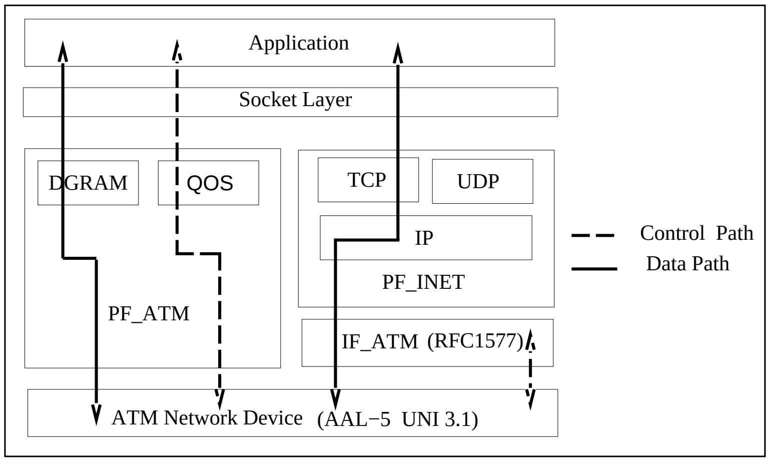

Figure 2: Control and data flows for ATM protocol family.

ATM Network Device. The ATM Network Device interfaces with the ATM hardware and presents a link-level interface to upper layer users. This device is responsible for maintaining virtual connections and providing mechanisms for accessing ATM signaling functions.

The ATM device may be opened by one or more entities (kernel or user) and the device is responsible for appropriate demultiplexing. Moreover, for kernel users, AIX provides an upcall mechanism whereby the user can register handlers with the ATM device drivers for functions such as transmit completion, receive message, and status updates. This mechanism can be exploited to (a) support the native ATM family as well as traditional protocol families, and (b) provide virtual connection based demultiplexing and connection specific data processing.

In the control path, incoming call setup messages are directed to the appropriate user module based on the BLLI information. Once the connection is established, the user modules can register connection specific handlers with the ATM device.

IP Interface. The IP protocol family is supported using the IF_ATM Network Interface layer. The network interface layer provides a connectionless data-link service to its users and it demultiplexes incoming data packets based on the logical link control (LLC) information. In the case of ATM, this interface layer adapts the connectionless LLC service to ATM virtual connections. For IP traffic, RFC1577 [3] defines a "classical" IP subnet mapping wherein this adaptation is localized to the data-link layer using a modified ARP mechanism, leaving the upper layer protocols and applications unchanged. In the figure, this is reflected by the absence of ATM signaling flows in the PF_INET family.

Native ATM Interface. The PF_ATM protocol family is supported directly on top of the ATM network device and bypasses the network interface layer.

The QOS module opens a signaling connection with the ATM device and establishes a handler for call setup messages. It directs incoming call setup messages to the appropriate application using the application port number information contained in the BHLI element of the setup messages. If no suitable application is found, the call is automatically rejected. Once a call has been successfully established, the QOS module inserts the resulting connection handle into the appropriate data socket maintained by DGRAM. It also installs handlers for transmit and receive operations for the new connection that point to the DGRAM module. However, it retains the handler for the status function so that connection control information can be received and presented to the application on the control socket.

For an ATM datagram connection the segmentation and reassembly functions for AAL5 are handled, with hardware support, by the ATM network device. For outgoing packets, the DGRAM module passes the appropriate connection handle along with the packet to direct the ATM device to send the packet on a particular virtual connection (VC). An incoming data packet is demultiplexed by the ATM network device based on the VC number and directed to the DGRAM module along with the connection handle representing the VC. The DGRAM module then uses the connection handle to place the packet into the appropriate socket buffer for the application.

3 Exploiting Native Mode ATM

One of the major bottlenecks in communication performance is the protocol processing overhead. A large component of this overhead stems from protocol redundancy in the existing layered protocol architecture, such as the PF_INET protocol family. In the PF_INET protocol family ATM is considered to be just another link layer no different from ethernet or token ring. Hence, to maintain the same interface to all the link layers the PF_INET protocol stack ignores the special features provided by ATM, and many of the functionalities are replicated in higher layers. The PF_ATM protocol family eliminates much of this redundant processing by keeping the protocol stack simple and exploiting features specific to ATM. For example, PF_ATM provides application-to-application direct virtual connections and demultiplexes application PDUs based on connection identifiers right at the network interface. This eliminates much of the demultiplexing stack in common protocol architectures such as PF_INET. In PF_ATM the common case protocol stack is limited to segmentation and reassembly functions only. The rest of the processing, when required, is performed on a per-connection basis using connection specific handlers. The use of a light weight protocol stack allows PF_ATM to keep protocol processing overhead low. Connection specific handlers complement the limited functionality provided by the basic stack using connection specific processing. As a result, PF_ATM can tailor its services based on the application need. It can provide fast data channels with latencies and bandwidth close to the physical limitations imposed by the hardware. It is also capable of providing data channels with sophisticated error control, flow control and other mechanisms. The other major benefit of the PF_ATM interface is the separation of control and data flows. This allows data transfer mechanisms to be fast and dumb, while the control mechanisms can be as complicated as necessary. We use this feature to establish device-to-device in-kernel data paths, significantly improving performance of many multimedia applications. The following experiments on a video conferencing testbed present a proof of the concept.

3.1 System Architecture

The conferencing system is based on an IBM RS/6000s equipped with an IBM ATM network interface, and an MMT prototype adapter. The ATM adapter [1] 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. The adapter is equipped with 2Mbytes of on-board buffer and a DMA master. It can support up to 1024 connections. The MMT adapter supports full-duplex real-time audio/video compression and decompression at video frame rates up to 30 frames/second. One video/audio stream is compressed while multiple audio/video streams (upto 32) may be decompressed simultaneously. The whole system is controlled by a dedicated DSP[2] processor. Further details of hardware architecture can be found in [6].

The conference software 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 conference communication structure consists of control paths between all participants as well as audio, video, and data paths emanating from each participant to other participants. The connections require reliable end-to-end delivery, with no stringent end-to-end delay requirements. The video and audio connections are considerably different. These are constructed as point-to-multipoint connections with specific quality of service (qos) characteristics. Hence, the application may set up the control connections using a traditional TCP/IP stack, while it would set up the audio and video connections using the PF_ATM extensions. The VASU handles the video/audio devices and controls the flow of multimedia data through the system. Further details of the software architecture can be found in [2].

3.2 Performance Results

In this section we compare the data path latencies of the PF_INET and PF_ATM protocol stacks.

In our first prototype implementation (figure 3) VASUs exchange video and audio data using UDP/IP (PF_INET) running over AAL5. 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 this mode of communication, exchanging one data packet requires two system calls (read from MMT and send to socket) on the transmitting side and two more system calls (receive from socket and write to MMT) on the receiving side.

In our second implementation, this time on PF_ATM stack (see Fig. 3), we separated the control and data paths. The VASUs still exercise control over the data transfer, but without

[2]DSP stands for Digital Signal Processor.

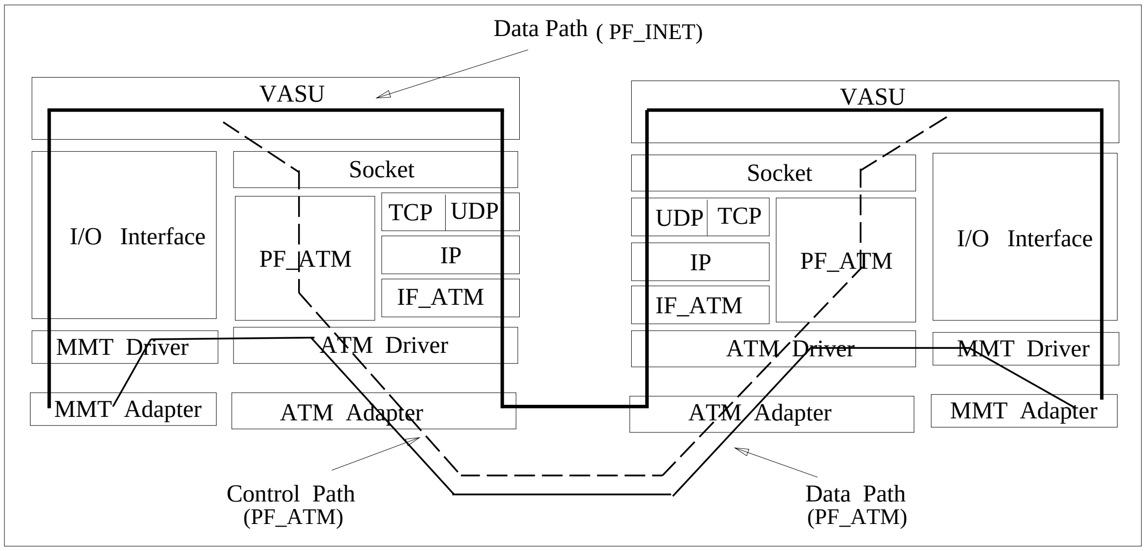

Figure 3: Control and Data Paths.

being directly involved in moving data. The data path connects the peer MMT adapters directly, and data transfers occur in-kernel in almost autonomous fashion. Now, in the send path, when the MMT card has data to send, it copies it directly into the buffer on the ATM adapter. On the receive side, whenever an (AAL5) packet is completely received on a VC terminating at the MMT, it is DMAed into an mbuf chain in the main memory. It is then copied into the dual-port buffer on MMT.

To establish the data path, VASUs first open data connections among themselves using the socket interface. The end point of the connection is then redirected to the MMT adapter by splicing the ATM upcall handlers for the VC opened by VASU with the MMT device handler. This requires, among other things, modifications and extensions to the MMT drivers. We extended the MMT driver to open and close ATM connections. The processing of incoming and outgoing MMT datagrams is then handled by connection specific handlers.

Separation of control and data enables application transparent data transfers. The in-kernel data data eliminates the system calls, and consequently both i) the overhead due to data copying across domain boundaries and ii) the cost of context switching. The latencies in the optimized data path reflects the cost of moving data from the MMT adapter to the ATM interface. Figures 4 and 5 compare the transmit and receive side latencies over PF_INET and PF_ATM stacks. These measurements have been taken on an RS/6000 Model 530H with a 32 Mbyte memory and a 33 MHz processor. The measurements were made using the system's real-time clock which is an integral part of the RS/6000 architecture. The results speak for themselves.

Note, that the gain on the transmit side is more than the gain on the receive side. This difference can be attributed to the asymmetry in the send and receive data paths in the PF_ATM based system. On the transmit side, data generated by MMT is copied directly into the network interface buffer. On the other hand, on the receive side, datagrams destined to MMT are first

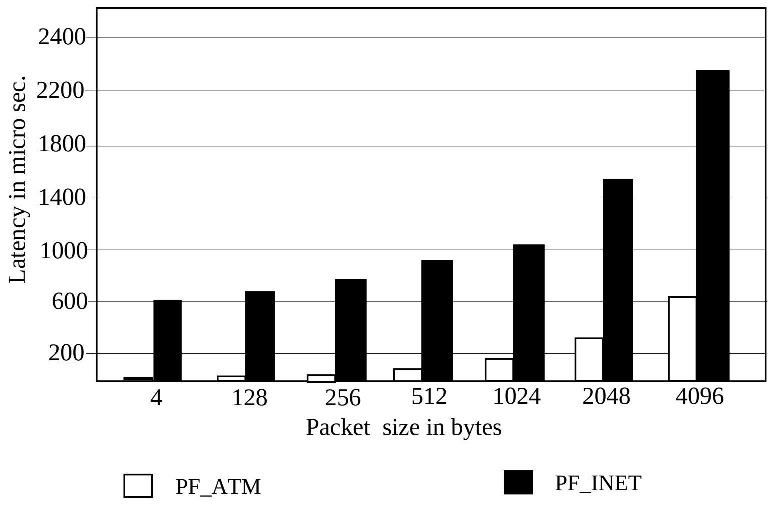

Figure 4: Transmit Side Latency (in microseconds).

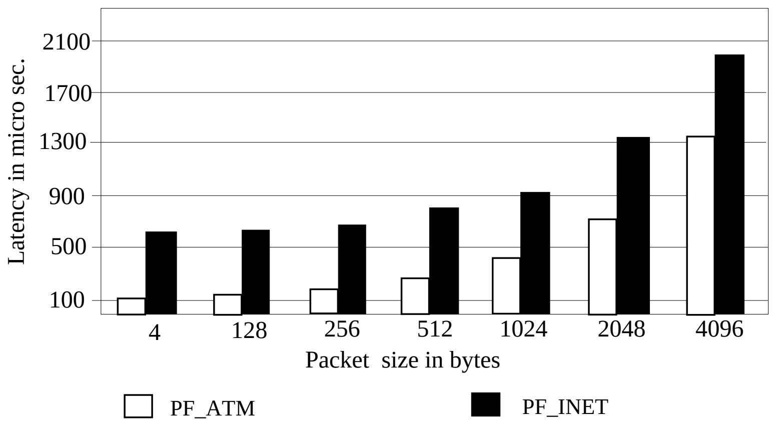

Figure 5: Receive Side Latency (in microseconds).

copied into the system's main memory and then to the buffer on MMT. The reason behind this asymmetry is the lack of buffering on MMT. When MMT generates data, it can be moved directly to the ATM adapter since it has 2Mbytes of on board buffer. On the other side, when data is to be delivered to MMT it needs to be buffered in the main memory first, since the 4Kbytes receive buffer on MMT is often not large enough. This example illustrates the effectiveness of connection specific handlers in optimizing data paths in a fashion most suitable for the device/application.

Besides substantial improvements in throughput, there are other benefits of the PF_ATM protocol stack. It provides a simple interface to setup point to multipiont connections, a feature particularly useful to the conferencing application. It also helps establish a flexible end-to-end quality of service architecture. The socket based interface allows applications to negotiate QoS parameters and alter traffic contracts and service requirements during the lifetime of a connection. When completely implemented, this will be a very attractive feature for the video-conferencing application, operating in an interactive and highly dynamic environment.

4 Concluding Remarks

We have proposed a new protocol architecture tailored to ATM networks which is particularly suitable for multimedia applications. Our design is based on three basic principles – separation of control and data flows, minimal overhead and duplication of function, and application access to ATM level QoS guarantees. The performance gains of the new protocol architecture has been demonstrated on a video-conferencing testbed.

Acknowledgement

The authors would like to thank Tsipora Barzilai and Yakov Rekhter for many useful discussions.

References

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

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

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

[4] B. Manning and R. Colella. DNS NSAP resource records. Internet RFC–1706, October 1994.

[5] A. Romanow and S. Floyd. Dynamics of TCP Traffic over ATM Networks. In *Proceedings, ACM SIGCOMM*, August 1994.

[6] D. Saha, D. Kandlur, T. Barzilai, Z. Shae, and M. Willebeek-LeMair. A videoconferencing testbed on ATM: Design, implementation, and optimizations. In *Proceedings IEEE Conference on Multimedia Computing and Systems*, April 1995.

[7] The USENIX Association and O'Reilly & Associates, Inc., 103 Morris Street, Suite A, Sebastopol, CA 94572. *4.4BSD Programmer's Reference Manual*, April 1994.

[8] A. Wolman, G. Voelker, and C. A. Thekkath. Latency Analysis of TCP on an ATM Ntework. In *Proceedings, Winter Usenix*, January 1994.