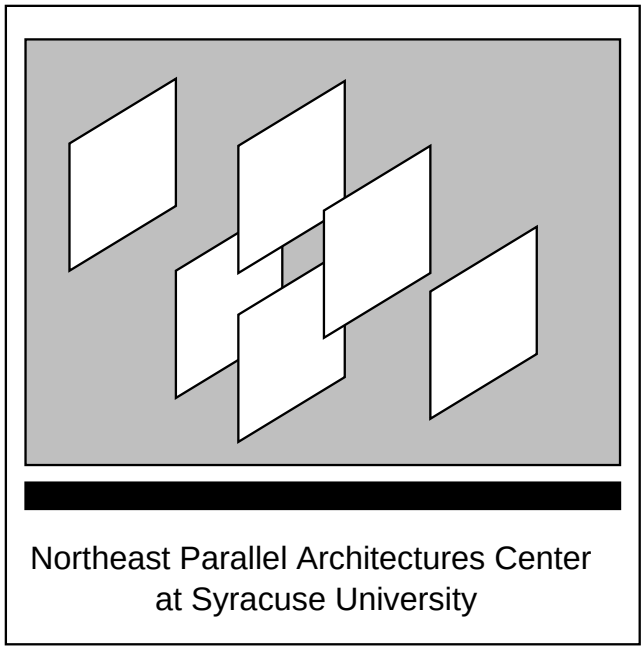

Protocol Overhead in ATM networks
SCCS-678

Mahesh Subramanyan

Northeast Parallel Architectures Center
111 College Place
Syracuse University
Syracuse, NY 13244-4100

Abstract

This report discusses the performance and sources of protocol overhead in ATM networks. It first introduces ATM followed by a description of the inhouse ATM network. Our primary goal was to study the performance tradeoff of chosing TCP/IP versus ATM API in a local ATM network.

1 Introduction

While networks, especially local area networks, have been getting faster, perceived throughput at the application has not always increased accordingly. Various performance bottlenecks have been encountered, each of which has to be analyzed and corrected. The layer most often suspected of contributing to low throughput is the transport layer of the protocol suite. This layer has considerable functionality and is typically executed in software by the host processor at the end points of the network. It is thus a likely source of processing overhead.

TCP is the transport protocol from the Internet protocol suite. In this set of protocols, the functions of detecting and recovering lost or corrupted packets, flow control, and multiplexing are performed at the transport level. TCP uses sequence numbers, cumulative acknowledgment, windows, and software checksums to implement these functions. TCP is on top of a network protocol called Internet Protocol (IP). This protocol, which is connectionless or datagram packet delivery protocol, deals with host addressing and routing, but the latter function is almost totally the task of the internet level packet switch, or gateway.

Our Primary goal was to study the performance tradeoff of choosing different APIs in a local ATM environment [LHD+95]. In our test environment, several workstations were connected via a Fore's ASX-100 ATM switch. The details of this local ATM environment will be discussed in Section 2. We consider the following two APIs:

- Fore's API [For93],
- BSD socket programming interface [SLQ90, Sun90].

Fore's API provides several capabilities which are not normally available in other APIs, such as guaranteed bandwidth reservation, selection of different ATM adaptation layers (AAL), multicasting, and other ATM specific features. The BSD socket interface provides facilities for Interprocess Communication (IPC). It was first introduced in the 4.2BSD Unix operating system. For interprocess communication, any API can be used. However, the performance of the application may be affected by the decision made. Each layer may also represent communicationg in a different protocol layer. In the socket interface, application can choose different transport protocol combinations such as Transmission Control Protocol/Internet Protocol (TCP/IP), User Datagram Protocol/Internet Protocol (UDP/IP), or even raw sockets for interprocess communication. ATM can use either ATM Adaptation Layer 3/4 or 5 for IP. In this study we focus on the communication efficiency. An echo program is used to measure end-to-end communication characteristics and to explore the underlying communication capabilities of different APIs over local ATM and Ethernet networks.

The next section gives an overview of ATM technology. Section 3 presents a description of the two APIs. The configuration of our experimental hardware and software is discussed in section 4.

2 Overview

In this section, we give an overview of ATM technology.

Figure 1: BISND Protocol Reference Model

ATM is a best effort delivery system. Flow control is to be provided by the user. A sequence of cells in an ATM connection will be received in the same order as it was transmitted. ATM guarantees that cells will not be disordered.

The foundation elements of ATM were synthesized by researchers at AT&T Bell Laboratories and France Telecom's Research Center in the early to mid-1980s. Fast packet switches differ from X.25-like packet-switching in that they minimize storing, processing, and storing activity at each link. For example, error control and flow control are performed on an end-to-end, rather than on a link-by-link basis. By reducing the activities at each link, additional throughput is possible.

ATM is a cell-switching and multiplexing technology designed to combine the benefits of circuit switching (constant transmission delay, guaranteed capacity) with those of packet switching (flexibility, efficiency of intermittent traffic).

Technologies that fragment data into small pieces can have disastrously low performance if any of the

Figure 2: STM and ATM Channels

In STM, time-division multiplexing are employed to preassign users to time slots. ATM time slots are made available on demand with labels identifying the source of transmission in each cell. TDM is inefficient relative to ATM because, if a station has nothing to transmit when its time slot comes up, that time slot is wasted. The converse situation, where one station has lots of information to transmit, is also less efficient. In this case, the station can only transmit when its turn comes up, even though all the other time slots may be empty. STM services dedicate a physical path to a voice call for the duration of the call. No other call

can use this facility. Once the call is completed, everything is torn down and made available for use by the next call. ATM - asynchronous transfer millennium is a sophisticated well conceived technology.

The ATM-adaptation Layer (AAL) maps information into cells and performs segmentation and reassembly. The services provided by this layer can be classifed into four types depending on whether a timing relationship must be maintained between source and destination, whether the application requires a constant bit rate, and whether the transfer is connection-oriented or connectionless. Recently the ATM Forum specified a new type of AAL, called AAL 5. The objective is to offer a service with less overhead and better error detection. This layer is also referred to as *Simple and Efficient AAL*.

3 Application Programming Interface

This section describes two APIs: Fore Systems' ATM API, and BSD's socket interface.

3.1 Fore Systems ATM API

Fore Systems' user-level ATM library routines provide a portable interface to the ATM data link layer. It ia an ATM adaptation layer based programming interface, allowing software designers to create applications without the TCP/IP protocol stack and the socket layer buffering mechanisms. The library routines for SunOS and IRIX platforms are STREAMS-based.

The ATM library provides a connection-oriented client and server model. Before data can be transferred a connection has to be established between client and server. Once the connection is established, the network makes a "best effort" to deliver the ATM cells to the destination. The cells may be dropped depending on the network resources remaining. End-to-end flow control and retransmission are left to the application.

Applications first open a file descriptor with *atm_open()* and then bind a local ASAP to the file descriptor with *atm_bind*. The local NSAP is also implicitly bound to the same file descriptor. The remote ASAP and NSAP are associated with the file descriptor when a connect indication or a connect confirmation is received. Connections are established using *atm_connect()* in combination with *atm_accept()*. These operations allow the data transfer to be simplex, duplex, or multicast. ATM *Virtual Path Identifiers* (VPI) and *Virtual Channel Identifiers* are assigned by the network during connection establishment. The ATM device driver associates the VPI/VCI with an ASAP which is in turn associated with a file descriptor. When a connection is duplex, both an incoming and an outgoing VPI/VCI are associated with the ASAP; the two need not be the same.

It provides guaranteed bandwidth reservation for each connection. The network uses the bandwidth information, and will refuse connection if the requested bandwidth is not available. Applications can also select the kind of ATM Adaptation Layer (AAL) to be used for data exchange. In the current implementation of Fore, AAL type 1 and 2 are not supported, and types 3 and 4 are treated identically. The main difference between AAL 3/4 and AAL 5 lies in their multiplexing and error detection capabilities as well as in the amount of overhead generated by each of them and reducing the effective avaialabe bandwidth.

atm_send() and *atm_recv()* are used to transfer user messages. One Protocol Data Unit (PDU) is transferred on each call. The maximum size of the PDU depends on the AAL chosen for the connection and the constraints of the underlying socket-based or stream-based device driver implementation.

3.2 BSD Socket-Based Programming Interface

The 4.2BSD kernel introduced an Interprocess Communication mechanism *(sockets)*. A socket is an end-point of communication referred to which a name may be bound. Two processes each create a socket, and then connect to those two end-points to establish a reliable byte stream or unreliable datagram. Once connected, the descriptors for the socket can be read from or written to by user processes similar to regular file operations. The transparency of sockets allows the kernel to redirect the output of another process residing on another machine.

Sockets exist within communication domains. Domains are abstractions which imply both an addressing structure and a set of protocols which implement various socket types with the domain. BSD socket supports the Unix domain, the Internet domain, and the NS domain. In our environment, we are limited to use the Internet domain for communication over local ATM networks. A connection is a mechanism used to avoid having to transmit the identity of the sending socket with each packet of data. Instead, the identity of each end-point of communication is exchanged prior to transmission of any data, and is maintained at each end so that it can be referred to at any time when sending or receiving messages. In the Internet domain, *stream sockets* and *datagram sockets* use TCP/IP and UDP/IP as the [Sun90] as the underlying protocols. A *stream socket* provides for the connection-based, bidirectional, reliable, sequenced, and unduplicated flow of data without record boundaries. A *datagram socket* on the other hand supports connectionless, bidirectional flow of data that is not promised to be sequenced, reliable, or unduplicated. In the experiments presented here,

4 Environment

The experiments were performed on a pair of Sun IPX connected by a Fore ASX-100 ATM switch. The ATM environment was provided by the NYNET (NewYork NETwork). This is a part of the NPAC network as shown in Figure 3. Fore Systems, Inc. host adapters and local area switches were used. The host adapters were SBA-200 adapters for the SUN SBus. The physical media is the 140 Mbits/sec TAXI interface.

4.1 ATM SBus Adapter

The SBA-200 host adapter is Fore's second generation interface and uses an Intel i960 as an onboard processor. The i960 takes over most of the AAL and cell related tasks including the SAR functions for AAL 3/4 and AAL 5, and cell multiplexing. With the Series-200 adapter, the host interfaces at the packet level feeds lists of outgoing packets and incoming buffers to th i960. The i960 uses local memory to manage pointers to packets, and uses DMA to move cells out of and into host memory. Cells are never stored in adapter memory. The SBA-200 currently supports a Fore Systems proprietary signalling protocol (SPANS). The next signalling release will be based on ATM Forum signalling.

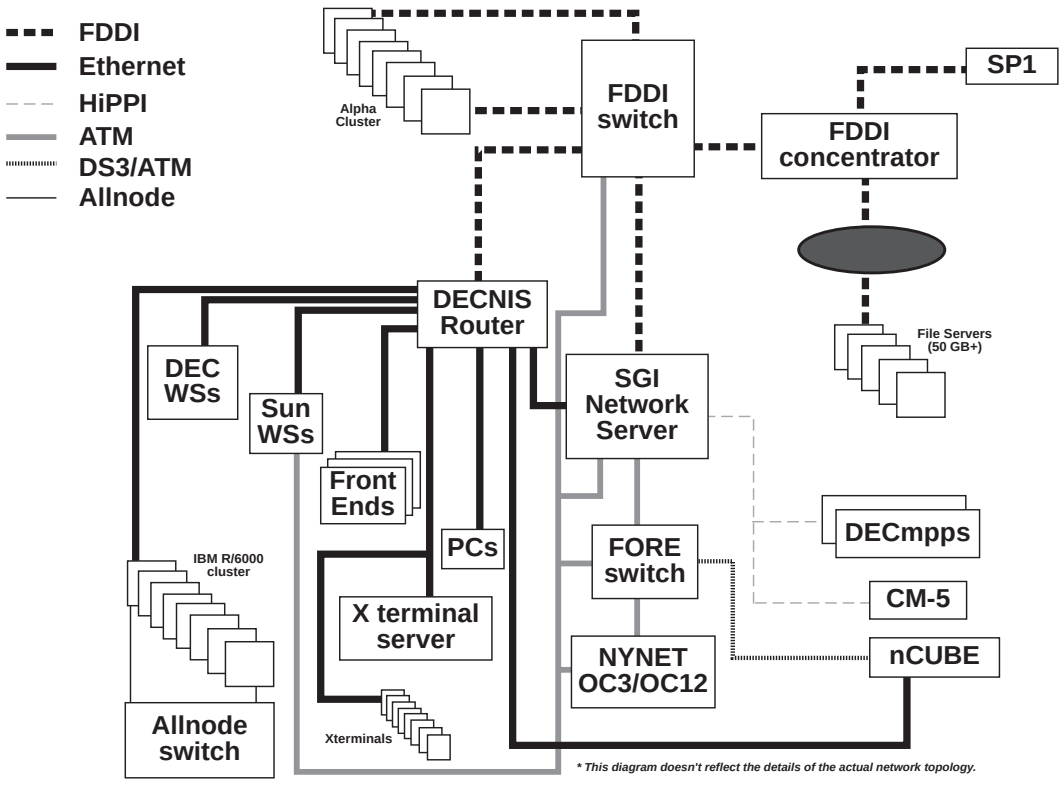

Figure 3: NPAC Networking Infrastructure

4.2 ATM Switch

The ASX-100 local ATM switch is based on a 2.4 Gbps switch fabric and a RISC control processor. The switch supports fore network modules with each module supporting upto 622 Mbps. Modules installed in the NPAC ATM switch include two four-port 140 Mbps TAXI modules, and one two-port 155 Mbps SONET OC-3c module. This ATM switch is connected to other ASX-100 ATM switches in Rome Labs and Cornell. The ASX-100 supports Fore's SPANS signalling protocol with SBA-200 series adapter, and can establish Switched Virtual Circuits (SVCs) or Permanent Virtual Circuits (PVCs). All of the experiments ignored circuit setup time and thus the ATM circuits can be viewed as PVCs.

4.3 Multi-mode Fiber

The multi-mode fiber [Cav94] interface is based on the physical medium used for FDDI. It uses 62.5 micron optical fiber with a 125 or 175 Mbps line rate and a 4B/5B encoding, yielding rates of 100 or 140 Mbps. The multi-mode fiber interface is sometimes referred to as the TAXI interface, after a chipset used in FDDI (and multi-mode fiber) interface implementations. The 4B/5B encoding of the multi-mode interface provides 32 5-bit codes. Sixteen of these are used to indicate 4-bit nibbles of data. Some of the remaining sixteen are used as command symbols. Three pairs of these command symbols have special significance in the multi-mode interface. They are JK, the sync (or idle), code, TT, the *the start of cell* code, and QQ, the *the loss of signal* code.

The overhead of the multimode fiber interface consists of a single *single start of cell* code inserted at the

Table 1: Bandwidth Available after Protocol Overhead (Multi-mode Fiber).

	140Mbps		
	MTU=576	MTU=9,180	MTU=65,527
Line rate		140.000	
To ATM		134.909	
To AAL	122.182	122.182	122.182
To IP	114.350	121.811	122.120
To Transport	108.867	121.439	122.068
To appl. via TCP	104.951	121.174	122.031

start of each cell and a single *idle* code between each pair of cells. At 100 Mbps, this overhead amounts to 3.636 Mbps, leaving 96.364 Mbps for the next higher layer. The format of an ATM cell stream on multi-mode fiber is shown in Figure 4

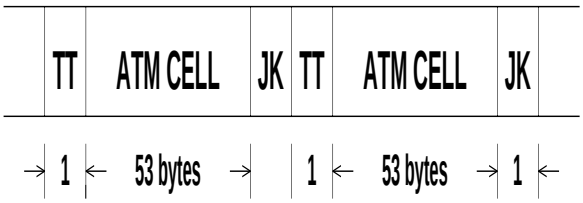

Figure 4: Multi-mode Fiber Transmission Format

Table 1 shows the amount of bandwidth that remains after each successive layer has claimed its share of the overhead [Cav94]. We assume that IP contributes 20 bytes of overhead per datagram and TCP contributes 20 bytes of overhead per segment. MTU is the size of the IP datagram. We consider three MTUs: 576 (the Internet inter-network default), 9,180 (the proposed default for IP over ATM), and 65,527 (the maximum for IP over AAL5).

5 Experimental Results

The experimental results are shown in Table 2. This section provides measurements from a real environment showing performance gain using the Fore SBA-200 SBus ATM TAXI adapter. The workstations are SUN Sparcstation IPX, runing SUN OS 4.1.3. The target of these measurements was to get an idea of the throughput that can achieved using Fore ATM API. The SBA-200 is a follow up of the SBA-100 and features an on-board processor (i960) as well as DMA transfer capabilities via the SBus. The throughput of the SBA-200 is more ethan five times as high as the one on SBA-100 allows depending on the size of the transmission unit. This is due to the architecture of the SBA-200, which performs AAL and ATM layer computing onboard. The SBA-100 is particularly slow when using AAL 5, because the CRC-32 computing at CS layer is done in software.

From the above table we observe that using ATM API we can obtain a throughput of around 34.2 Mbps (for TU=4092 bytes) where as using TCP/IP we get a throughput of about 15 Mbps. However one point to

Table 2: Fore API Throughput results using 140Mbps TAXI adapter (Timings in microsec and TU in bytes).

140Mbps	TU=1	TU=1024	TU=2048	TU=4092
Using API	0.35	278	523	958
Using TCP/IP	0.614	587	1120	2184

be noted is that TCP/IP has error recovery and flow control built into it whereas in ATM API these need to be implemented at the application level.

6 Conclusions and Future Work

We studied the performance of two different APIs. The experimental results demonstrate that local ATM networks are very promising. The two APIs presented in this report present a distributed programming environment over a different communication layer in the protocol hierarchy. Fore API provides better communication performance, but lacks distributed programming support. The users will then have to put extra effort to develop distributed applications. Fore's API has a maximum transfer unit of 4 Kbytes. Thus a user level message segmentation/reassemble is required The data transmission is also a best effort delivery system. Hence message retransmission and flow control is left to the application. The communication interface of Fore's API is similar to that of a socket interface. The current implementation of Fore API does not support a concurrent server model. We are currently developing a multimedia communication protocol for NYNET which will provide low latency and high throughput.

7 Acknowledgement

I like to thank Dr. Marek Podgorny and Prof. Salim Hariri, and Dr. Roman Markowski for their invaluable guidance and encouragement during the course of this study.

References

[Cav94] John David Cavanaugh. Protocol Overhead in IP/ATM Networks. Technical Report 1994-08-12, Minnesota Supercomputer Center, Inc., 1994.

[For93] Fore Systems, Inc., 174 Thorn Hill Road, Warendale, PA 15086-7535. *ForeRunner SBA-200 ATM SBus Adapter User's Manual*, 1993.

[LHD+95] Mengjou Lin, Jenwei Hsieh, David Du, Joseph Thomas, and James McDonald. Distributed Network Computing over Local ATM Networks. *IEEE Journal on Selected Areas in COmmunications Special Issue of ATM LANs: Implementations and Experiences with an Emerging Technology*, Early '95. Accepted to appear.

[SLQ90] M. Karels S. Leffler, M. Mckusick and J. Quaterman. *The Design and Implementation of the 4.3 BSD Unix Operating System*. Addison-Wesley, 1990.

[Sun90] Sun Microsystems, Inc. *Network Programming Guide*, March 1990.