Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 97-050

High-Speed Network Support for High-Performance Network Computing and Multimedia Communications

by: Jenwei Hsieh
(Ph.D. Thesis)

HIGH-SPEED NETWORK SUPPORT FOR HIGH-PERFORMANCE NETWORK COMPUTING AND MULTIMEDIA COMMUNICATIONS

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

JENWEI HSIEH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

October 1997

UNIVERSITY OF MINNESOTA

This is to certify that I have examined this bound copy of a doctoral thesis by

JENWEI HSIEH

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

David H.C. Du

_____

Name of Faculty Adviser(s)

_____

Signature of Faculty Adviser(s)

_____

Date

GRADUATE SCHOOL

Abstract

The prevalence of computer networks has shifted the computing paradigm from mainframe or *host-centric* computing to *network-centric* computing. In *network-centric* computing, applications are executed distributedly on a collection of computers interconnected via local and wide area networks. The performance of network-centric applications can be dramatically improved with switch-based high-speed networks, such as HIPPI, ATM, and Fibre Channel. In this study, we focus on the high-speed network support for two important applications in *network-centric* computing: high-performance network computing and multimedia communication.

One important class of network computing is cluster computing, which enables a collection of locally interconnected computers to be used as a general-purpose parallel computing system. Large problems can be solved cost effectively by using the aggregate processing power and memory space of a cluster. However, communication between processors has long been the bottleneck of cluster computing. We have especially concentrated on maximizing the achievable throughput and minimizing the communication delay for cluster computing in homogeneous environments. We have enhanced a popular cluster computing environment, Parallel Virtual Machine (PVM) with clusters of workstations on either local ATM or HIPPI networks.

One possible extension of cluster computing is to incorporate clusters of computers via wide area networks. This is called *meta-computing*. For example, a group of diverse high-performance computers from several geographically distributed supercomputer centers can be employed to solve large problems. ATM is the de facto standard for wide area networks. However, most of the supercomputer centers use

HIPPI in their computing facilities. The internetworking of HIPPI networks and wide area ATM networks becomes an important issue for meta-computing. Two feasible solutions for the problem, *HIPPI Tunneling* and *IP Routing*, have been studied in this thesis.

Multimedia communication imposes another challenge for high speed networks. The delivery of continuous media requires high communication bandwidth and real-time constraint. We have studied two new CBR transmission schemes, called PCR-assist CBR (PCBR) and PCR-assist Dual-Rate CBR (PDCBR), which employ the Program Clock References (PCR) embedded in the MPEG-2 Transport Streams to regulate their transmission. The two schemes provide flexible trade-off between buffer requirement and transmission rates.

Acknowledgments

I wish to express my sincere gratitude to my advisor, Professor David H.C. Du for his guidance, encouragement and support throughout the past years. It is my privilege to have his supervision. I would like to thank the members of my thesis committee, Professors Pen-Chung Yew and Jaideep Srivastava from the Computer Science Department, Professor David J. Lilja from the Electrical Engineering Department, and Professor Walter Littman from Mathematics Department for their valuable feedback on my work. In particular, I would like to thank Professors David J. Lilja and Jaideep Srivastava for reviewing this thesis. I would also like to thank Professor Pierre C. Robert from Soils Science Department for his funding during the early years of my Ph.D. study.

My research was inspired by many colleagues of our research group. In particular, I would like to thank Mengjou Lin for his support, suggestions and hardwork on various topics we worked on together. I would like to thank Rose Tsang and James Schnepf for their valuable comments on my research work. I would like to thank Jonathan Liu for his hardwork and support on video-on-demand server issues. I would like to thank Horng-Juing Lee, Taisheng Chang for their support and feedback on PCR-assist video transmission. I would like to thank Simon Shim and Yue-wei Wang for their hardwork and support on serial storage issues. I also wish to express my appreciation to Yen-Jen Lee, Wei-hsui Ma, Ying-Li Wu, Sheue-Ling Chang, Paisal Keattithananant, and Sudesh Kamath for their support on all the exciting projects we have worked on.

The system staffs of Computer Science Department have solved many tedious

problems for my experimental work. I would like to thank James MacDonald, Andrew Nelson, Paul Dokas, Dan Mack, Dr. Norman Troullier, Mike McShane, and Irrene Jobcoson for their support. I would like to thank Tom Ruwart from the Laboratory of Computational Science and Engineering for his support on various projects. Many friends from industry community also provided support to my experimental work. I would like to thank Joseph Thomas and Timothy Salo from the Minnesota Supercomputer Center Inc., Jack Pugaczewski and Jeffrey Kays from U S WEST Communications, Dave Archer, Gary Delp, Kevin Plotz and Walt Krapohl from IBM Rochester for their suggestions and support. I would also like to thank Reza Roohollamini at Dell Computer Corporation for his support during the final stage of my thesis work.

Finally, I wish to thank my parents, James and Chen-shi Hsieh, and my parents-in-law for their love and support. I would like to dedicate this thesis to my wife, Wei-wen Pan, and our daughter, Dana, for their continual support and love.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The prevalence of computer networks has shifted the computing paradigm from mainframe or *host-centric* computing to *network-centric* computing. In host-centric computing, powerful computers and servers provide service to a large number of users in a time-sharing style. In *network-centric* computing, applications are executed distributedly on a collection of computers interconnected via local and wide area networks. Applications can be arranged to run on different computers, depending on data access patterns, types of processors available, amount of data to be transmitted, and the communication latency between computers sharing data. This allows the computer system best suited for a particular function to be fully utilized.

The performance of network-centric applications can be dramatically improved with switch-based high-speed networks, such as HIgh Performance Parallel Interface (HIPPI) [60, 61, 62, 63], Asynchronous Transfer Mode (ATM)[6, 36], and Fibre Channel [40, 64]. Compared to the legacy networks, these high-speed networks have the following superior features:

- *High data transfer rates.* Switch-based high-speed networks provide higher data transfer rates than traditional local area networks, such as Ethernet, Token Ring, and Fiber Distributed Data Interface (FDDI). The typical date transfer rates of ATM, HIPPI, and Fibre Channel are 155 or 622, 800, and 800 Megabits per second (Mbps), respectively. In the standards of these network technologies, transfer rates higher than Gigabits per second have also been defined.

- *Low latency.* With hardware assistance in the physical and network layer, such

as segmentation and re-assembly and fast packet switching in ATM platform, these high-speed networks have lower communication latency. We have observed less than 300 μsec end-to-end latency at the application level between two processors [26]. As the technology involved in these networks matures, their hardware and software components will be refined to provide greater reductions in latency.

- *Scalability.* In switch-based high-speed networks, each host usually has a dedicated connection to the switch. The communication between each pair of hosts is established through the switch. A switch is capable of supporting multiple connections simultaneously. The aggregate throughput may easily be scaled up by connecting more hosts via additional links and switches.

- *Flexible paradigm.* Diverse applications can be benefited by the flexible communication paradigm provided by switch-based high-speed networks. For example, multicasting communication can be used for multimedia applications such as video-conferencing and computer supported cooperative work (CSCW). The switch can prevent multicasting traffic from disturbing other hosts whose communication paths do not overlap with the multicasting communication.

- *Support for multiple classes of service.* By segmenting user data into fixed-size packets (as in ATM) or using multiple routing fabrics in a switch (as in Fibre Channel), traffic from different types of service may be multiplexed together. For example, ATM provides connection-oriented and connectionless circuits, realtime and non-realtime connections, and constant bit-rate and variable bit-rate transmission. Fibre Channel supports three classes of services (circuit-oriented connection, datagram transmission with acknowledgment, and datagram without acknowledgment) and intermix of these service.

With this rich set of features, it is important to understand how network-centric applications can benefit by using these networks as their underlying communication facilities. In this study, we focus on the high-speed network support for two important applications in *network-centric* computing: high-performance network computing and multimedia communications.

1.1 High-Performance Network Computing

Network computing refers to distributed parallel computing based on networks of computers. With the connectivity provided by various networks, it is possible to employ a collection of computers together for large computation tasks. One important feature of network computing is the potential of partitioning a computing task based on the service provided by different types of processors. Since networked computing environment consists of a variety of computing capabilities, its ability to execute subtasks of a computation on the processor most suited to a particular function enhances performance and improves resource utilization [23].

One important class of network computing is cluster computing. Cluster computing enables a collection of locally interconnected multiprocessor computers (e.g. shared memory multiprocessors), workstations, or personal computers with off-the-shelf components to be used as a general-purpose parallel computing system. Large computational problems can be solved more cost effectively by using the aggregate processing power and memory space of a cluster than using expensive special purpose computers. In cluster computing, networks of computers are controlled by a cluster management software (CMS) to provide a cluster computing environment (CCE). Distributed applications are implemented by partitioning the computation into subtasks and assigning them to individual processes. Processes communicate with each other based on a message passing model.

Usually, the message passing model is supported by special communication libraries in a CCE. It allows flexible communications among collaborative processes, such as point-to-point communication, multicasting and broadcasting communication, group communication, and collective communication. Most of CCEs are designed to be used on a collection of general purpose computer systems such as Unix workstations. Their communication libraries are implemented with standard transport protocols, such as Transmission Control Protocol (TCP) or User Datagram Protocol (UDP), for portability. For example, the communication library of a popular CCE, Parallel Virtual Machine (PVM), is built on top of TCP, UDP, and Unix domain inter-process communication (IPC). Although a CCE provides flexible and portable communication facilities, the message exchange between processors usually has long latency. Latency is incurred by hardware and software components. Hardware latency is incurred by memory and bus architecture of the host, network interface board, switches, and signal propagation delay. Software latency is incurred by interactions among the host's operating system, device drivers, and high level protocols. Long communication latency not only affects the performance of cluster computing, but may also becomes the bottleneck.

Fast message passing is achievable with the following three approaches. First, employment of switch-based high-speed networks increases the message passing speed. Second, efficient implementation of device drivers of network interfaces can improve the performance of message passing. Third, the performance can be further enhanced via bypassing the high-level protocol stack and using lower layer protocols to reduce the overhead of protocol process. In this study, we have especially concentrated our effort on minimizing the communication delay and maximizing the achievable throughput with the first and the third approaches. We have also focused on cluster computing in homogeneous environments in order to use the third approach. In these environments, all participating computers use same network interfaces and the

associated low-level application programming interface (API).

We have enhanced the communication subsystem of PVM on clusters of workstations with either local ATM or HIPPI networks. From the experimental results, the achievable throughput is improved while reducing the end-to-end communication delay. The study shows that not only the performance of a CCE is improved, the approaches also allow a CCE to utilize unique features of the switch-based high speed networks. For example, the inherent multicasting feature of ATM switch facilitates the point-to-multipoint communication in a CCE. The credit-based flow control mechanism of HIPPI network reduces the chance of packet loss due to buffer overflow at the receiver.

1.2 Meta-Computing

The advantage of cluster computing have already attracted many institutes to use it as an alternative form of high-performance computing. One possible extension of cluster computing is to incorporate clusters of computers via metropolitan or wide area networks. This is called *meta-computing*. In meta-computing, an assembly of diverse high-performance computers or clusters of computers interconnected via local and wide area networks are perform as one computing system. A typical example is to employ a group of distinct computers with special functionalities from several supercomputer centers, which are geographically distributed in different locations, to solve large scientific problems. From the user's point of view, a meta-computer is a powerful virtual computing system that can be tailored to fulfill their processing requirements.

It is obvious that networking is critical to the performance of meta-computing. Some of meta-computing environments use different network technologies in their local and wide area networks. For example, most of the supercomputer centers use

HIPPI as high-speed local area networks in their computing facilities. This is because HIPPI was originally designed by supercomputing community to provide transmission of large volume of data between supercomputers or high-end servers. For wide area networks, ATM is the de facto standard. Therefore, the internetworking of local area HIPPI networks and wide area ATM networks becomes an important issue for the meta-computing.

Two feasible solutions for the problem, *HIPPI Tunneling* and *IP Routing*, have been studied in this thesis. HIPPI Tunneling provides interconnection between HIPPI and ATM at the physical layer while IP Routing forwards data packets between them at the network layer. In the former solution, HIPPI packets are encapsulated in the ATM Adaptation Layer (AAL) 5 packets and forwarded between HIPPI networks. In the last solution, HIPPI packets are delivered in a store-and-forward fashion with standard network routing approach. We have compared these two schemes in terms of their network connectivities, protocol overheads, and flow controls. Experimental results of both schemes are presented.

1.3 Multimedia Communications

Multimedia communications refer to the transmission of continuous media, such as video and audio, via any type of communication networks while satisfying their real-time constraint. It imposes challenges for high speed networks. First, the delivery of compressed continuous media requires sustained high communication bandwidth. For example, video compressed in Motion Pictures Expert Group (MPEG)-2 format usually require 4 to 20 Mbps bandwidth. Second, in order to provide continuous playback of these media, video frames or audio clips must be received before the time they need to be displayed.

Video transmission has become an essential component of multimedia applications, such as video-conferencing, video on demand and distance learning. Before video or other continuous media are transmitted through the network, they must be compressed. Otherwise, the network will not be able to accommodate their bandwidth requirement. Among the compression schemes and network technologies, MPEG-2 over ATM has been embraced by the industry to deliver high quality video and audio over high speed networks [20]. MPEG-2 uses both intra-frame and inter-frame encoding schemes[21]. It achieves high compression ratio by the use of bi-directional motion compensation, which generates a more accurate prediction that requires fewer bits to represent. Thus, for constant video quality, MPEG-2 compressed video demonstrate the property of variable bit rate.

Intuitively, variable bit-rate (VBR) service is suitable to deliver MPEG-2 compressed video. However, VBR service supports statistical multiplexing feature by sharing the bandwidth dynamically among all traffic within the same service class. It only guarantees statistical quality of service based on a set of traffic descriptors, such as peak rate, burst length, and sustained rate. On the other hand, transmitting compressed VBR video with traditional constant bit-rate (CBR) service often requires a large buffer at the viewer's side to absorb the difference between the rate that video is received and the rate that video is displayed.

In this thesis, we have studied two new CBR transmission schemes which utilize the timing information (called Program Clock References, PCR) embedded in the MPEG-2 streams. PCRs are readings of the system clock of a MPEG-2 encoder. They are embedded in a stream periodically or not longer than 100 ms apart. PCRs are used for media synchronization and clock synchronization. Media synchronization allows multiple media (video, audio, and associated data) to be presented together at the right time. Clock synchronization forces the decoder to synchronize its clock with the source, thus reduces the buffer requirement. The two new schemes, called

PCR-assist CBR (PCBR) and PCR-assist Dual-Rate CBR (PDCBR), which employ the PCRs embedded in the MPEG-2 streams to regulate their transmission.

The PCBR scheme uses PCRs to examine its transmission regularly. It holds up the transmission if it is ahead of schedule, based on the knowledge provided by PCRs. The PCBR scheme requires slightly higher transmission rates than the traditional CBR service. To reduce the transmission rate, we introduce the PDCBR scheme which dynamically changes its transmission between two rates. It uses a low rate if the transmission is ahead of schedule and uses a high rate if the transmission is behind of schedule. The experimental results of these two schemes with real video traces are presented. Based on the results, we have found that the two schemes provide flexible trade-off between buffer requirement and transmission rates.

1.4 Thesis Organization

The organization of this thesis is as follows. In chapter two, we study the high speed network support for cluster computing. The popular PVM is used as the CCE on ATM and HIPPI local area networks. We first describe the popular PVM with emphasis on its existing communication subsystem. Then we outline the approaches that we used to enhance the communication performance of PVM. PVM's communication subsystem was re-implemented directly using two low-level APIs, Hewlett Packard's Link Level Access (LLA) and FORE Systems' ATM API [18]. The performance results of both existing and enhanced versions of PVM are presented.

In chapter three, two solutions (HIPPI Tunneling and IP Routing) to form clusters of computers into meta-computers are presented. We first present how HIPPI networks are extended by encapsulating HIPPI packets on ATM networks. Then, we describe how IP packets are forwarded between HIPPI and ATM networks. The

experimental results of both approaches on the same environment are presented. In both schemes, we used TCP as the transport protocol and measure the end-to-end performance. The experimental results also demonstrate the impact of flow control of high level protocol on the communication performance.

In chapter four, we first provide background information of MPEG-2 system and one of its bit streams, the MPEG-2 Transport Stream, which is designed for network transmission. Then, we introduce the system model of the two new transmission schemes, PCBR and PDCBR. Their analytical models are used to determine the required buffer space and transmission rates. Finally, we present their performance with real video traces.

In chapter five, we conclude the thesis with a summary of contributions.

Chapter 2

High-Speed Network Support for Cluster Computing

Network computing offers a great potential for increasing the amount of computing power and communication facility for large-scale distributed applications. The aggregate computing power of a cluster of workstations or personal computers (PC) interconnected by switch-based high-speed local area networks (LAN) can be employed to solve a variety of scientific and engineering problems. Because of mass production, commercial workstations and PCs have much better performance to price ratio than Massively Parallel Processing (MPP) machines, which uses proprietary components and interconnection networks. With switch-based networks such as HIPPI, ATM, or Fibre Channel, a cluster of workstations also provides cost-effective, high-bandwidth communications. The advantages of network computing have already attracted many companies to use it as an alternative form of high-performance computing. A recent report shows that several companies in aeronautics industry utilize clusters of workstations for computational fluid dynamics processing and propulsion applications during off hours and weekends[1] [59].

[1]Three case studies of recent aeronautics industry experience show how workstation clusters are being used as alternative forms of high-performance computing: (1) McDonnell Douglas has as many as 400 workstations (with an average of 200 per session) divided into clusters of 20 workstations per parallel job doing CFD (computational fluid dynamics) processing during off hours and weekends. (2) Pratt & Whitney (P&W) has achieved the throughput equivalent of 16 CRAY C90 CPUs by using networked workstations at two sites. P&W harnessed 600 workstations in East Hartford plus 300 in West Palm Beach during off hours. (3) Boeing has been conducting a variety of cluster pilot tests including interdepartmental testing on two workstation clusters. One cluster, completed in 1994, involved 14-20 workstations dedicated to propulsion applications.

The following items are the motivational factors for considering the implementation of a parallel computing platform over switch-based high speed local area networks.

- *High data transfer rates.* Traditional local area networks, such as Ethernet and Fiber Distributed Data Interface (FDDI), are shared medium architectures. In shared medium architectures, network capacity is shared among all the interconnected processors. Aggregate network capacity is limited to speeds between 10 Mbits/sec (Ethernet) to 100 Mbits/sec (FDDI). High speed switch-based network architectures, such as the HIgh Performance Parallel Interface (HIPPI), Fibre Channel, and Asynchronous Transfer Mode (ATM) feature aggregate throughput of several gigabits/sec. Moreover each host usually has a dedicated high-speed (155 Mbits/sec or more) connection to the switch.

- *Scalability.* In shared medium architectures, since network capacity is shared among all the interconnected processors, as the number of processing nodes is increased, network saturation quickly occurs. High-speed switch-based networks may easily be scaled up, in terms of processing power or storage, by simply connecting the new devices via additional links and switches.

- *Potentially low latency.* Inherent features, such as dedicated connections, of switch-based high-speed networks lend themselves to potentially supporting low latency data transfers. However currently, as shown in Section 2.3.5.3, traditional networks, like Ethernet, provide slightly lower latency than ATM. This is because Ethernet is a mature technology, and hence software and hardware components have been fine-tuned to provide low latency. As the technology involved in high-speed switch-based networks matures, their software and hardware components will, likewise, be fine-tuned to provide greater reductions in latency.

- *Paradigm flexibility.* The attributes of switch-based high speed networks are likely to foster performance improvements in many existing network applications, as well as increase the feasibility of potential network applications. These applications may belong to disparate paradigms. For instance, high data transfer rates are very attractive to applications requiring large data shipments such as visualization applications. Also switch-based networks inherently support efficient multicasting, and thus may be attractive for supporting distributed shared memory, where multicasting operations are frequently used to update, lock or unlock multiple data copies.

The typical methodology for cluster computing is based on a software framework that executes on participating workstations. The cluster is controlled by a cluster management software (CMS) which is designed to administer and manage application jobs submitted to a cluster. It may also supports functions like configuration, job scheduling and monitoring, load balancing, process management, resource utilization, and fault tolerance. The software framework provides a parallel programming environment that allows programmers to implement distributed algorithms based on a message passing model. Distributed applications utilize the computational power and communication facility of the cluster by using special libraries provided by the software framework. Those libraries usually support process management, synchronization, and message passing based on standard network protocols. Examples of such software framework are PVM (Parallel Virtual Machine) [4, 23, 42], P4 [7], Express [37], Linda [8], and MPI (Message Passing Interface) [19].

Given an environment consisting of a cluster of workstations interconnected by a LAN, it is well-known among programmers that message passing facilities lack the performance found in distributed memory computers such as the Connection Machine CM-5 or the nCUBE which provides specialized high speed switch(es) or interconnect

hardware. This is especially true about current slow speed (e.g., Ethernet) LAN. However, fast message passing is possible via three approaches. First, the change of the underlying network to a high speed network greatly increases the message passing speed. Second, improving the efficiency of device drivers of network interfaces. Third, bypassing the high-level protocol stack and using a lower layer protocol reduces overhead, and thus increases message passing speed. In this study, we have adopted the first and the third approaches to improve communication performance of cluster computing in homogeneous environments.

To reduce the communication latency and exploit high speed networks, we enhanced a popular message-passing library, PVM, on clusters of workstations. The PVM message passing library was originally implemented using the BSD socket programming interface. The transport protocols used are TCP, UDP, and Unix domain inter-process communication mechanism. Figure 2.1 shows how PVM was implemented on the BSD socket programming interface (on the right side of Figure 2.1). The main idea of improving PVM's message passing is to reduce the overhead incurred by the high-level protocols. The overhead incurred by the high-level protocols and the delay caused by the interactions with the host operating system can be varied by using different Application Programming Interfaces (APIs) which are available on different protocol layers [41]. In order to provide as close to optimal performance as possible, part of PVM's communication subsystem is re-implemented directly using two APIs, Hewlett Packard's Link Level Access (LLA) and FORE's ATM API [18], (on the left side of Figure 2.1) instead of the BSD socket interface. Since both APIs reside at a lower layer in the protocol stack, the overhead incurred when directly using these APIs is expected to be lower than the overhead incurred by using the BSD sockets programming API.

In Section 2.2, we provide a brief description of PVM with emphasis on its existing communication subsystem. The re-implementation of PVM using FORE's

ATM API on a cluster of workstations interconnected via a local ATM network is presented in Section 2.3. Another re-implementation of PVM using HP's LLA on a cluster of workstations interconnected via a local HIPPI network is presented in Section 2.4.

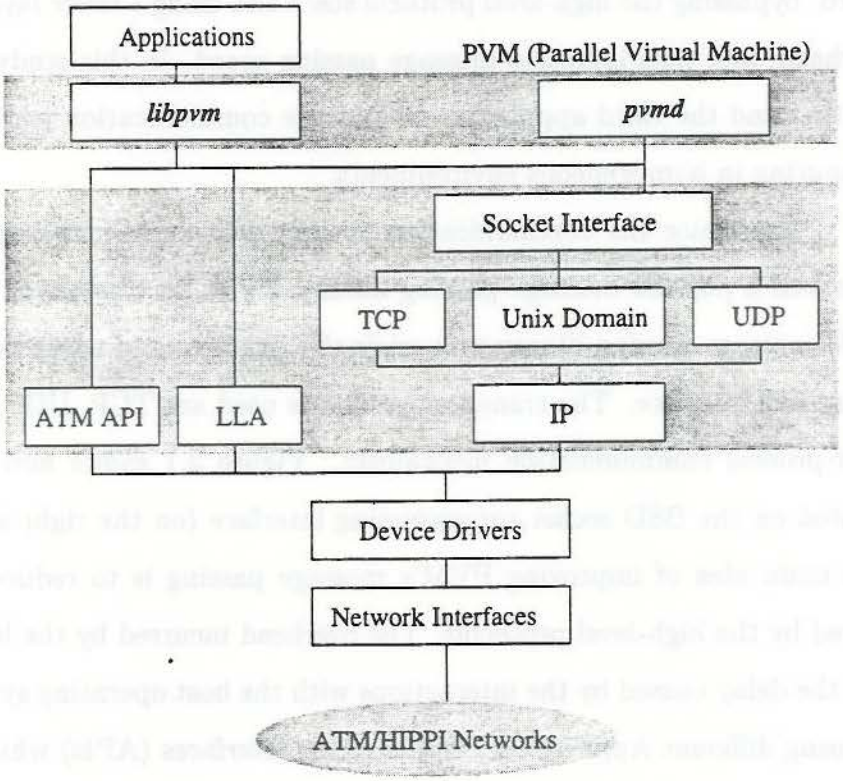

Figure 2.1: Protocol hierarchy of PVM's communication subsystem.

2.1 Related Work

Several communication models have been proposed to enhance the performance of message passing for cluster computing [14, 57, 5, 54, 56, 13]. The Network of Workstations (NOW) project at the University of California at Berkeley demonstrates a new approach to large-scale system design enabled by scalable interconnect

networks that provides low latency and high bandwidth [14]. The interconnect was formed as a variant of a Fat-tree to connect more than one hundred workstations. Active Messages are the basic communication primitives in NOW [57]. The basic idea of Active Messages is that the control information at the head of a message is the address of a user-level instruction sequence that will extract the message from the network on the message arrival and integrate it into the on-going computation at the receiver side. The low overhead of Active Messages for small messages is due to elimination of buffering and rapid handling upon message arrival. Active Messages is a low-level communication primitive for homogeneous clusters. General-purpose message passing libraries or parallel programming models, such as PVM or MPI, can be implemented upon its support.

The SHRIMP multicomputer project at Princeton University studies the use of commodity PCs or workstations and commercially available routing switches to construct scalable multicomputers [5]. SHRIMP uses custom designed network interfaces which allow processes to establish channels connecting virtual memory pages on two nodes such that data written into a page on one side gets propagated automatically to the other side. To create a virtual memory mapping from one node to another, appropriate physical mapping information in the page tables of both network interfaces need to be set up. Based on the virtual memory-mapped communication model, implementations on two network interfaces show that the model eliminates operating system involvement in communication, supports user-level buffer management, and minimizes software communication overhead [12]. A similar memory-based network access model was proposed by Thekkath et al [54]. Their communication model consists of a set of primitives to access remote memory. These primitives allow processes on one node access to a set of remote memory segments, which are contiguous pieces of another process' virtual memory.

A new abstraction for low-latency communication was proposed in the U-Net

communication architecture [56]. The U-Net model virtualizes the network interface so that each process has the illusion of owning the interface to the network. The basic idea in U-Net is to incorporate message multiplexing and demultiplexing into the network interface and to move buffer management and protocol processing to user-level. The approach basically removes the kernel from the critical path of sending and receiving messages. The processing overhead on messages can be reduced. With the influence of the U-Net architecture and other research experiences, the Virtual Interface architecture (VI architecture) is being jointly specified by a number of companies [13] for cluster computing. The VI architecture defines mechanisms for low-latency, high-bandwidth message-passing for clusters of high-volume servers or workstations. In VI architecture, a process is allowed to send and receive messages to and from the network interface without the involvement of the operating system.

2.2 Parallel Virtual Machine

Parallel Virtual Machine (PVM) [4, 23, 42] is a software system for the development and execution of parallel applications. It allows an interconnected collection of independent heterogeneous computers to appear as a single *virtual* computational resource or a single parallel machine. The independent machines may be ordinary workstations, multiprocessors, supercomputers, or specialized processors. The interconnection network may be a general network such as an Ethernet, the Internet, or any high-speed network.

Computing resources are accessed by applications via a suite of PVM defined user-interface primitives. The PVM suite provides a standard interface that supports common parallel processing paradigms, such as message passing and shared memory. An application would embed well-defined PVM primitives in their procedural host language, usually C or Fortran. The PVM suite provides primitives for such

operations as point-to-point data transfer, message broadcasting, mutual exclusion, process control, and barrier synchronization. In most cases, the user views PVM as a loosely coupled, distributed memory computer with message passing capabilities programmable in C or Fortran.

In a PVM *virtual machine* environment there exists a support process, called *pvmd*, or daemon process, which executes on each host. Pvmds execute independently from one another. During normal operations they are considered equal peer processes. However, during startup, reconfigurations, or operations such as multicasting, there exists a master-slave relationship between pvmds. Each pvmd serves as a message router and a controller. Pvmds are used to exchange network configuration information, and dynamically allocate memory to store packets traveling between distributed tasks. Pvmds are also responsible for all application component processes (tasks) executing on their hosts.

Figure 2.2 depicts a network of three hosts. Each host has a local pvmd and a number of local tasks. Communications between hosts may occur as a task-task, task-pvmd-pvmd-task, or pvmd-pvmd interaction. Communication within a host, task-pvmd, occurs via Unix domain sockets. PVM message routing will be discussed in detail in Section 2.2.1. In Figure 2.2, Machine C has two tasks, task 6 and a console program. A console program may be used to perform tasks such as configuring the virtual machine, starting and killing processes, and checking and collecting status information of processes. The network of independent PVM pvmds forms the basis for support of important features for a cluster computing environment. These features include dynamic reconfigurability, fault tolerance and scalability.

PVM provides dynamic reconfigurability by allowing hosts to enter and exit the virtual machine via notification messages [42]. PVM version 3 also supports the notion of dynamic process groups. Processes can belong to multiple named groups, and groups can be changed dynamically at any time during a computation. Functions

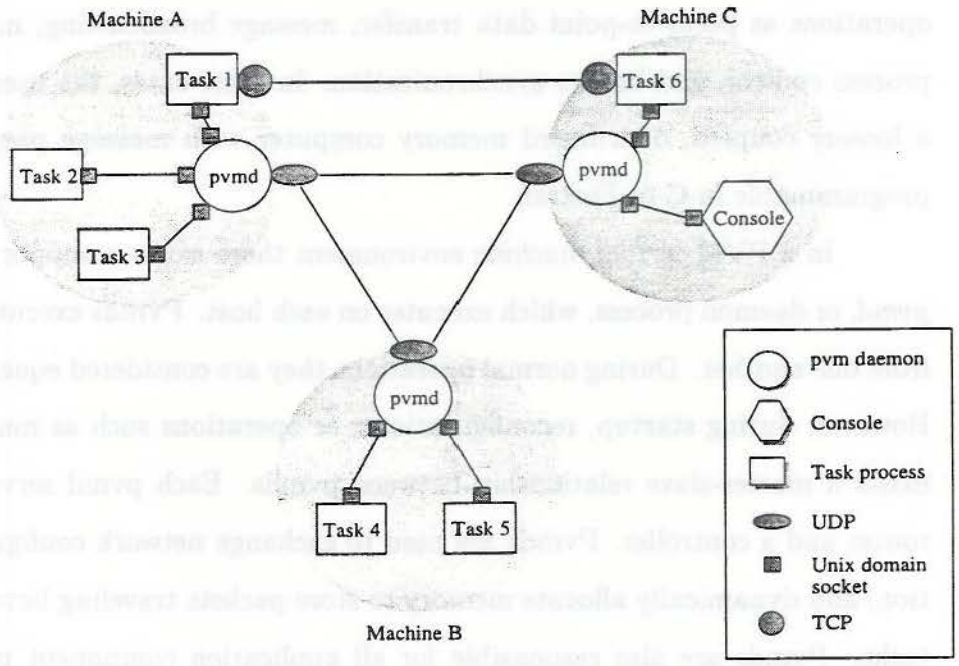

Figure 2.2: An instance of PVM configuration

that logically deal with groups of tasks such as broadcast and barrier synchronization use the user's explicitly defined group names as arguments. Routines are provided for processes to join and leave a named group. This dynamic reconfigurability also provides support for scalability and fault tolerance.

Since management is decentralized and localized, a PVM virtual machine may potentially scale up to hundreds of hosts executing thousands of tasks. However, the largest reported virtual machines consist of approximately 100 hosts [42]. This is due to, in part, the lack of availability of high-speed networks. Also, in general, there do not exist interesting algorithms which can make use of hundreds of relatively fast processors interconnected by a low-speed network. The growing availability of high-speed networks may make very large virtual machines more likely and feasible.

2.2.1 The Communication Subsystem of PVM

As shown in Figure 2.1, the three main components of PVM are the *pvmd* daemon program, *libpvm* programming library, and the applications which are running as PVM tasks. In this section, we discuss the role of *pvmd* daemon programs and the *libpvm* library in the communication subsystem of PVM. We also describe how these components employ transport protocols and control protocols to provide process control, dynamic configuration, and reliable transmission of messages and packets.

2.2.1.1 Components of Communication Subsystem

The *pvmd* daemon process is running on each participating host. The pvmd serves as a message router and process controller. As a message router, the pvmd provides intra-host and inter-host communications. Local tasks (application processes running on the same host) can exchange messages with one another through the pvmd. Local tasks can also request pvmd to forward messages to remote tasks (application processes running on different hosts). The message exchange between local tasks and remote tasks will be discussed in detail in Section 2.2.1.4.

The first pvmd manually started by a user is the *master pvmd*, others invoked by the master pvmd are *slave pvmds*. Many of the control and management operations are done by the collaboration of the master pvmd and slave pvmds. Some of the operations are: startup of remote PVM tasks, addition or deletion of hosts, and fault detection and recovery. A special pvmd, called *shadow pvmd*, is used by the master pvmd to startup new slave pvmds. The shadow pvmd runs on the same host as the master pvmd. The purpose of using shadow pvmd is to prevent the master pvmd from blocking because of the startup of slave pvmds. The pvmd is more robust than application tasks. An idle pvmd will occasionally check if others are still running. The main body of pvmd is a work loop which repeatedly executes the following jobs.

- Send packets to remote pvmds. The packet could contain a message input from local tasks. In this case, the pvmd acts like a message router. The packet could also be control messages that related a certain task between pvmds. In Section 2.2.1.3, we will discuss the control and management protocols used between pvmds and between a pvmd and a task.

- Receive packets from a remote pvmd. If these are messages carried in the packet, pvmd will forward the message to the corresponding local tasks. The packet might be a control message, such as the *host table* issued from the master pvmd to slave pvmds. The host table describes the configuration of the virtual machine.

- Accept the new connection from a task. The first time an application uses a PVM system call, the process becomes a new PVM task by requesting connection with its local pvmd. The pvmd authenticates the new PVM task before granting the new connection.

- Output messages to local tasks. The pvmd forwards messages destined to local tasks. The message could come from remote tasks or local tasks.

- Input messages from local tasks. The pvmd receives a message from local tasks. The pvmd uses Unix domain socket for local message and standard network protocol for forwarding the message to remote sites.

The *libpvm* library provides a programming interface which allows applications to communicate with the pvmd and other tasks. The libpvm contains routines for conversion of data formats, message passing, and PVM system calls for process control, dynamic configuration, and barrier synchronization. A fast message passing mechanism based on TCP/IP is also included in the libpvm library. The message

passing mechanism sets up direct connections between two tasks instead of using the routing function provided by the pvmd. For example, in Figure 2.2, there is a TCP connection between task 1 on Machine A and task 6 on Machine C. Each application process is linked with the libpvm library and executed as a PVM task. The message passing model supported by the PVM requires users to partition their tasks properly based on their distributed algorithms.

2.2.1.2 Transport Protocols

The communication subsystem of PVM is based on TCP, UDP, and Unix domain socket. TCP is a stream-oriented protocol. It supports the reliable, sequenced and unduplicated flow of data without record boundaries. UDP is a datagram protocol which is conceptually similar to conventional packet switched networks such as Ethernet. Because of the consideration of scalability, overhead, and fault tolerance, PVM uses UDP sockets for the communications between pvmds. UDP is a datagram protocol which provides a connectionless unreliable datagram service. Messages delivered via UDP sockets are not guaranteed to be in-order, reliable or unduplicated. Therefore, PVM uses its own acknowledgment and retransmission mechanism on top of UDP sockets for reliable transmission.

For the communications among local tasks and between tasks and the pvmd, PVM uses Unix domain sockets as the message exchange mechanism. In earlier versions of PVM (before version 3.3), TCP sockets were used as the message exchange mechanism. The Unix domain sockets were later adopted for better transfer rate and lower latency. A local task can use the following routing mode to communicate with remote tasks: (1) *Normal Routing* mode, and (2) *Direct Routing* mode. We will discuss these two routing modes in Section 2.2.1.4. In Table 2.1, we summarize the transport protocols or inter-process communications used by PVM.

Table 2.1: Transport protocols or inter-process communications used by PVM.

	local task	remote task	local pvmd	remote pvmd
local task	Unix domain	TCP or pvmd	Unix domain	
remote task	TCP or pvmd			
local pvmd	Unix domain			UDP with ACK
remote pvmd			UDP with ACK[2]	

PVM has its own header formats in messages and packets. The message header contains a integer tag for message type and an encoding field to pass the encoding style of the message. The message type can be used by PVM tasks to differentiate messages. The encoding field is used to exchange messages in a heterogeneous environment which uses different data formats. The header has two formats, one for packets exchanging among pvmds and the other one for packets exchanging between the pvmd and tasks. Some of the common fields appear in both formats are source task identifier and destination task identifier. Because the data streams between entities of PVM contain headers and data. A PVM entity always reads header first and then the message or packet body. A simple optimization can be applied to reduce the read operation from two-step reading to one-step reading.

2.2.1.3 Control and Management Protocols

Based on the communication mechanism provided by TCP, UDP and Unix domain sockets, PVM uses its high-level protocols for control and management purposes. The high-level control and management protocols are Task-Daemon protocol and Daemon-Daemon protocol. Many configuration operations are performed by employing both protocols. For example, the addition of a new host (start a new pvmd) is performed by pvmds with Daemon-Daemon protocol. A case of employing both protocols to set up a multicasting connection is presented in Section 2.2.1.4.

There are two types of important interactions which are frequently used: *task-to-pvmd* communication and *pvmd-to-pvmd* communication. As mentioned above, task-to-pvmd communication occurs via Unix domain sockets, and pvmd-to-pvmd communications occurs via UDP in the Normal mode. We illustrate how these interactions occur by discussing some common PVM constructs.

pvm_addhost() is executed when a new host joins a virtual machine. A task on an existing host of the virtual machine, either on the master host or a slave host, initiates the *pvm_addhost()* function. If a task on an existing slave host initiates the function, a task-to-pvmd interaction occurs, i.e., the initiating task sends an "add host" message to its local pvmd and waits for an acknowledgment. Then the local pvmd of the initiating host sends an "add host" message to the master pvmd, i.e., a pvmd-to-pvmd communication occurs. The master pvmd then forks a process pvmd on the new host and takes other appropriate actions to include the new host in the virtual machine. When these actions are completed, the master daemon sends an acknowledgment to the local pvmd of the initiating slave host, i.e., pvmd-to-pvmd interaction, and the local pvmd sends an acknowledgment to the task, i.e., task-to-pvmd interaction. Similar actions occur if the initiating task is on the master host.

Some other PVM constructs which use both task-to-pvmd and pvmd-pvmd interactions include *pvm_delhost()*, delete hosts from a virtual machine, and *pvm_mcast*, execute a multicast operation. An example of a PVM construct which uses only task-to-pvmd interaction is *pvm_config()*, where a task fetches its machine configuration information from its local pvmd.

2.2.1.4 Message and Packet Routing

In this section, we first briefly introduce the PVM system calls used for message passing. Then discuss the two message routing options available for communications

between local PVM tasks and remote tasks. We also provide a detail description of PVM's implementation of multicasting communications.

PVM System Calls for Message Passing

Sending a message with PVM is composed of three steps. First, a send buffer must be initialized by a call to pvm_initsend() or pvm_mkbuf(). Second, the message must be packed into a buffer using any number of pvm_pk*() routines. Each of the pvm_pk*() routines packs an array of a given data type into an active send buffer. The pvm_pk*() routines also perform data type conversation in a heterogeneous environment. Calls to pvm_unpk*() routines unpack the message from the active receive buffer into an array of a given data type. Third, the message is sent to another process by calling the pvm_send() routine or the pvm_mcast() (multicasting) routine. The message is received by calling either a blocking receive using pvm_recv() or a non-blocking receive using pvm_probe() and pvm_recv().

Normal Routing Mode and Direct Routing Mode

PVM provides two types of communication modes, *Normal Routing* mode and *Direct Routing* mode. In the Normal Routing mode, in order for a source task to communicate with a remote task, it must first communicate through a Unix domain socket to its local pvmd daemon. The local pvmd daemon then communicates through a UDP socket to the remote pvmd. The remote pvmd then communicates to the destination task through a Unix domain socket. Thus two Unix domain connections and two UDP connections are required for a bi-directional communication between the two communicating application processes. In the Direct Routing mode, PVM sets up a direct TCP connection between the two communicating processes or tasks. The detailed transmission facilities of the Direct and Normal Routing modes are hidden from the end-users. Figure 2.3 depicts these two communication modes.

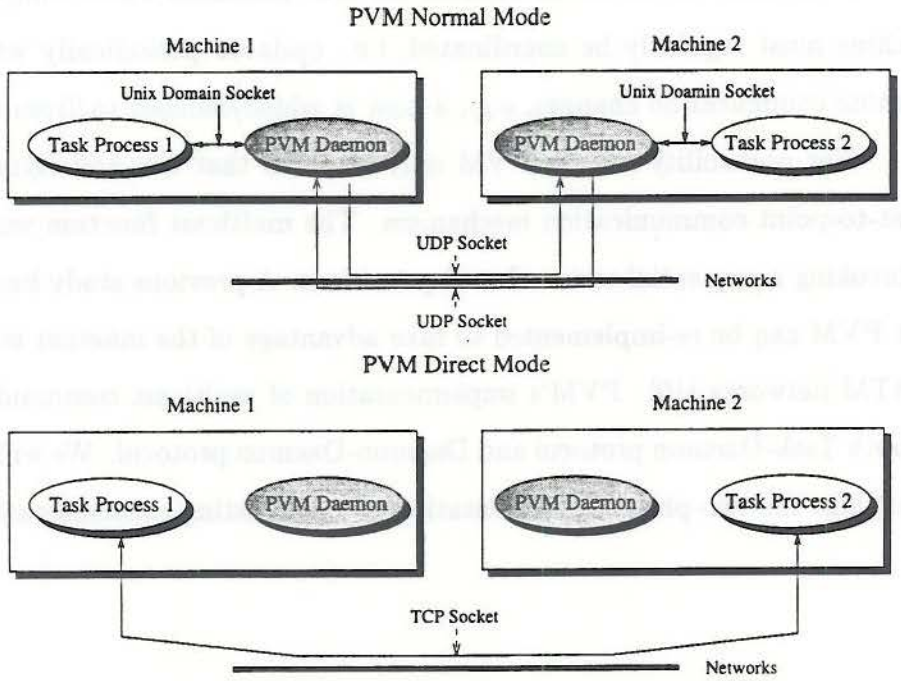

Figure 2.3: Comparison of PVM Normal and PVM Direct modes

The advantage of the Direct Routing mode is that it provides a more efficient communication path than the Normal Routing mode. A previous report observed more than a twofold increase in communication performance when using Direct Routing mode [41]. The main reason PVM provides the Normal Routing mode, despite its lower performance, is because of the limited number of file descriptors some Unix systems provide. Each open Unix domain or TCP connection consumes a file descriptor. Some operating systems limit the number of open files to as few as 32. If a virtual machine consists of N hosts, each machine must have $N-1$ connections to the other hosts. Thus the drawback of the Direct Routing mode is its limited scalability [42].

Multicasting

Efficient support for multicasting is important because multicast data flow patterns are often found in parallel programming applications. It is also important in

a virtual machine environment because the host machines which comprise the virtual machine must regularly be coordinated, i.e., updated periodically when the virtual machine configuration changes, e.g., a host is added/deleted to/from the host pool.

For portability reason, PVM only assumes that the underlying network has point-to-point communication mechanism. The multicast function was implemented by invoking a sequential series of send primitives. A previous study has demonstrated that PVM can be re-implemented to take advantage of the inherent multicast nature of ATM networks [10]. PVM's implementation of multicast communications replies on both Task-Daemon protocol and Daemon-Daemon protocol. We will use Figure 2.4 to explain the two-phase implementation of multicasting communications.

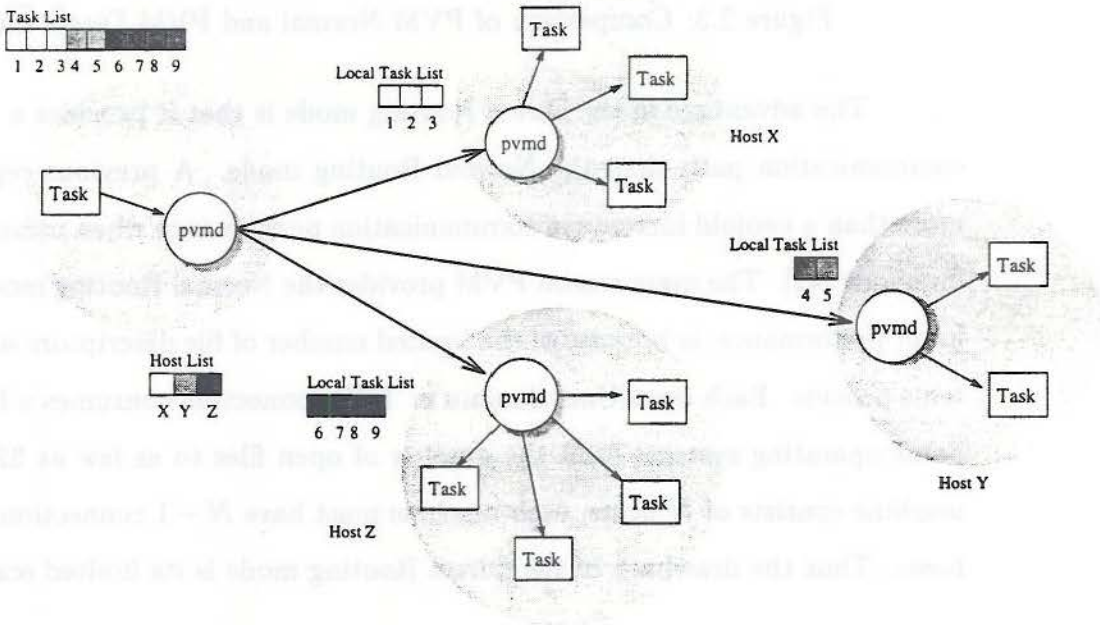

Figure 2.4: Two-phase multicasting communications.

- *Phase I: Set up a multicasting connection.* To initiate a multicast communication, the sender task uses the Task-Daemon protocol for requesting the local pvmd to setup a multicasting connection. The sender task sends the task id list and the message to its local pvmd. The task id list consists of a list of tasks which will be receiving the multicasting message. The sender's pvmd sorts the list according to the receiving hosts. For example, in Figure 2.4, Host X corresponds to the unshaded boxes (task id 1, 2, and 3); Host Y corresponds to the lightly shaded boxes (task id 4 and 5); Host Z corresponds to the darkly shaded boxes (task id 6 to 9). The task id list is distributed to the pvmd of each host which has a task in the sender's task id list. N sends are required if there are N remote receiving hosts. Now the sender's local pvmd need only retain a list of the hosts, instead of a list of the task ids, for which the data must be multicasted. The receiving hosts now have a list of local tasks for which the multicast is intended. The multicasting of task id list is performed by pvmds with the Daemon-Daemon protocol. Finally, a unique multicasting address is assigned to each *pvm_mcast()* and sent back the sender's task. This also completes the Task-Daemon protocol between the sender task and its local pvmd.

- *Phase II: Message forwarding.* The sender's pvmd multicasts the data to the pvmds of all the hosts on the host list via N serial sends (assuming there are N remote receiving hosts). Each receiving pvmd receives the data and then distributes it to the appropriate tasks on its local task list.

2.3 Enhanced PVM Communications on ATM Networks

Emulating a parallel machine via a collection of homogeneous independent

hosts and a general-purpose local area network has obvious advantages such as cost-effectiveness and large aggregate processing power and memory. However, the ability of most current general-purpose local area networks to support communication-intensive parallel applications has been questionable. Today with the emergence of several high-speed switch-based networks, such as HIPPI, Fibre Channel, and ATM, the possibility of networks effectively supporting communication-intensive parallel applications may soon prove a reality.

Several advantages to cluster computing exists. First, by using independent commercially available systems and a general local area network, advances in processor and network technology may be readily incorporated. Second, due to the large amount of memory and processing power available in the aggregate collection of individual host systems, very large applications may be executed using a collection of relatively low-priced host systems. Third, the underlying network may be able to support high-speed I/O to applications, for instance, by using disk arrays.

One of the factors which previously caused much skepticism on the feasibility of network-based parallel computing was the limitations imposed by using traditional local area networks, such as an Ethernet, as the system interconnect. For many typical network applications which require only occasional file transfers, or infrequent small amounts of data to be transmitted between workstations, an Ethernet based cluster of workstations is adequate. However, for network-based applications, such as communication intensive, course grain parallel applications, it is well known that traditional networks such as Ethernet are incapable of providing adequate performance. Thus in our study we have chosen to use a high-speed transport mode as the supporting communication medium.

2.3.1 PVM and ATM Advantages

The following enumerates the advantages of the particular parallel programming environment (PVM) and high speed network platform (ATM) we have chosen as the basis for our cluster computing environment.

- *Fast message passing.* Given an environment consisting of a cluster of workstations interconnected by a local area network, it is well-known among parallel programmers that message passing facilities lack the performance found in distributed memory computers such as the Connection Machine CM-5 or the nCube. This is because most distributed memory computers provide specialized hardware - high speed switch(es) and interconnect hardware - which provide speeds and latencies that local area networks cannot match. This is especially true about current available slow speed (e.g., Ethernet) local area networks. Thus PVM designers originally developed their system with assumptions about the underlying network being slow and unreliable. For this reason, as well as for portability reasons, the BSD socket programming interface was chosen to act as the interface between the message passing interface and the network medium. Fast message passing via PVM is possible via two changes. First, the change in the underlying network to a high speed network, such as ATM, greatly increases the message passing speed. Second, bypassing the BSD socket programming interface and using a lower layer protocol reduces overhead, and thus increases message passing speed.

- *PVM application portability.* As mentioned before, PVM is a widely used message passing library for distributed computing. It is available to the public and is easily installable. Moreover, several vendors including Cray Research, Convex, SGI, IBM, DEC and Thinking Machines have committed to supplying and supporting PVM on their systems. Thus PVM programs are portable from

many distributed memory machines to a cluster of workstations interconnected by a local or wide area network.

- *Network flexibility.* ATM provides a great deal of flexibility in terms of supporting varying types of application traffic. It may support constant bit rate traffic, variable bit rate traffic, traffic with low data loss tolerances, high bandwidth requirements, and delay-sensitive data. Much of ATM's flexibility is due to its transport mechanisms such as its fixed size data cells, and switch-based network architecture. Features of ATM are explained in greater detail in Section 2.3.2.

- *Availability of high-speed network components.* The past year has seen the burgeoning of high speed local area networks, namely ATM networks. Although the ATM standard is not yet fully complete, the major aspects of the definition of the ATM high speed transport mechanism are complete. Optical lines based upon SONET, the most commonly associated physical layer for ATM, are available. ATM local area switches and ATM interface cards for most workstations are also readily available. Current market forces as well as ATM's wide-spread acceptance in the networking community has caused it become the most likely transport mode for high-speed local and wide-area networking [41].

In this study, we sought to achieve high performance (e.g., low latency, high bandwidth) not only by implementing PVM on a high speed medium, such as ATM, but also by minimizing possible sources of overhead. Overhead is incurred by hardware and software components. Hardware overhead, which is incurred by memory and bus architecture of the host, the network interface board, the switch, and the signal propagation delay, is a function of the particular system components which are used, and hence considered unavoidable in this study. Software overhead is incurred by interactions with the host system's operating system, device driver and higher

layer protocols. The device driver overhead is mainly caused by the design of the host interface and the bus architecture of the host computer. The overhead incurred by the high-level protocols and the delay caused by the interactions with the host operating system can be varied by using different Application Programming Interfaces (APIs) which are available on different communication layers.

A protocol stack is a conceptual diagram where each layer in the stack corresponds to a set of services provided to the adjacent higher layer. For example, the TCP and UDP layer corresponds to the transport layer, as defined by the Open Systems Interconnection Reference Model. The major service that the transport layer protocol provide to the higher layer is end-to-end service, i.e., transport from the source machine to the destination machine. TCP provides reliable connection-oriented service; UDP provides unreliable packet-oriented service.

Figure 2.1 depicts the layers of a protocol stack. A user application message would have to be processed at each layer of the stack, beginning at the application layer, until it is finally in the form suitable for physical transmission via the network medium. Each layer performs processing on the user message by fragmenting/reassembling the message and appending/stripping the appropriate headers, depending upon whether the message is traversing down or up the protocol stack. Figure 2.1 shows how PVM may be implemented on the BSD socket programming interface (on the right side of Figure 2.1) or directly on the ATM API (on the left side of Figure 2.1). Since, the ATM API resides at a lower layer in the protocol stack, the overhead incurred when directly using this API is expected to be lower than the overhead incurred by using the BSD sockets programming API. A previous study [41] validated this performance gain when evaluating the performance of four API's: Fore Systems' ATM API [18], BSD socket programming interface [39, 45], Sun's Remote Procedure Call (RPC) [45], and PVM over the BSD socket programming interface [42]. The Fore Systems' ATM API provided the best performance of the four APIs.

The existing PVM message passing library is implemented using the BSD socket programming interface. The transport protocols used are TCP and UDP. In order to provide as close to optimal performance as possible, in our study, PVM was implemented directly over the ATM Adaptation layer protocol via the Fore Systems' ATM API instead of the BSD socket interface. The experimental environment for this study consisted of several workstations interconnected via a Fore Systems' ASX-100 ATM switch. Details of this environment are discussed in Section 2.3.5.

Despite the performance gains of using a lower layer protocol, the consequential drawback is that it lacks features found in higher layer APIs such as distributed programming support, loss-free transmission, and flow control. It also lacks the portability found in the original PVM over BSD socket programming interface. In our study, we provided two main enhancements to the existing communications facilities. Since the Fore Systems' ATM API provides only "best-effort" delivery and no flow control, we implemented an end-to-end protocol which provides cell retransmissions as well as imposes flow control. We also took advantage of the inherent multicasting capability provided by the ATM switch to significantly improve upon existing PVM multicasting functionality.

2.3.2 ATM: the Next Generation Network

ATM [6, 36] is a standard developed by the networking standards community (CCITT) which specifies the network layer protocol of broadband networks (B-ISDNs - Broadband Integrated Services Digital Network). It specifies a fast packet switched network where data is fragmented into fixed-size 53 byte cells. Cells consist of 53 bytes - a 5 byte header and a 48 byte information payload. ATM resides above the physical layer and directly below the ATM Adaptation Layer (AAL).

ATM is expected to serve as the transport network for a wide spectrum of

traffic types with varying performance requirements. Using the statistical sharing of network resources (e.g. bandwidth, processing buffers, etc.), it is expected to efficiently support multiple transport rates from multiple users with stringent requirements on loss, end-to-end delay, and cell-interarrival delay. Even though the ATM standard was initially developed and intended to serve as an infrastructure for wide-area (telecommunications) networks, it is currently being much more rapidly adopted for local area networks.

ATM is distinguished from conventional local area networks, such as Ethernet and FDDI, by the following features:

- *Connection-oriented service.* ATM provides a virtual connection for any two physically dislocated processes which wish to communicate. All cells from the same call traverse the same physical path, or virtual connection. Virtual connections are specified by a virtual circuit identifier (VCI) and virtual path identifier (VPI), found in each cell header. The VPI and VCI are used for multiplexing, demultiplexing, and switching the cells through the network. ATM connection-oriented service has the potential to provide low-latency.

- *High data transfer rates.* ATM is independent of any particular physical layer, but is most commonly associated with Synchronous Optical Network (SONET). SONET defines a standard set of optical interfaces for network transport. It is a hierarchy of optical signals that are multiples of a basic signal rate of 51.84 Mbits/sec called OC-1 (Optical Carrier Level 1). OC-3 (155.52 Mbits/sec) and OC-12 (622.08 Mbits/sec) have been designated as the customer access rates in B-ISDN. OC-3, 155 Mbits/sec, is the rate currently supported by first generation ATM networks. Recall, that the aggregate throughput of current available high-speed shared-medium networks, such as FDDI, is 100 Mbits/sec. Since ATM is a switch-based network architecture, the aggregate throughput

is usually several gigabits. The ASX-100 Fore switch, used in our experiments, provides an aggregate throughput of 2.4 Gbits/sec. Each host on an OC-3 ATM network has access to a link speed of 155 Mbits/sec. In a FDDI network, all hosts attached to the network must share the same 100 Mbits/sec network capacity.

- *Support for multiple classes of service.* ATM was intended for the support of multiple classes of service, i.e., classes of traffic with varying quality of service parameters such as cell loss, delay, cell inter-arrival times, and data transfer rates. These parameters reflect the varying types of traffic ATM was intended to support, such as connection-oriented traffic types (e.g., audio), connection-less traffic types (e.g., file transfers), etc. The purpose of the ATM adaptation layer (AAL) is to provide a link between the services required by higher network layers and the generic ATM cells used by the ATM layer. Five AAL protocols are defined for various types of services, such as constant bit rate services, connection-oriented/connection-less services, etc.

The overriding factor which distinguishes ATM from other network architectures lies in its flexibility. It is based upon a high-speed medium and thus provides the basic infrastructure for supporting high-speed transport. It also provides a network architecture which is based upon fast packet switching which is suitable for a wide range of applications.

2.3.3 Application Programming Interface

Figure 2.1 depicts the protocol stack from the application layer to the network transport (ATM) layer. From this figure, we note that there are two possible APIs which we can use to interface to the ATM AAL layers - namely the BSD socket

programming interface which includes TCP/IP and UDP/IP, or the ATM API. In this study, we sought to minimize unnecessary overhead [41], and hence chose to implement PVM on the Fore Systems' ATM API rather than the BSD socket interface.

The Fore Systems' ATM API library routines support the client-server model. Consistent with ATM specifications, a connection (Switched Virtual Circuit or Permanent Virtual Circuit) must be established before data can be exchanged between a client and a server. Typical connection-oriented client-server interactions described below may then proceed.

The Fore Systems' user-level ATM library routines provide a socket-like interface. Applications first use *atm_open()* to open a file descriptor and then bind a local Application Service Access Point (ASAP) to the file descriptor with *atm_bind()*. Each ASAP is unique for a given end-system and is comprised of an ATM switch identifier and a port number on the switch. Connections are established using *atm_connect()* within the client process in combination with *atm_listen()* and *atm_accept()* within the server process. These operations allow the data transfer to be specified as simplex, duplex, or multicast. *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 selected for the connection and the constraints of the underlying socket-based or stream-based device driver implementation. Applications can select the type of ATM AAL to be used for data exchange. In the Fore Systems' implementation, AAL Types 1 and 2 are not currently supported by Series-200 interfaces, and Type 3 and Type 4 are treated identically.

Bandwidth resources are reserved for each connection. Resource allocation is based upon the following three user specifications: (i) peak bandwidth, the maximum data injection rate which the source may transmit, (ii) mean bandwidth, the average bandwidth over the lifetime of the connection, and (iii) mean burst length, the average amount of data sent at the peak bandwidth. The network control function will

compute the chances that the requested connection will create buffer overflow (cell loss) and consequentially accept or reject the connection request.

If the connection is accepted, an ATM VPI and VCI is allocated by the network. The device driver associates the VPI/VCI with an ASAP, which is in turn associated with a file descriptor. Bandwidth resources are then reserved for the accepted connection. The network then makes a "best-effort" attempt to deliver the ATM cells to the destination. A "best-effort" attempt implies that during transmission, cells may be dropped depending on the available resources. End-to-end flow control between hosts and cell retransmissions are left to the higher layers.

2.3.4 PVM Communications: Existing and Enhanced

In order to enhance the performance and functionality of the existing PVM communication facilities, we made several changes to the existing PVM platform. First, we bypassed the BSD socket interface and directly implemented PVM on the lower level Fore Systems' ATM API. This resulted in a performance gain discussed in Section 2.3.5.3. Since the Fore Systems' ATM API only provides "best-effort" delivery, as opposed to the reliable delivery TCP provides, we implemented an end-to-end flow control protocol which ensures reliability via a selective retransmission mechanism. The second change we made was to improve PVM's multicasting capabilities. PVM assumes the underlying network cannot support multicast. We capitalized on the inherent multicasting capabilities of ATM, and re-implemented a more efficient multicasting operation. Section 2.3.5.4 presents the performance gain of the re-implemented multicasting operation. The following two subsections describe the two implementations.

2.3.4.1 End-to-End Flow Control Protocol with Selective Retransmissions

As mentioned in the previous section, the Fore Systems' ATM API provides only "best-effort" transmission (i.e. a message may be lost during its transmission) and no flow control. In order to guarantee the delivery of user messages while not sacrificing performance, we developed a flow control scheme which incorporates a selective retransmission scheme.

Our experimental results revealed that user messages begin to experience losses as the message size increases beyond 256 KBytes [41, 10]. So in our scheme, we chose to divide each user message into 256 KBytes segments. The message unit that Fore Systems' implementation of ATM recognizes is 4 KBytes. Thus, at the ATM layer, each user message segment of 256 KBytes is divided into 64 4 KBytes segments.

In the selective re-transmission flow control algorithm, the sender begins by setting and starting a timer, timer1, for the next user message. The sender sends out a number of 4 KB segments until either an entire 256 KB segment has been transmitted or the end of the user message has been reached. The sender then sets and starts another timer, timer2, and sets the number of retry attempts, n. The sender then waits for either the timer to expire or a (selective) acknowledgment to be received. If the sender receives the acknowledgment, it checks to see if any of the 4 KB segments were lost. If not, the sender process repeats the algorithm by transmitting the next set of 64 4-KB segments, otherwise, the missing packets are re-sent, the number of re-tries is decremented and the sender waits again for an acknowledgment. If the sender times out (does not receive the acknowledgment), it resends the previous segment, decrements the number of re-tries, resets the timer and begins the process of waiting for an acknowledgment again.

The receiver process acknowledges (by sending a selective retransmission or

acknowledgment packet) to the sender, when it receives the last 4 KB segment of a 256 KB segment or the last 4 KB of the user message, or when it (the receiver) times out. When the receiver acknowledges a 256 KBytes segment, it does so by sending a 64 bit bit-map, where each bit signifies whether the corresponding 4 KBytes segment had been received or not.

2.3.4.2 Multicasting Protocol

PVM implements the multicast function by invoking a sequential series of send primitives. By taking advantage of the inherent multicast nature of ATM, we re-implemented the multicast function to occur as a parallel send to multiple receivers. Both multicast implementations occurs in two phases. Our re-implementation of the multicasting operation is the same as the above two-phase operation except the N serial sends are replaced (in both phases) by a simultaneous send to N remote receiving hosts. Performance results of this re-implementation and the original PVM multicast operation are presented in Section 2.3.5.4.

2.3.5 Performance Measurements

2.3.5.1 Experimental Network Computing Environment

The ATM environment was provided by the MAGIC (Multidimensional Applications and Gigabit Internetwork Consortium) project and the Army High Performance Computing Research Center at the University of Minnesota. Fore Systems, Inc. host adapters and local area switches were used. The host adapter was a Series-200 interface for the Sun SBus. The physical media for the Series-200 adapter was the 100 Mbits/sec TAXI interface (FDDI fiber and signal encoding scheme). The local

area switch was a Fore ASX-100. Four Sun Sparc 2 machines and two Sun 4/690 machines were directly connected to the Fore switch.

The Series-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 cell level segmentation and re-assembly (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 the i960. The i960 uses local memory to manage pointers to packets, and uses DMA (Direct Memory Access) to move cells out of and into host memory. Cells are never stored in adapter memory.

The ASX-100 local ATM switch is based on a 2.4 Gbits/sec (gigabit per second) switch fabric and a RISC control processor. The ASX-100 supports Fore's SPANS signaling protocol, and can establish either Switched Virtual Circuits (SVCs) or Permanent Virtual Circuits (PVCs). All of the experiments conducted ignored circuit setup time and thus the ATM circuits used can be viewed as PVCs.

2.3.5.2 Echo Programs

The echo program is used for measuring the end-to-end communication latency between two machines. In this program, a client sends a M-byte message to a server and waits to receive the M byte message back. The client/server repeats this interaction N times. The round trip timing for each iteration in the client process is collected. The timing starts when the client sends the M byte message to the server, and ends when the client receives M bytes of the response message. Thus the problem of synchronizing clocks in two different machines is avoided. The communication latency for sending a M-byte message can be estimated as half of the total round-trip time. The communication throughput is calculated by dividing $2 \times M$ by

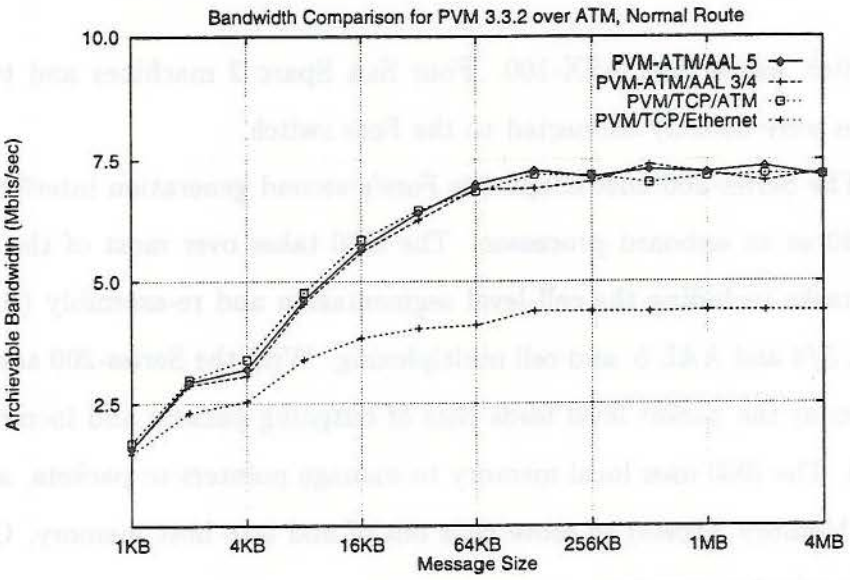

Figure 2.5: Normal mode: Bandwidth as a function of message size

the round-trip time (since $2 \times M$ bytes of message have been physically transmitted). A previous study [41] presents the round trip delay (milliseconds) as a function of message size for ATM/AAL5.

2.3.5.3 End-to-End Performance

We measured the performance of four different PVM platforms. PVM-ATM (AAL4) and PVM-ATM (AAL5) refers to the implementation of PVM directly on the Fore Systems' ATM API with the appropriate adaptation layer. PVM/TCP/ATM refers to the implementation of PVM on the BSD socket programming interface on an ATM network. PVM/TCP/Ethernet refers to the implementation of PVM on the BSD socket programming interface on an Ethernet network.

Figure 2.5 shows bandwidth as a function of message size for messages using the Normal route. PVM-ATM (AAL4) achieves the highest maximum bandwidth of 7.403 Mbits/sec. PVM-ATM (AAL5) and PVM/TCP/ATM achieve close to PVM-ATM (AAL4) bandwidth measurements, i.e., their maximum bandwidth measurements are

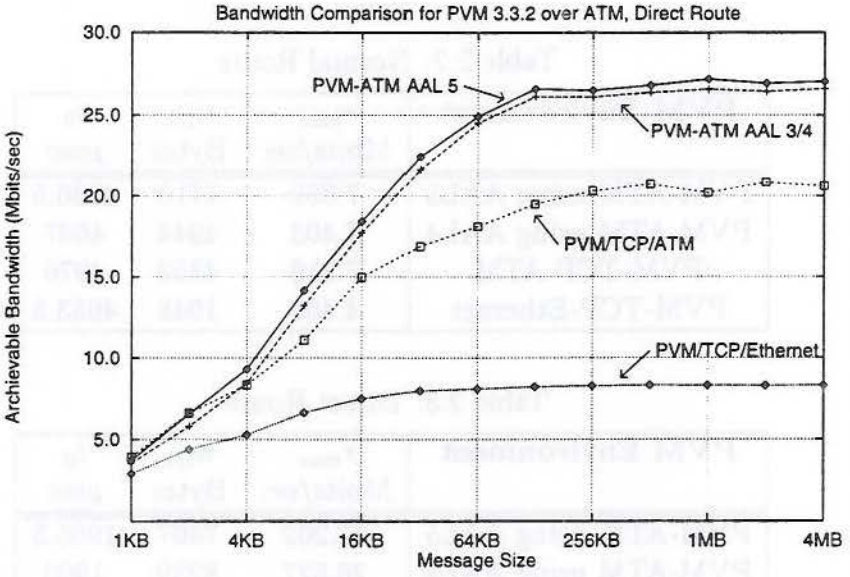

Figure 2.6: Direct mode: Bandwidth as a function of message size

within 0.2 Mbits/sec of each other. PVM/TCP/Ethernet achieves a maximum bandwidth measurement of 4.406 Mbits/sec, approximately 60% that achieved by PVM-ATM (AAL4), PVM-ATM (AAL5) and PVM/TCP/ATM. We conclude from these results that, when using the Normal mode, the significant performance gain occurs primarily from using the high-speed ATM medium as opposed to the slower-speed Ethernet medium.

Figure 2.6 shows bandwidth as a function of message size for messages using the Direct route. PVM-ATM (AAL5) achieves the highest maximum bandwidth of 27.202 Mbits/sec. PVM-ATM (AAL4) achieves close to PVM-ATM (AAL5) bandwidth measurements, i.e., their maximum bandwidth measurements are within 0.6 Mbits/sec of each other. PVM/TCP/ATM achieves a maximum bandwidth value of 20.826 Mbits/sec. And PVM/TCP/Ethernet achieves a maximum bandwidth measurement of 8.312 Mbits/sec. From these results, we conclude that when PVM bypasses TCP and directly uses the ATM API a rather significant performance gain

Table 2.2: Normal Route

PVM Environment	r_{max} Mbits/sec	$n_{1/2}$ Bytes	t_0 μsec
PVM-ATM using AAL5	7.369	4710	4986.5
PVM-ATM using AAL4	7.403	4944	4687
PVM-TCP-ATM	7.216	4352	4076
PVM-TCP-Ethernet	4.406	1948	4053.5

Table 2.3: Direct Route

PVM Environment	r_{max} Mbits/sec	$n_{1/2}$ Bytes	t_0 μsec
PVM-ATM using AAL5	27.202	7867	1905.5
PVM-ATM using AAL4	26.627	8239	1903
PVM/TCP/ATM	20.826	7649	1839
PVM/TCP/Ethernet	8.312	1945	1541

occurs of approximately 6 to 7 Mbits/sec. Again we observe a significant performance gain when using ATM as opposed to Ethernet.

The maximal achievable throughput is bounded by the speed of the TAXI interface, 100 Mbits/sec. In our previous study [41], we observed the maximum achievable throughput to be 46.08 Mbits/sec. In this study, we observed the maximum achievable throughput of PVM-ATM (AAL5) to be 27.202 Mbits/sec. Thus the overhead occurs at two levels: the end system and ATM interface (software and hardware) limits the throughput to 46.08 Mbits/sec, and the overhead from PVM limits the maximal throughput of PVM-ATM to 27.202 MBits/sec.

Tables 2.2 and 2.3 show the above measurements in terms of the following three performance metrics. These metrics are crucial to the communication performance at the application level.

- r_{max} (*maximum achievable throughput*) : the maximum achievable throughput which is obtained from experiments by transmitting very large messages. This

is an important measure for applications requiring large volumes of data transmissions.

- $n_{1/2}$ (*half performance length*) : the message size needed to achieve half that of the maximum achievable throughput. This number may not be compared with the corresponding numbers from different hardware and software configurations, since the maximum achievable throughput may be different for different configurations. This measure provides a reference point that shows the effect of message sizes on the achievable throughput. Users can observe more than half of the maximum achievable throughput with messages larger than $n_{1/2}$.
- t_0 (*startup latency*) : the time required to send a short message of 16 bytes to a receiver and receive the same message back. This is an important measure when transmitting short messages.

From both tables, PVM-ATM (AAL5), PVM-ATM (AAL4), PVM-TCP-ATM, and PVM-TCP-Ethernet provided decreasing values for t_0, respectively. The greatest time difference occurs between using ATM or Ethernet. The overhead, in terms of latency, for the ATM network is thought to be primarily caused by the device driver. It is believed that the firmware code for Ethernet has been fine-tuned for better communication latency [41].

2.3.5.4 Multicasting Measurements

On ATM, we measured the performance of the multicasting operations (PVM's original multicast operation and our re-implementation) by iteratively executing the multicast operation. During each iteration, a timer is started, the sender then performs the multicast operation and then waits to receive positive acknowledgments

from all the members of the multicast receiving group. Once all acknowledgments have been received, the timer is stopped, and another iteration begins.

In Figure 2.7, the top (bottom) two graphs depict the time to perform the multicasting operation as a function of message size using the existing PVM multicasting (our re-implementation). When using PVM's existing multicasting facilities, for message sizes of 64 KB, the time to multicast to 1, 2, 3, 4, 5 remote hosts is approximately 108, 130, 184, 235, 290 milliseconds, respectively. For message sizes of 1 MB, the time to multicast to 1, 2, 3, 4, 5 remote hosts is approximately 1550, 1800, 2650, 3200, 3850 milliseconds, respectively. When using our PVM-ATM re-implementation (bottom two graphs of Figure 2.7), we observe that when increasing the number of remote hosts of the receiving pool from 1 to 4 the largest time difference is approximately 20 milliseconds. The total latency (T_{total}) of a multicasting operation includes: (1) the latency to send out multicast messages ($T_{multicast}$), and (2) the latency to collect acknowledgments from all of the receivers ($T_{acknowledgment}$).

$$T_{total} = T_{multicast} + T_{acknowledgment} \tag{2.1}$$

To compare the two implementations of the multicasting operation, we derived the following approximate gain factor based on the message size of 64 KB:

$$\frac{70 + 37n}{95.83 + 4.17n} \tag{2.2}$$

where n is the size of the receiving pool. The numerator (denominator) was derived by examining the increase in latency caused by the increase in the number of receiving hosts when using the original PVM multicast operation (our re-implementation). At a fixed message size, the average increase per additional receiving host was used to extrapolate to the case of n receiving hosts. In Equation 2.2, the original PVM

has more per-receiver latency. This is because the multicasting operation is implemented by a series of point-to-point message passing. The per-receiver latency includes $T_{acknowledgment}$ and a portion of $T_{multicast}$.

Thus our re-implementation results in a performance gain (for 4 remote receiving hosts) of a factor of approximately two. For 10 remote hosts, our re-implementation results in a performance gain of a factor of approximately 3.2. Note that these performance gains are amortized by other PVM processing functions which occur during the PVM multicasting operation.

2.4 Enhanced PVM Communications on HIPPI Networks

In this section, we present a study of improving Parallel Virtual Machine's (PVM) communication performance over a HIPPI local area network. After a detailed examination of PVM's communication subsystem, we re-implemented PVM using the Hewlett Packard's Link Level Access (LLA) interface instead of the BSD socket interface which was originally used by PVM. From the experimental results of the performance evaluation, our study demonstrates the potential and feasibility of high-performance network computing over a high-speed switch-based local area network.

In this study, we utilized high speed networks and reduced the overhead of protocol processing. For the underlying high-speed network, we used the HIPPI [60, 61, 62, 63] as the switch-based network platform. HIPPI offers a connection-oriented service with peak data transmission rates of 800 or 1600 Mbits/sec. HIPPI is a mature technology, which is widely used in supercomputers and high-end workstations. For the parallel programming environment, we chose the popular Parallel Virtual Machine (PVM). A detailed description of PVM and HIPPI can be found in Section 2.2 and

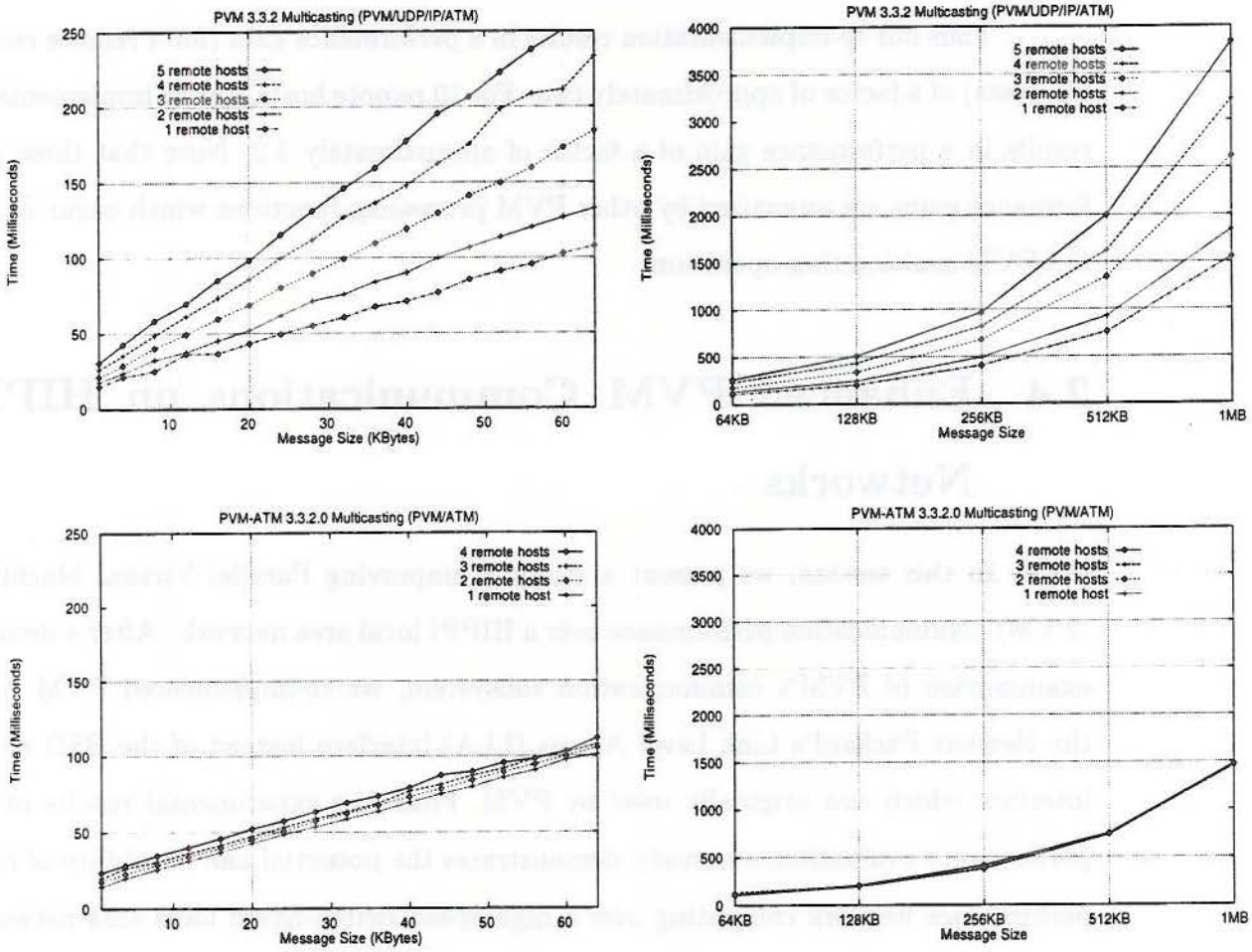

Figure 2.7: The latencies of original PVM multicasting (top graphs) and the re-implementation of PVM-ATM multicasting (bottom graphs).

2.4.1.

For our study, we sought to achieve high performance (e.g., low latency, high bandwidth) by enhancing PVM's communication subsystem to utilize the high-speed HIPPI LANs. As mentioned before, the PVM message passing library was originally implemented using the BSD socket programming interface. The transport protocols used are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). Figure 2.1 shows how PVM was implemented on the BSD socket programming interface (on the right side of Figure 2.1). The main idea of improving PVM's message passing is to reduce the overhead incurred by the high-level protocols. In order to provide as close to optimal performance as possible, in this study, part of PVM's communication subsystem is re-implemented directly using Hewlett Packard's Link Level Access (LLA) API (on the left side of Figure 2.1). We called the re-implemented version of PVM as **PVM/LLA**. Since, HP's LLA API resides at a lower layer in the protocol stack, the overhead incurred when directly using this API is expected to be lower than the overhead incurred by using the BSD sockets programming API.

A prototype of the PVM/LLA is presented in this section. The performance of the PVM/LLA is obtained by conducting a series of experiments in our test environment. The experimental environment consists of two HP 9000 series 735 workstations equipped with HP's HIPPI interface boards. The experimental measurement shows that our PVM/LLA on a HIPPI LAN can achieve comparable performance as the Message Passing Library (MPL) in IBM's scalable POWERparallel system SP2 [52]. The performance measurement also demonstrates that clusters of workstations inter-connected with switch-based high-speed LANs can be used for high-performance network computing.

2.4.1 HIPPI Networks

The High-Performance Parallel Interface (HIPPI) [60, 61, 62, 63] is one of the high-speed network or channel solutions commercially available. HIPPI is a simplex point-to-point interface for transferring data at peak rates of 800 or 1600 Mbits/sec over distances up to 25 meters. A related standard defines the usage of a crossbar switch to support multiple interconnections between HIPPI interfaces on different hosts [63]. Standards [62, 49] were also defined for running standard network protocols, such as TCP/IP and UDP/IP, over HIPPI. To extend HIPPI's connectivity, an implementor's agreement (the Serial-HIPPI [25]) specifies how the HIPPI packets are to be carried over a pair of fiber optical cables. The HIPPI can be extended up to 10 km on single-mode fiber.

HIPPI provides reliable communication and connection oriented service among hosts. With the crossbar switch, HIPPI can be used as a high-speed LAN. Multiple simultaneous connections can exist through a switch with their own switch resource. All of the connections can pass data concurrently at full HIPPI speed. The HIPPI network is suitable for distributed applications and network attached storage which may require many simultaneous interactions. The reliable communication provided by HIPPI is based on its credit-based flow control. The credit mechanism provides positive flow control to prevent buffer overflow at the receiving-side. The flow control is performed at the physical layer. HIPPI is a mature technology, most supercomputers and many high-end workstations are equipped with HIPPI interfaces for high-throughput data connections. The success and widespread use of HIPPI is due to its *"KISS"* (Keep It Sweet and Simple) design philosophy[55].

2.4.2 A Re-implementation of PVM over a HIPPI Network

The prototype of a re-implementation of PVM Version 3.3.4 (PVM/LLA) over

a HIPPI LAN is presented in this section. As shown in Figure 2.1, the communication subsystem of PVM was originally designed to use the BSD socket interface which is a common interface for accessing standard network protocols and inter-process communications. To reduce the overhead of protocol processing and utilize the throughput of underlying networks, we re-implemented part of PVM's communication subsystem using a low-level LLA HIPPI API. The low-level LLA HIPPI API is discussed in Section 2.4.2.1. In Section 2.4.2.2, we present the re-implementation.

2.4.2.1 The LLA Application Programming Interface

The Link Level Access (LLA) application programming interface is a low-level communication interface provided in Hewlett Packard's workstation platform. The LLA interface allows application to encapsulate data into 802.2 frames. LLA uses standard HP-UX file system calls, such as *open()*, *close()*, *read()*, *write()*, *select()*, and *ioctl()*, to access the device drivers that control the network interface card. To communicate with remote processes through LLA interface, the following information must be provided by the sending process:

- SSAP: Source Service Access Point.
- Local Address: The MAC (Medium Access Control) address of the sending host.
- DSAP: Destination Service Access Point.
- Destination Address: The MAC address of the receiving host.

These four tuples (*Local Address, SSAP, Destination Address, DSAP*) are used in a similar way as TCP or UDP connections. In BSD socket interface, each TCP or UDP connection is identified by {*source IP address, source port number, destination IP address, destination port number*}.

Table 2.4: LLA commands used in the re-implementation of PVM.

Command	Description
LOG_SSAP	modify the 802.2 SSAP field (integer value, 0-255).
LOG_READ_CACHE	increase the receive caching to 16 packets for normal users, and up to 64 packets for the super-users.
RX_FLOW_CONTROL	sets the inbound flow control of the current LLA session.
LOG_DSAP	modify the 802.2 DSAP field (integer value, 0-255).
LOG_DEST_ADDR	specifies the destination MAC address.
LOCAL_ADDRESS	get the local station MAC address.

Two types of LLA commands are used for accessing the network interface: *NETSTAT* (NETwork STATus) and *NETCTRL* (NETwork ConTRoL) commands. NETSTAT commands are used for querying status information of drivers and devices. NETCTRL commands are used to control and set up drivers and devices. Some of the LLA commands used in our re-implementation are listed in Table 2.4.

The following code segment is a simple LLA example to illustrate the usage of LLA interface. The program shows how to open the HIPPI device, set up source and destination Service Access Point (SAP), and specify the destination MAC address. Both NETCTRL and NETSTAT commands can be issued to LLA by the *ioctl* system call. A data structure (arg in the example) was used to specify (1) the NETCTRL or NETSTAT command type; (2) the data type of the argument value, and (3) the argument value.

```
if((s = open("/dev/hippi", O_RDWR)) == -1) {
    perror("mroute() lla\n");
    exit(1);
}
arg.reqtype = LOG_SSAP;
arg.vtype = INTEGERTYPE;
arg.value.i = ssap;
```

```
ioctl(s, NETCTRL, &arg);
arg.reqtype = LOG_DSAP;
arg.vtype = INTEGERTYPE;
arg.value.i = dsap;
ioctl(s, NETCTRL, &arg);
arg.reqtype = LOG_DEST_ADDR;
arg.vtype = 6;
memcpy(arg.value.s, dmac, 6);
ioctl(s, NETCTRL, &arg);
read(s, &rxbuf, PACKET_SIZE);
write(s, &txbuf, PACKET_SIZE);
close(s);
```

2.4.2.2 Enhanced Communications of PVM with LLA API

The LLA provides a generic communication interface for upper layer protocols (in our case, PVM's communication subsystem) to access network devices. The re-implementation of PVM over LLA (called *PVM/LLA*) can be used over Ethernet and HIPPI without any change. Upper layer processes can specify the device name to use any network interface and its device driver. For example, /dev/hippi represents the HIPPI interface card and its device driver, and /dev/lan0 corresponds to the Ethernet interface and driver. However, the LLA interface may have slightly different functionalities which are depend on the device driver of network interfaces. For example, the LLA interface provided by Ethernet doesn't support flow control mechanism, while HIPPI's LLA interface provides in-bound flow control to prevent buffer overflow at the receiving side.

We re-implemented part of the communication subsystem of PVM using the LLA interface as follow:

- The connections between PVM daemons was changed from UDP sockets to LLA interface. The master daemon also uses LLA to exchange messages with the shadow daemon.
- The direct communications between local tasks and remote tasks (Direct Routing mode) is changed from TCP/IP to LLA.

As mentioned in Section 2.2.1.4, the Direct Routing mode employs TCP sockets for reliable communications. To achieve the same reliable communication as TCP sockets, the PVM/LLA relies on the sequence number of messages and the inbound flow control of LLA interface. We also increase the number of the read buffers of each LLA connection for high performance. There two features are specified by using RX_FLOW_CONTROL (inbound flow control) and LOG_READ_CACHE (receiving cache) commands.

2.4.3 Performance Evaluation

We present the performance evaluation of a prototype PVM/LLA implementation in this Section. The experimental environment is first described in Section 2.4.3.1. A preliminary performance evaluation of PVM/LLA over Ethernet is presented in Section 2.4.3.2. The performance data is used as a proof of concept. It demonstrates that PVM/LLA provides better performance even without using any high-speed network. The performance evaluation of PVM/LLA over a HIPPI network is presented in Section 2.4.3.3 followed by a performance tuning in Section 2.4.4.

2.4.3.1 Experimental Environment

The environment we used for PVM/LLA over Ethernet is different from that for PVM/LLA over HIPPI. For Ethernet environment, two HP 9000 Series 735 workstations and their Ethernet interface cards were used for the preliminary performance

measurement. For PVM/LLA over the HIPPI network, we used two HP 9000 Series 735/125 workstations which were connected with point-to-point HIPPI links. Each 735/125 workstation equipped with 125 MHz PA-RISC processor and 80 MBytes memory. These workstations are faster than those used in the preliminary performance tests.

The HIPPI interface card is directly connected to HP's Standard Graphics Connection (SGC) I/O bus as the main I/O sub-system card. The SGC I/O bus is a 32-bit wide I/O bus which was optimized for write operations (graphical display involves lots of write operations). The theoretical throughput of the SGC bus is 60 MB/sec for outbound writing and 38 MB/sec for inbound reading. A recent performance measurement [3] shows that the the LLA interface provided by the HIPPI interface card can achieve up to 55 MB/sec throughput for outbound transmission [46]. However, the LLA interface can only achieve around 24 MB/sec throughput for inbound reception.

2.4.3.2 PVM and PVM/LLA on Ethernet

The preliminary tests of PVM/LLA were conducted on HP's Ethernet driver which also support LLA application programming interface. The results of PVM/LLA over Ethernet demonstrate that our re-implemented PVM/LLA can achieve better performance even in a traditional low-speed network. Figure 2.8 shows the user-to-user (between two PVM tasks) round-trip latency of original PVM's (Version 3.3.4) Normal Routing mode and Direct Routing mode, and PVM/LLA's Normal Routing mode and Direct Routing mode. Figure 2.9 shows the achievable user-to-user throughput of these four communication modes.

As shown in Figure 2.8, the round-trip latency of PVM Normal mode is twofold

[3] The measurement was done with a HIPPI analyzer and the *netperf* benchmark program.

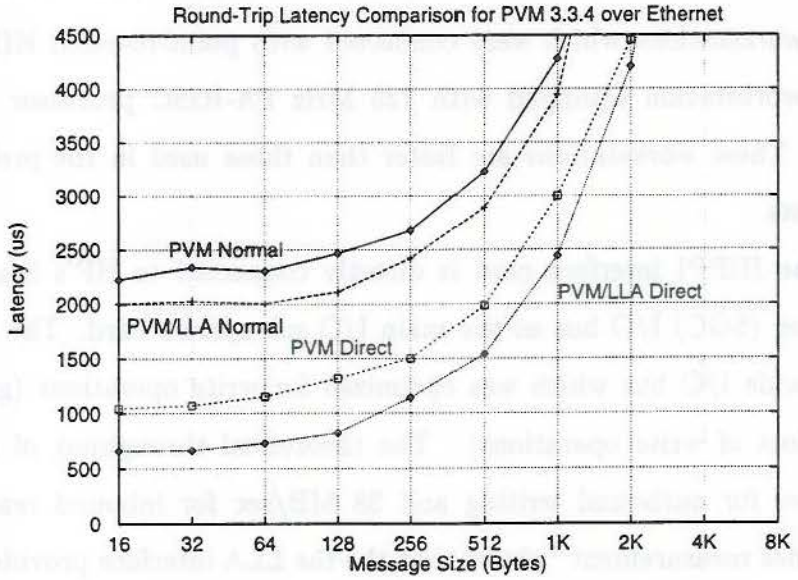

Figure 2.8: Preliminary latency measurement of original PVM and PVM/LLA over an Ethernet network.

more than PVM Direct mode for short messages (less than 512 bytes). For Direct Routing mode, the re-implemented PVM/LLA has up to 38% improvement for the round-trip latency. For the achievable throughput, the 10 Mbits/sec Ethernet does not have much space for PVM/LLA to demonstrate the improvement. Figure 2.9 illustrates that PVM/LLA improves the achievable throughput for PVM Direct mode. Table 2.5 summarizes the performance of PVM and PVM/LLA over 10 Mbits/sec Ethernet network with the three performance metrics as Section 2.3.5.3:

Table 2.5: User-to-User performance of PVM and PVM/LLA over Ethernet.

PVM Environment	Normal Routing mode			Direct Routing mode		
	r_{max} Mbits/sec	$n_{1/2}$ Bytes	t_0 μsec	r_{max} Mbits/sec	$n_{1/2}$ Bytes	t_0 μsec
PVM 3.3.4	6.514	797	2226	8.400	540	1045
PVM/LLA	5.817	541	2004	8.879	382	661

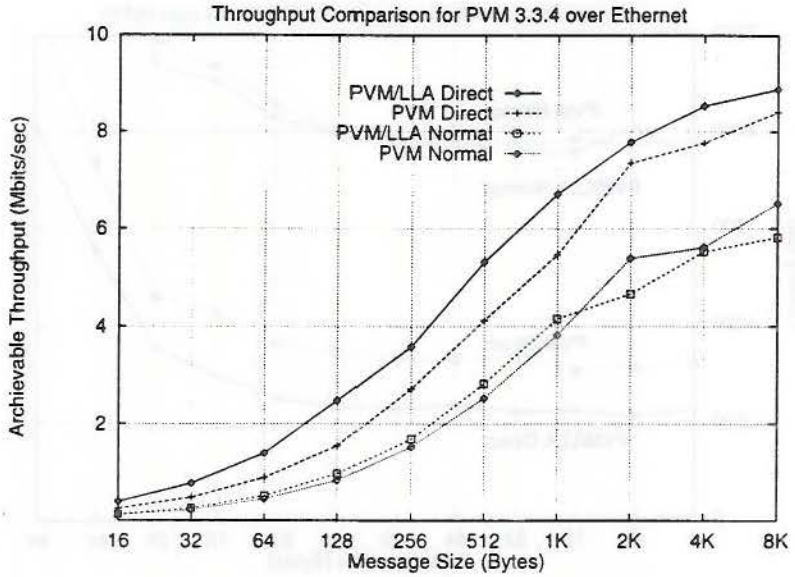

Figure 2.9: Preliminary throughput measurement of original PVM and PVM/LLA over an Ethernet network.

2.4.3.3 PVM and PVM/LLA on HIPPI

The same set of experiments was conducted on two HP 9000 Series 735/125 workstations which were connected with point-to-point HIPPI links. In this section, the re-implemented PVM/LLA uses the LLA interface provided by the HIPPI device driver. Figure 2.10 depicts the round-trip latency measurement of the original PVM and the re-implemented PVM/LLA over the HIPPI network. For the Direct Routing mode, the re-implemented PVM/LLA shows consistent improvement of the round-trip latency. The improvement was reflected in Figure 2.10 for a wide range of message sizes, from 4 bytes to 8 KBytes. For the message sizes shown in Figure 2.10, the re-implemented PVM/LLA achieved up to 33% of latency reduction.

Figure 2.11 depicts the measurement of achievable throughput of the original PVM and the re-implemented PVM/LLA over the HIPPI network. For this test, we did not use the flow control feature provided by the LLA interface. As shown

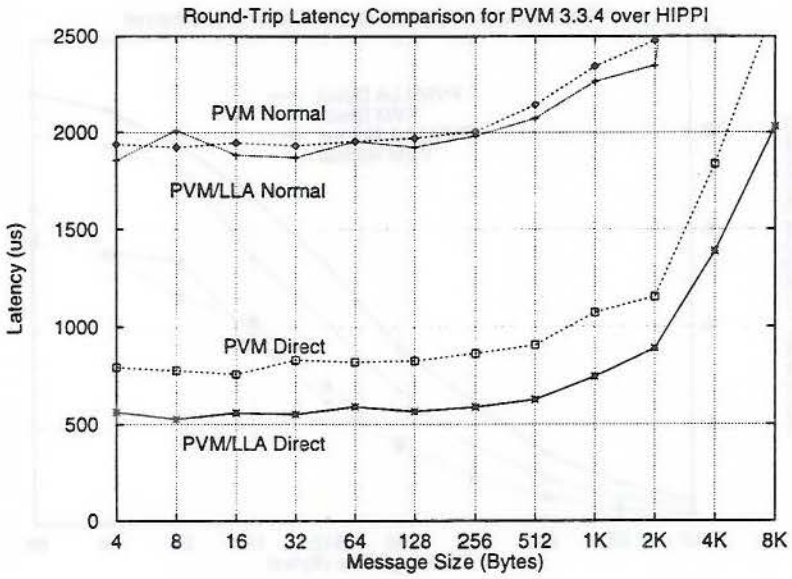

Figure 2.10: Latency measurement of PVM and PVM/LLA on a HIPPI network.

in Figure 2.11, the re-implemented PVM/LLA can achieve higher throughput than original PVM for messages size less than or equal to 256 KBytes. However, the achievable throughput of PVM/LLA reaches the peak with 64 KBytes and can not obtain higher throughput after that. On the other hand, the original PVM's Direct mode reach its peak achievable throughput 9.679 MBytes/sec with message size of 256 KBytes.

Figure 2.11 shows one interesting behavior of the re-implemented PVM/LLA. For message sizes larger than 256 KBytes, the achievable throughput drops dramatically from 11.72 MBytes/sec to 7.35 MBytes/sec. The reason for the performance degradation is due to the throughput mismatch between the speed of HIPPI and the processing speed of a PVM task. The PVM task can not send out or receive messages in a comparable speed as HIPPI. We experienced loss of data with messages larger than 256 KBytes. The data lost problem was also due to receiving buffer overflow when RX_FLOW_CONTROL feature was off. In next Section, we will demonstrate the

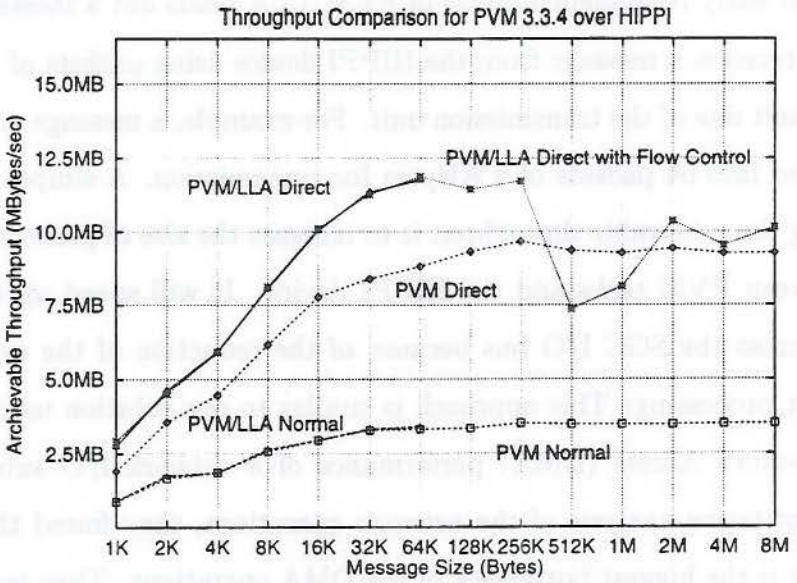

Figure 2.11: Throughput measurement of PVM and PVM/LLA on a HIPPI network.

effect of flow control and the size of message transmission unit on the performance of PVM/LLA.

2.4.4 Performance Tuning of PVM/LLA over HIPPI

The throughput mismatch problem between the HIPPI network and the SGC I/O bus of HP Series 735/125 workstations suggests that SGC I/O bus is the bottleneck. As mentioned before, the SGC I/O bus can sustain 55 MBytes/sec for write operations and only 24 MBytes/sec for read operations, which are much lower than the 100 MBytes/sec bandwidth of HIPPI network (with 32-bit data channel). To improve the performance of PVM/LLA, we should try to retrieve messages across SGC I/O bus as fast as possible. Therefore, the main design principle of our PVM/LLA is to preserve the low user-to-user latency while increasing the achievable throughput for larger messages.

Our early re-implementation of PVM/LLA sends out a message to the HIPPI device or receives a message from the HIPPI device using packets of 4 KBytes, which is the default size of the transmission unit. For example, a message of 256 KBytes will be chopped into 64 packets of 4 KBytes for transmission. A simple optimization for improving the achievable throughput is to increase the size of packets used to transfer data between PVM tasks and the HIPPI device. It will speed up the transmission of data across the SGC I/O bus because of the reduction of the overhead from the per-packet processing. This approach is similar to one solution used to improve the Direct Memory Access (DMA) performance of a network I/O subsystem [40]. In their quantitative analysis of the network operations, they found that the per-page processing is the biggest bottleneck of the DMA operations. They increased the page size for each DMA operation to reduce the total overhead.

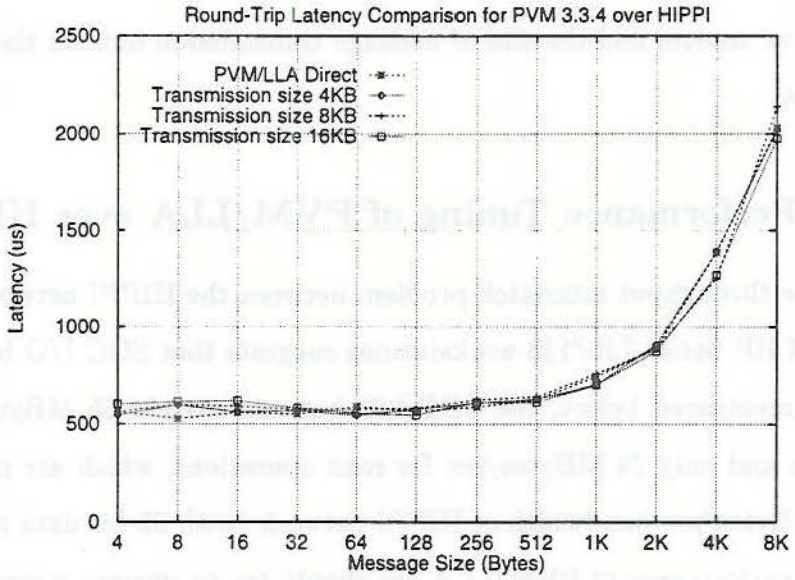

Figure 2.12: Latency measurement of PVM/LLA with different transmission sizes.

To verify that the improvement of throughput does not affect the latency, we tested the PVM/LLA with different transmission sizes. Figure 2.12 shows the

round-trip latency measurement of PVM/LLA over HIPPI network with different transmission sizes, which correspond to the size of packets transferred across SGC I/O bus. As shown in Figure 2.12, the low latency of PVM/LLA was preserved with different transmission sizes. The latencies are very close to each other with message sizes from 4 bytes to 8 KBytes. Figure 2.13 depicts the achievable throughput of PVM/LLA over HIPPI network with different transmission sizes. In this test, we increased the number of read buffers and used the flow control mechanism provided by the HIPPI LLA interface.

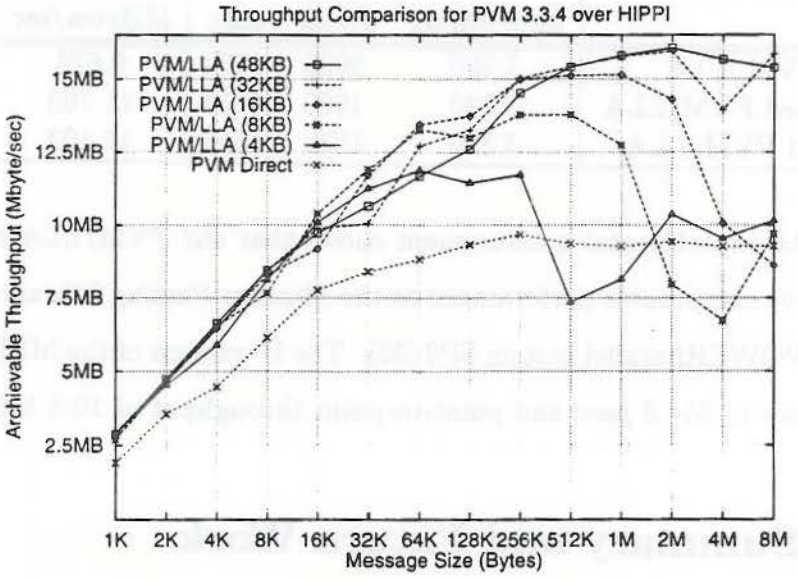

Figure 2.13: Throughput measurement of PVM/LLA with different transmission sizes.

Two interesting observations can be found in Figure 2.13. First, the peak achievable throughput of PVM/LLA was increased with larger transmission sizes. The peak achievable throughput is 11.763 MBytes/sec with transmission size of 4 KBytes. With transmission size of 48 KBytes, PVM/LLA has peak achievable throughput of 16.103 MBytes/sec. Second, the performance degradation problem was alleviated

with larger transmission sizes and the activation of in-bound flow control. There is a small performance degradation when we used 32 KBytes as the transmission size. For PVM/LLA with transmission size of 48 KBytes, there is no performance degradation when the size of messages are less than 4 MBytes. Table 2.6 summarizes the performance of PVM and PVM/LLA over 100 MBytes/sec HIPPI network with the same performance metrics as before.

Table 2.6: End-to-end performance over of PVM and PVM/LLA over HIPPI.

PVM Environment	Normal Routing mode			Direct Routing mode		
	r_{max} MBytes/sec	$n_{1/2}$ Bytes	t_0 μsec	r_{max} MBytes/sec	$n_{1/2}$ Bytes	t_0 μsec
PVM 3.3.4	3.506	3086	1922	9.679	4717	758
un-tuned PVM/LLA	3.390	1989	1855	11.763	4050	528
tuned PVM/LLA	3.390	1989	1855	16.103	7551	540

The experimental measurement shows that our PVM/LLA on a HIPPI LAN can achieve comparable performance as the Message Passing Library (MPL) in IBM's scalable POWERparallel system SP2 [52]. The IP version of the MPL provides round-trip latency of 554.0 μsec and point-to-point throughput of 10.8 MBytes/sec.

2.5 Summary and Future Work

PVM communication is primarily based upon the BSD socket interface. For this study, we chose to bypass the BSD socket interface and implement PVM over lower layer protocols, the Fore Systems' ATM API and HP's LLA interface. We have achieved a performance gain resulting from two factors - utilizing the higher speed network media (ATM and HIPPI) and reducing overhead with lower layer protocols. We observed the following performance results:

- Using the Direct Routing mode of PVM over ATM, we observed greater than

twofold improvement with ATM networks, compared to Ethernet. When the Fore Systems' ATM API was used, instead of the TCP protocol, we observed an improvement of 6 to 7 Mbps. The maximum throughput achieved by PVM-ATM (AAL5) is 27.202 Mbits/sec.

- With the Direct Routing mode of PVM over HIPPI, we observed 66% of throughput improvement (from 9.679 MBytes/sec to 16.103 MBytes/sec) and 30% of reduction for round-trip latency (from 758 μsec to 540 μsec). The improvement of our re-implemented PVM/LLA was restricted by the SGC I/O bus which is used to connect the HIPPI interface card.

One of the drawbacks of using lower layer protocols is that it does not provide support typically found in higher layer protocols. For instance, the Fore Systems' ATM API does not provide flow control and guaranteed delivery. In Section 2.3.4.1, we described an end-to-end flow control mechanism which provides guaranteed delivery by using a selective retransmission mechanism. We also took advantage of the inherent multicasting capability ATM provided, and re-implemented the original PVM multicasting facility. PVM assumes the underlying network cannot perform simultaneous sends from a single source, and thus implemented the multicast operation as a series of sequential sends. In our re-implementation, sending to multiple receivers occurs in parallel. Therefore, with the original PVM multicast, as the number of receivers increases, the latency increases. In our implementation, as the number of receivers increases, the latency remains relatively constant. Figure 2.7 depicts this performance gain.

In this study, we achieved performance improvement at the expense of PVM's heterogeneousness. PVM was originally designed to be used on a network of heterogeneous computer systems. The computer systems may be workstations, multiprocessor computers, or even supercomputers. Our re-implemented PVM over ATM

or PVM/LLA can only be used on a cluster of homogeneous workstations in a LAN environment. Nevertheless, the re-implemented PVM/LLA can potentially be used on a heterogeneous network environment. The LLA interface is a generic communication interface, which provides a common interface for accessing the underlying network devices. A possible extension of this study is to re-implement PVM/LLA such that it can be used on a cluster of homogeneous workstations with heterogeneous high-speed network interfaces, HIPPI, ATM, and Fibre Channel. The PVM tasks can dynamically choose the appropriate network interface based on their communication requirement and utilize the features provided by the underlying network.

An important conclusion from our results is that the performance improvement at the application level is not as good as one may expect from an implementation on a high speed network platform. The maximum achievable bandwidth at the application level, 27.202 Mbits/sec, is far below the "raw" available network bandwidth, 100 Mbps provided by the TAXI interface. Also the measured latency of ATM networks was slightly higher than that of Ethernet. A previous study [40] discussed how the communications overhead has shifted from the network transmission medium to the network subsystem, or I/O subsystem. The network subsystem includes host architecture, software system on the host, and the network interface [40]. To take advantage of a high-speed networking medium such as ATM, the overhead induced by these components and their interactions must be evaluated and reduced.

The computing power of a single workstation and personal computer (PC) is increasing at a very fast pace. How to connect several of them together to form a cluster to perform computing intensive jobs becomes an interesting research topic. The key issue of creating a high-performance cluster of workstations or PCs is to find ways to reduce user level communication latency and to increase user level communication throughput. In this chapter we presented some research results. Our work has been concentrated on standard switch-based high-speed networks like Fibre Channel,

HIPPI and ATM. However, it is possible to use other types of interconnect like SCI (Scalable Coherent Interface [53]), ServerNet [31] and MyriNet. In fact, SCI, ServerNet and MyriNet may provide better performance than HIPPI, Fibre Channel and ATM. However, they are not yet as popular as HIPPI, Fibre Channel and ATM. We are currently investigating these configurations. Another emerging standard which may have profound impact on I/O performance is the "Intelligent I/O" (I_2O). This standard is still under development. It may potentially reduces the communication latency and increases the throughput by using an extra IOP (I/O Processor).

Chapter 3

High-Speed Network Support for Meta-Computing

Among the networks used for distributed computing, the High Performance Parallel Interface (HIPPI) [60, 61, 62, 63] networks are widely used for connections between supercomputers, or between supercomputers and high-end workstations. HIPPI is a simplex point-to-point interface for transferring data at peak data rates of 800 or 1600 Mbits/sec over distances of up to 25 meters. A related standard defines the usage of a cross-point switch to support multiple interconnections between HIPPI interfaces on different hosts [63]. HIPPI is a mature technology, most supercomputers and many high-end workstations are equipped with HIPPI interfaces for high-throughput data connections. Thus, most of the supercomputing institutes use HIPPI as high speed local area networks in their communication infrastructure.

To extend cluster computing to meta-computing, the physical limitation of HIPPI must be solved. The 25-meter limitation of HIPPI restricts the distance from channel endpoint (HIPPI interface on the host) to channel endpoint, channel endpoint to switch port (HIPPI interface on the switch), or switch port to switch port. Extension mechanisms are required to increase the distance between channel or switch connection points. There are three options available for alleviating the problem of distance limitation of HIPPI networks: *Serial-HIPPI*, *HIPPI/SONET or HIPPI-ATM mapping*, and *IP Routing*.

- *Serial-HIPPI*: The serial-HIPPI specifies how the HIPPI packets are to be carried over a pair of fiber optical cables [25]. The HIPPI can be extended up to

10 km on single-mode fiber. This option provides a transparent *extension cord* for HIPPI-PH. However, serial-HIPPI is an implementation agreement, not an ANSI standard project.

- *HIPPI/SONET* or *HIPPI-ATM mapping*: This approach extends HIPPI's connectivity using SONET (Synchronous Optical NETwork), which operating at multiple of OC-1 rates (51.840 Mbits/sec), or ATM (Asynchronous Transfer Mode). Popular data transfer rates are OC-3 (155.520 Mbits/sec) and OC-12 (622.080 Mbits/sec). A HIPPI/SONET mapping scheme over STS-12 SONET was proposed in [27]. In this approach, a HIPPI/SONET link extender is required at each channel endpoint. Each HIPPI burst is encapsulated in one STS-12 frame. For the HIPPI-ATM mapping, the HIPPI-ATM [65] defines the frame formats and protocol definitions for encapsulation of HIPPI-PH packets for transfer over ATM networks.

- *IP Routing*: Using commercial available IP routers, e.g. the GigaRouter from NetStar, Inc. to extend the HIPPI's connectivity [29]. A IP router operates at the network layer, which recovers the data block from one protocol, and maps it into the other protocol, e.g. IP on HIPPI to IP on ATM.

The Serial-HIPPI, HIPPI/SONET mapping and HIPPI-ATM mapping provide extended HIPPI connectivities at the physical layer, while IP Routing forwards data packets between HIPPI networks and other networks at the network layer. With ATM as the de facto standard for wide area network, HIPPI-ATM mapping and IP Routing are two feasible solutions for the internetworking of HIPPI and ATM networks.

In this chapter, we describe a join effort by Computer Science Department, University of Minnesota, Minnesota Supercomputer Center, Inc., and US WEST Communications to interconnect HIPPI networks via private and public ATM networks. The GigaRouters from NetStar, Inc. are used as an IP router and as a

HIPPI-ATM converter. As an IP router, the GigaRouter can route the entire IP packet between HIPPI networks and ATM networks. We called this scheme *IP Routing*. As a HIPPI/ATM converter, the GigaRouter encapsulates the HIPPI bursts in the ATM Adaptation Layer 5's (AAL 5) Packet Data Unit (PDU), then forwards the AAL 5 PDU via ATM networks. At the receiving end, the GigaRouter extracts the HIPPI bursts from AAL 5 PDUs, then forwards the HIPPI bursts via HIPPI networks [65]. We called this scheme *HIPPI Tunneling*.

Two GigaRouters are used to connect the HIPPI network at the University of Minnesota's EE/CS Building and the HIPPI network at the Minnesota Supercomputer Center, Inc. (MSCI). The ATM networks was used as the intermediate media between the two GigaRouters. We investigate the performance issues of HIPPI Tunneling and IP Routing in the same environment. The performance issues we studied include end-to-end latency, user-level achievable throughput, and the protocol behavior of TCP/IP. The effect of TCP's window sizes and the maximum segment size on the end-to-end performance is verified by the result of experimental measurements and detailed timing trace of one ATM analyzer. We hope the practical performance data can provide valuable insight for the vBNS (very high speed Backbone Network Service) project.

The vBNS is funded by the National Science Foundation as the first operation of a very high speed network using the Internet Protocol (IP) over a nationwide ATM and SONET-based network. The vBNS uses the GigaRouter to provide IP routing among a combination of ATM, FDDI, and HIPPI media in a network that transmits high-bandwidth applications at ATM OC-3c speed (OC-12c at 622 Mbps eventually). The vBNS will provide the scientific and engineering community with a new environment for research and will serve as the national test bed for building new applications, increasing telecommunications speeds and developing new advanced national networking technology. The GigaRouters are installed at each of the five

NSF-funded supercomputing centers participating in the launch of the vBNS network.

3.1 Related Work

The first design to extend HIPPI connectivity was the Serial-HIPPI [25]. Serial-HIPPI is an implementor's agreement specifying how the HIPPI packets are to be carried over a pair of optical fiber. Serial-HIPPI was first implemented as HIPPI "modems", which converted HIPPI to serial and back. Further use of Serial-HIPPI can lead to the realization of large-scale HIPPI cross-point switches. Since HIPPI switches are limited in size by the number of parallel paths that must be interconnected and by the number of HIPPI's large-size connectors.

With the potential widespread of SONET connections by public long-haul carrier, SONET provides another mechanism to extend HIPPI's connectivity. Several research groups have proposed or implemented HIPPI/SONET gateways to interconnect HIPPI Local Area Networks (LAN) [27, 38, 66]. Among these approaches, [27] extends HIPPI's connectivity using SONET STS 12c over OC-12, which operating at 622.080 Mb/s with a payload of around 600 Mb/s. In this scheme, each row of the STS-12c frame was used for a HIPPI burst which has a maximum length of 1024 bytes. The key to the performance of this HIPPI extension is a method of relaxing the HIPPI protocol to eliminate the requirement of full round-trip times by the connection and flow control, and adequate data buffering at the gateways.

Los Alamos National Laboratory (LANL) has developed and delivered a HIPPI to multiple OC-3 SONET device that can strip up to eight 155 Mbits/sec connections with forward error correction [38]. Researchers in Bellcore also implemented a HIPPI/ATM/SONET interface that maps HIPPI over multiple ATM and SONET OC-3s streams [66]. Each HIPPI packet is placed in a single ATM virtual circuit, which is then carried over a single SONET OC-3 path. Up to 16 SONET OC-3 paths

can be multiplexed together to form an OC-48 stream when multiple HIPPI packets are being transferred. Both of these schemes utilize the ability to multiplex together multiple OC-3 paths.

The packet delay and loss characteristics of a wide-area HIPPI-based testbed was investigated in [11]. They show that HIPPI locking, receiving side is busy with existing connection, can degrade performance by increasing delay and/or packet loss. Study of the effect of HIPPI blocking on the performance of TCP shows the delay/loss tradeoffs manifests itself in TCP as inducing either the slow-start congestion avoidance algorithm or requiring TCP to adjust retransmission timeout value due to increased delay variance.

3.2 HIPPI Tunneling and IP Routing

An unique feature of our environment is that the same infrastructure can be used for both HIPPI Tunneling and IP Routing. The GigaRouter can act as an HIPPI-ATM converter for HIPPI Tunneling through ATM networks. It also can be used as an IP router which routes IP packet between HIPPI networks and ATM networks. In this section, we describe how the GigaRouter is used in both approaches.

3.2.1 HIPPI Tunneling Through ATM Networks

The *tunneling* mechanism operates at the physical layer, which does not look at the data, adds little latency, and does not require much buffering. In the case of HIPPI tunneling through ATM Networks, the HIPPI-ATM converter directly encapsulates low-level HIPPI-PH packets into AAL 5 PDUs at the sending side. The receiving HIPPI-ATM converter extracts the carried HIPPI-PH packets from the payload of AAL 5 PDUs. Virtual wide-area HIPPI networks can be built by connecting HIPPI

LANs with HIPPI Tunneling through ATM networks.

HIPPI-ATM [65] defines the frame formats and protocol definitions for encapsulation of HIPPI-PH packets for transfer over ATM equipment. A pair of HIPPI converters (the GigaRouters in our case) are used to perform the HIPPI tunneling task. In this scheme, the HIPPI-PH bursts and HIPPI-PH signals are encapsulated in *H-PDU*s, transferred transparently through the intermediate media, and reconstructed as HIPPI-PH signals and bursts. The *H-PDU* consists of a *HB_Header* and the data portion of HIPPI-PH bursts. *HB_Header* is an eight-byte header used to pass control information between HIPPI converters.

An end-to-end connection, shown in Figure 3.1, in the HIPPI-ATM environment is actually composed of three separate connections, two connections between HIPPI-based devices and HIPPI-ATM converters (called *HIPPI connections*) and the connection between two HIPPI converters (called *HIPPI-ATM connection*). Connection control, routing control, and flow control of the *HIPPI connections* shall be as specified by HIPPI-PH. The connection across an ATM intermediate media shall be as specified by HIPPI-ATM.

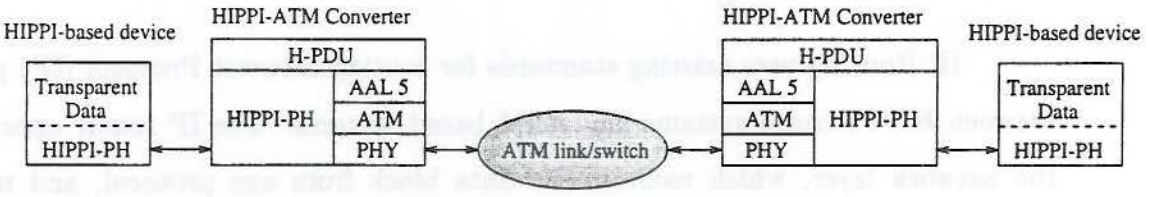

Figure 3.1: Extend HIPPI connectivities with HIPPI-ATM converters.

These connections are separated for performance reasons. The connection across the intermediate media may be independent of the HIPPI equipment making and breaking connections. For example, the ATM connection may last across multiple

packets and for a long time. The HIPPI converters are also assumed to be independent of each other to avoid the latency of intermediate media becoming part of the connection setup time. These separate connections allow a system to send packets in a store-and-forward fashion, with connection breaking on one link while the packet is being forward on the next link.

Figure 3.1 also depicts the protocol hierarchy of HIPPI Tunneling. To transfer data using high-level protocol such as TCP, user's data is prefixed by TCP header, IP header, IEEE 802.2 LLC/SNAP header, HIPPI-LE header, and HIPPI-FP header. Then, the HIPPI-FP packet is transferred via the HIPPI network as a number of HIPPI-PH bursts. The low-level HIPPI-PH bursts are carried by ATM AAL 5 PDUs, which in turn are transferred via ATM layer. The ATM layer provides the functionalities of a network layer protocol, which allow the AAL 5 PDUs being transferred through private and public ATM networks. The receiving HIPPI-ATM converter extracts the HIPPI-PH bursts and forwards them through the HIPPI network. At the receiving HIPPI-based device, user's data is sent to the application after protocol processing operations have been done through the protocol stack.

3.2.2 IP Routing

IP Routing uses existing standards for routing Internet Protocol (IP) packets between HIPPI based systems and ATM based systems. The IP router operates at the network layer, which recovers the data block from one protocol, and maps it into the other protocol, e.g. IP on HIPPI to IP on ATM. Compare to the tunneling mechanism, a router has more intelligence, requires more buffering, and might have longer latency.

The relationship of protocol hierarchy of IP over HIPPI and IP over ATM is depicted in Figure 3.2. On the HIPPI side, the IP PDU is placed in a HIPPI

packet as specified by HIPPI-LE [62] and Internet Request for Comment (RFC) 1374 [49]. HIPPI switches may be between the HIPPI-based device and the HIPPI-ATM IP Router for multiple connections. In the HIPPI portion of the sending-size of the HIPPI-ATM IP Router (the HIPPI-ATM IP Router on the left side of Figure 3.2), the HIPPI headers are discarded, and the IP PDU passed to the ATM side. In the AAL 5 portion of the sending-size of the HIPPI-ATM IP Router, the IP PDU is packaged in an AAL 5 packet. A similar scenario may be used to transfer IP PDUs from the ATM-based device to the HIPPI-based device.

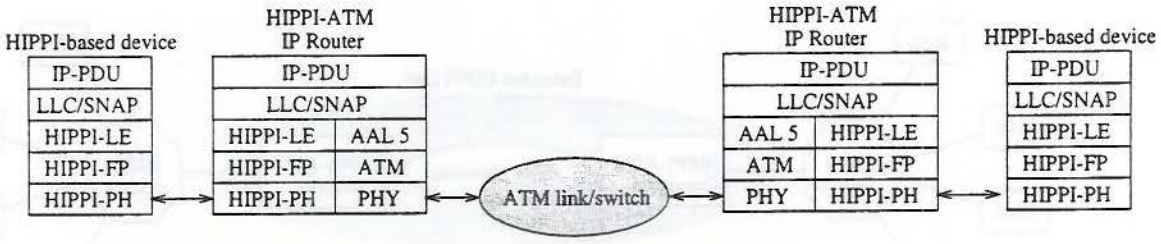

Figure 3.2: Extend HIPPI connectivities with IP Routers.

In the IP Routing scheme, the HIPPI-ATM IP Routers and any other IP routers located between the two HIPPI-ATM IP Routers must wait for the arrival of the entire IP PDU before forwarding it to the next link. The store-and-forward behavior on the IP layer (network layer in the OSI model) introduces longer latency and requires more buffer space than HIPPI Tunneling approach.

3.2.3 Extended HIPPI Connectivities

Both HIPPI Tunneling and IP Routing scheme provides extended HIPPI connectivities as shown in Figure 3.3. The HIPPI Tunneling (upper part of Figure 3.3) supports trunk lines between HIPPI-based LANs. The connection provided by the HIPPI-ATM converter and the intermediate ATM networks acts like an extended

HIPPI link between the HIPPI switches. The HIPPI packets are forwarded by the HIPPI-ATM converters through the trunk link in a multiplexing style. Any HIPPI-based device can setup a connection with the HIPPI-ATM converters (directly or through a HIPPI switch), send HIPPI packets, and tear down connection afterward. The receiving HIPPI-ATM converters connects to a HIPPI-based device according to the H-PDU it received, then forwards the packets. The entire configuration can be treated as one network which consists of two HIPPI switches interconnected with each other.

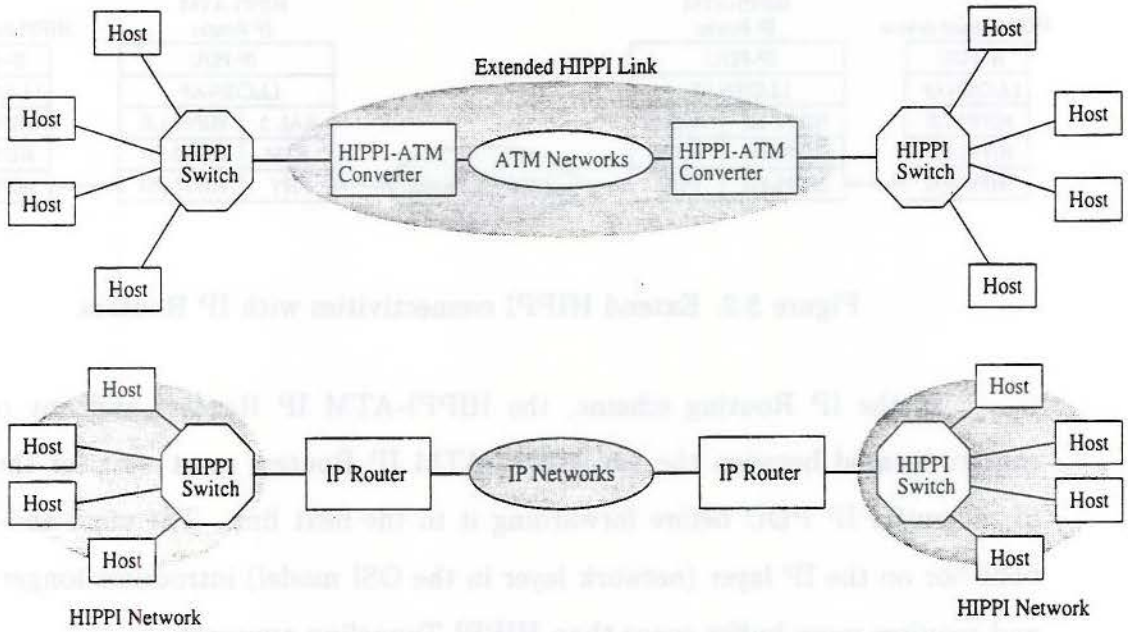

Figure 3.3: Extended HIPPI Connectivities with HIPPI Tunneling or IP Routing.

On the other hand, the IP Routing (lower part of Figure 3.3) connects two separate HIPPI networks via IP networks which can be any kind of media. In our environment, the intermediate media is an ATM-based network. Working on the network layer, the IP router forwards IP PDUs through any IP networks with different medium. However, the high bandwidth of HIPPI restricts the selection of intermediate

media to those high-speed networks like ATM or Fibre Channel.

3.2.4 Protocol Overhead

As a comparison of HIPPI Tunneling and IP Routing at the AAL 5 layer, HIPPI Tunneling does not package the entire IP datagram into the payload of one AAL 5 PDU. The IP datagram is chopped into data packets in the unit of one HIPPI-PH burst or two HIPPI-PH bursts. Whereas with IP Routing, each IP datagram is packaged into the payload of one AAL 5 PDU. Therefore, these two approaches introduce different degrees of protocol overhead.

HIPPI-ATM specifies that the HIPPI converter at the sending-side shall assemble up to 2048 bytes of HIPPI-PH bursts, with an HB_Header (8 bytes), into an H-PDU with size of 2056 bytes. The HIPPI converter can also assemble one HIPPI-PH burst (up to 1024 bytes) and an HB_Header into an H-PDU with size of 1032 bytes. When using ATM as the intermediate media, the ATM AAL 5 shall be used to carry the H-PDUs.

It is easy to find out the protocol overhead and available bandwidth of HIPPI-ATM with the above information. The OC-3c provides 135.632 Mbits/sec bandwidth to the AAL after considering the protocol overhead of SONET and ATM [9]. With 1032-byte H-PDUs, we need to use AAL 5 PDU of 1056 bytes, 22 ATM cells, to encapsulate one H-PDU. Since AAL 5 uses Unused Pad bytes to fill out the last ATM cell to right adjust the AAL 5 Tail. Consider the size of the MTU as 61440 bytes on the HIPPI side. To transfer 61440 bytes of data, we need a small AAL 5 PDU (3 ATM cells) for the protocol headers (from TCP, IP, LLC/SNAP, HIPPI-LE, and HIPPI-FP) and one AAL 5 PDU (22 ATM cells) for each HIPPI-PH burst. The channel utilization is calculated from the following equation.

$$\frac{61440}{(3 + 22 \times 60) \times 48} = 0.9675 \tag{3.1}$$

Therefore, only 96.75% of the 135.632 Mbits/sec bandwidth is used to transfer data via TCP/IP. With 2056-byte H-PDUs, one AAL 5 PDU of 2064 bytes (43 ATM cells) shall be used to carry one H-PDU. This means 98.99% of the 135.632 Mbits/sec bandwidth is used to transfer data via TCP/IP. The channel utilization is calculated from the following equation.

$$\frac{61440}{(3 + 43 \times 30) \times 48} = 0.9899 \tag{3.2}$$

The channel utilization of IP Routing depends on the maximum transmission unit (MTU) size used by the HIPPI-ATM IP Router. Before forwarding to ATM networks, an IP packet is segmented into a number of data packets with the size up to MTU size. For smaller MTU sizes, there is more protocol overhead due to packet headers or packet tailers. Figure 3.4 shows the effect of MTU size on the channel utilization. Figure 3.4 is calculated by considering the protocol overhead of the transmission of 61440 bytes data via TCP/IP. The two lines in Figure 3.4 represent the channel utilization of HIPPI Tunneling with 2056-byte H-PDUs and 1032-byte H-PDUs, respectively. Figure 3.4 suggests that the MTU size used by the HIPPI-ATM IP router must greater than 6000 bytes in order to have better channel utilization than HIPPI Tunneling.

3.2.5 Flow Control

In HIPPI Tunneling, the flow control is treated as three separate entities like the connection control. The credit-based flow control is used for both *HIPPI connections* and *HIPPI-ATM connections*. The credit-based flow control provides positive

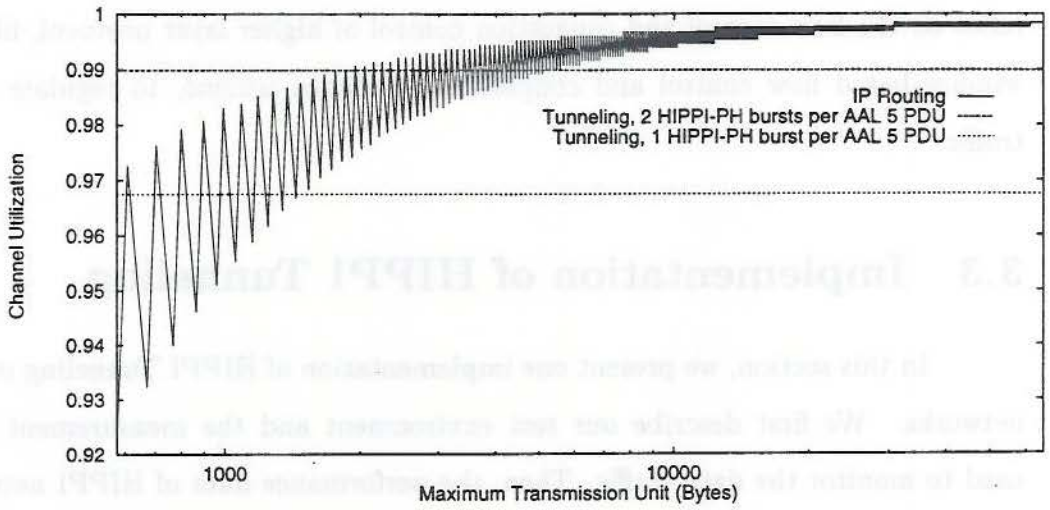

Figure 3.4: Protocol overhead of Tunneling and IP Routing.

flow control to prevent buffer overflow at the receiving-side. The credit-based flow control is accomplished by using HIPPI-PH's *READY* signals on the *HIPPI connection*. Each *READY* signal sent from the receiver represents that the receiver has one available buffer space for one HIPPI-PH burst. The receiving side use the *READY* signals to regulate the behavior of the sending side.

On *HIPPI-ATM connection*, the flow control is accomplished by using the credit information carried in the *HB_Header* as specified in HIPPI-ATM. The credit sent from the receiver to the sender is the number of buffer available for H-PDUs at the receiving-side. As HIPPI-ATM converter at the receiving-side forwards the HIPPI packets in the H-PDU to the destination HIPPI-based device, buffers are freed up. The receiving-side shall periodically inform the sending-side of the number of buffers freed up to avoid sending-side credit starvation. In our environment, the GigaRouter sends credit information every one second if there is no data packet for piggy-backing.

The HIPPI Tunneling provides a low-level flow control scheme. The applications on the HIPPI-based host can use the low-level communication interface to fully

utilize the high bandwidth of the physical link. On the other hand, the IP Routing relies on the flow control and congestion control of higher layer protocol, like TCP's window-based flow control and congestion avoidance scheme, to regulate the data traffic.

3.3 Implementation of HIPPI Tunneling

In this section, we present one implementation of HIPPI Tunneling over ATM networks. We first describe our test environment and the measurement tools we used to monitor the data traffic. Then, the performance data of HIPPI network and HIPPI Tunneling are presented. The performance of the HIPPI network is used as a reference, which represents HIPPI's performance without extended connectivity over ATM networks. In both cases, the performance data suggests that this implementation highly utilizes the network bandwidth.

3.3.1 Environment

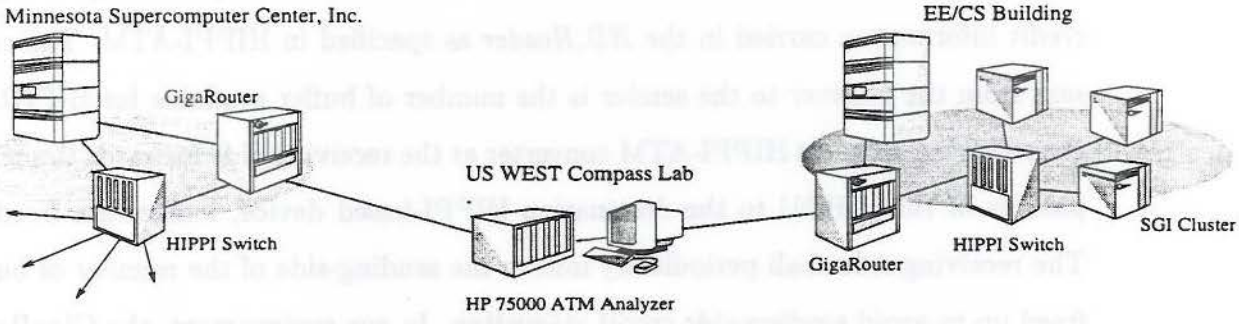

Figure 3.5: Extend HIPPI connections over dedicated OC-3 ATM Network

Figure 3.5 shows the connectivity between the two HIPPI networks via dedicated OC-3 ATM connections. At the EE/CS Building, the GigaRouter is used as

a HIPPI switch and an IP router. An ATM OC-3 connection over single-mode fiber connects the GigaRouter to a Hewlett Packard's 75000 series ATM Analyzer (Hewlett Packard Broadband Series Test System, HP BSTS) at US WEST's COMPASS Lab. The HP BSTS is also connected to the GigaRouter at the MSCI. The GigaRouter at the MSCI can be connected to a HIPPI switch or directly to the SGI Challenge workstation. The SGI Challenge workstation will be the primary computer systems on both HIPPI networks. Therefore, both sides have computer systems with similar performance.

The HP BSTS at US WEST's COMPASS Lab is used to measure the performance at the cell level and the AAL level. The HP BSTS is connected between both GigaRouters. The HP BSTS acts as a SONET repeater which retransmits the received signal without altering the SONET information. Therefore the HP BSTS does not cause any extra delay or introduce any jitter. SONET splitters are installed which will provide a monitor access. This will allow the HP BSTS to be turned off without taking the connection down.

Engineers at the COMPASS Lab are developing decoding software which will allow the HP BSTS to decode the HIPPI-FP/HIPP-LE/802.2 LLC/IP/TCP protocol between both GigaRouters. The software will also provide information in graphical form. For example, it is possible to see the HIPPI Credit and TCP Send/Receive Segment Size each versus time. These tools can be used to better understand the ATM affects on application performance.

With the decoding software, the HP BSTS can be used to monitor data traffic at any protocol layer, from ATM cell level to user-defined high-level protocol. For example, the HIPPI-ATM decoding software can provide the user with a timing diagram of the HIPPI signals (READY, PACKET, etc) which are encapsulated within the HB_Header. The HP BSTS when placed in line can also be used to inject SONET Section, Line, and Path error conditions. The reason for injecting errors is to study

the effects of errors on such items as TCP performance (retransmissions).

3.3.2 Performance of HIPPI-FP and TCP over HIPPI Networks

The performance evaluation of HIPPI-FP (HIPPI Framing Protocol) and TCP between two SGI Challenge workstations via a HIPPI switch is presented in this section. Two SGI Challenge machines in the EE/CS Building are used to perform the test in a HIPPI Local Area Network (LAN) environment.

To study the performance of HIPPI-FP, a pair of echo-style client and server programs using SGI's HIPPI-FP Application Programming Interface (API) [30] is executed on the two SGI Challenge machines. The client measures the latency required to send a message of a certain size to the server and from server back to the client. With the wall clock at the client side, the round-trip latency is used to calculated the end-to-end achievable throughput by dividing two times of the message size by the round-trip latency. The same echo-style client and server programs are also used to investigate the performance of TCP. To achieve the best performance of TCP, we set the size of socket buffer (TCP window size) to 512 KBytes and use the TCP_NODELAY option. For each experiment, we conduct the test for 30 times. The results of the test are used to calculate the mean value, minimum value, maximum value, and the 90% confidence interval. We show the mean value, minimum value, and the 90% confidence interval for the end-to-end latency measurement. For user-level achievable throughput, we present the mean value, minimum value, and the 90% confidence interval.

Figure 3.6 shows the round-trip latency of short messages, from 32 bytes to 1024 bytes. The round-trip latency is around 2.5 milliseconds for HIPPI-FP and around 2.6 milliseconds for TCP in this range of message sizes, which is reasonable

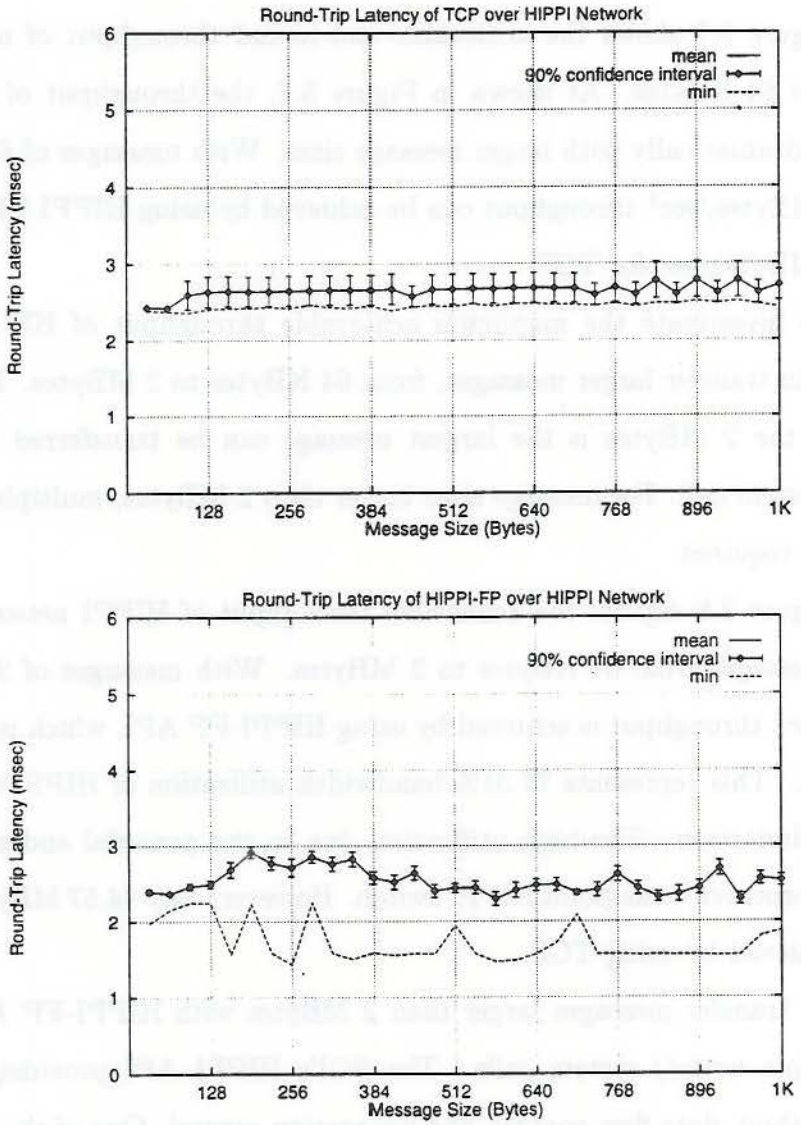

Figure 3.6: Round-Trip Latency of transferring short messages over HIPPI network, Bottom: HIPPI-FP; Top: TCP.

because each HIPPI-PH burst can transfer up to 1 KBytes. Only one HIPPI-PH burst is required in this range of message sizes. Figure 3.6 also depict the 90% confidence interval as error bars.

Figure 3.7 shows the achievable end-to-end throughput of messages from 1 KBytes to 64 KBytes. As shown in Figure 3.7, the throughput of HIPPI network increases dramatically with larger message sizes. With messages of 64 KBytes, more than 20 MBytes/sec[1] throughput can be achieved by using HIPPI-FP API and more than 17 MBytes/sec for TCP.

To investigate the maximum achievable throughput of HIPPI network, we continue to transfer larger messages, from 64 KBytes to 2 MBytes. In SGI's HIPPI-FP API, the 2 MBytes is the largest message can be transferred with one single *write()* system call. For message sizes larger than 2 MBytes, multiple *write()* system called are required.

Figure 3.8 depicts the achievable throughput of HIPPI network when transferring messages from 64 KBytes to 2 MBytes. With messages of 2 MBytes, 73.73 MBytes/sec throughput is achieved by using HIPPI-FP API, which is equal to 618.47 Mbits/sec. This represents 77.31% bandwidth utilization of HIPPI's 800 Mbits/sec physical limitation. The high utilization due to the powerful end-systems and the high-performance cross-point HIPPI switch. However, only 44.57 MBytes/sec throughput is achieved by using TCP.

To transfer messages larger than 2 MBytes with HIPPI-FP API, we need to use multiple *write()* system calls. The SGI's HIPPI API provides commands for access method, data flow control, and connection control. One of the API commands allows the user to keep the HIPPI connection while transferring large amount of data. During the transmission, the HIPPI connection will be maintain, which allow large

[1]MBytes/sec refers to 2^{20} bytes per second.

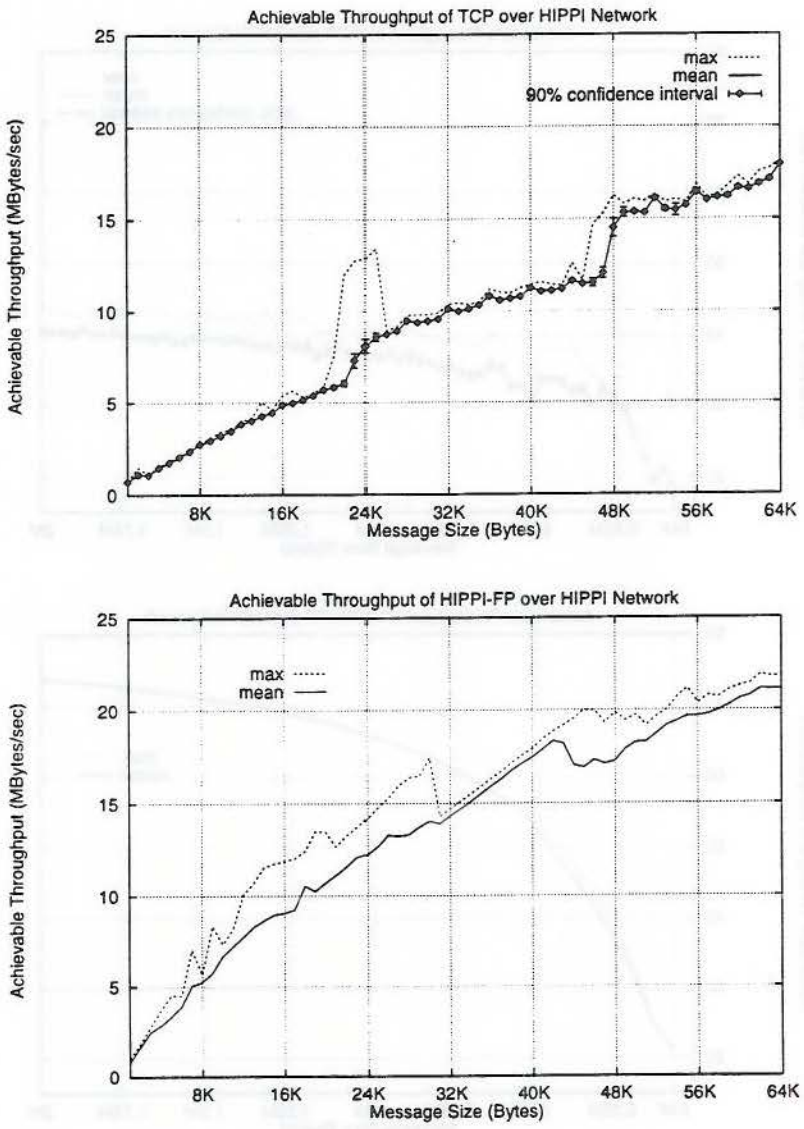

Figure 3.7: Achievable throughput of HIPPI network when transferring messages from 1 KBytes to 64 KBytes, Bottom: HIPPI-FP; Top: TCP. Note: MBytes/sec is 2^{20} Bytes per second.

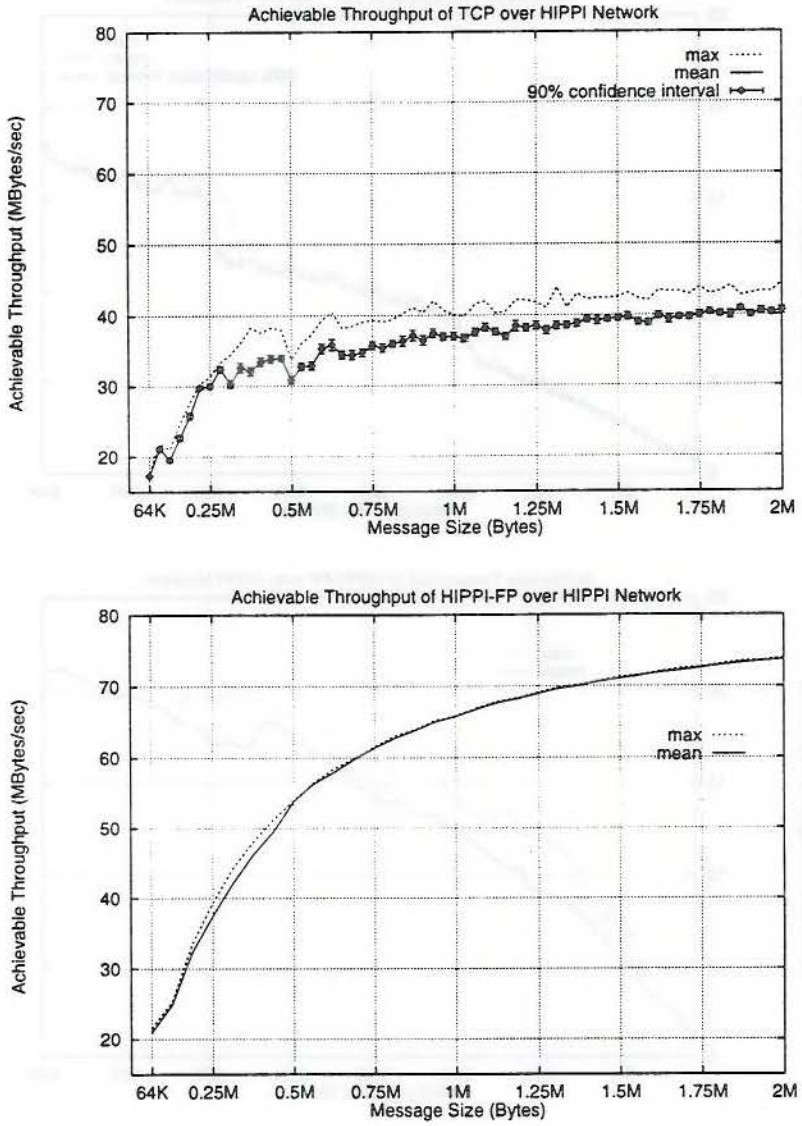

Figure 3.8: Achievable throughput of HIPPI network when transferring messages from 64 KBytes to 2 MBytes, Bottom: HIPPI-FP; Top: TCP.

volume data transmission as seen in many scientific visualization applications.

The end-to-end achievable throughput of large volume data transmission is shown in Figure 3.9. There is only a minor improvement when transferring message sizes larger than 2 MBytes. Part of the reason is because of the overhead from the end-system processing. In the case of HIPPI-FP, multiple *write()* system calls are required, which increase the end-system overhead. With message of 16 MBytes, 76.81 MBytes/sec throughput is achieved, which is equal to 644.363 Mbits/sec. This represents 80.05% bandwidth utilization of HIPPI's 800 Mbits/sec physical limitation. Again, only 49.06 MBytes/sec throughput is achieved by using TCP. The overhead is due to the protocol processing on the host system.

3.3.2.1 The Effect of TCP Window Size

There are several parameters that affect TCP's performance, such as sending and receiving window size, **TCP_NODELAY** option, and TCP's maximum segment size (MSS). In a communication path with large *bandwidth* × *delay* product such as the dedicated OC-3 link in our environment, the TCP performance depends on the product of the transfer rate and the round-trip delay. The *bandwidth* × *delay* product is the amount of data that would occupy the communication link. TCP uses sliding window flow control to regulate how many un-acknowledged data can be transferred. In order to fully utilize the communication link with large *bandwidth* × *delay*, larger window sizes are required to keep the link full.

The RFC-1323 presents a set of TCP extensions to improve performance over transmission paths of large *bandwidth* × *delay* product and to provide reliable operation over high-speed paths [32]. A *TCP Window Scale* option was designed to expand the size of the TCP window, which uses a scale factor to carry the actual window size in the 16-bit *Window* field of the TCP header. The SGI's TCP implementation (in

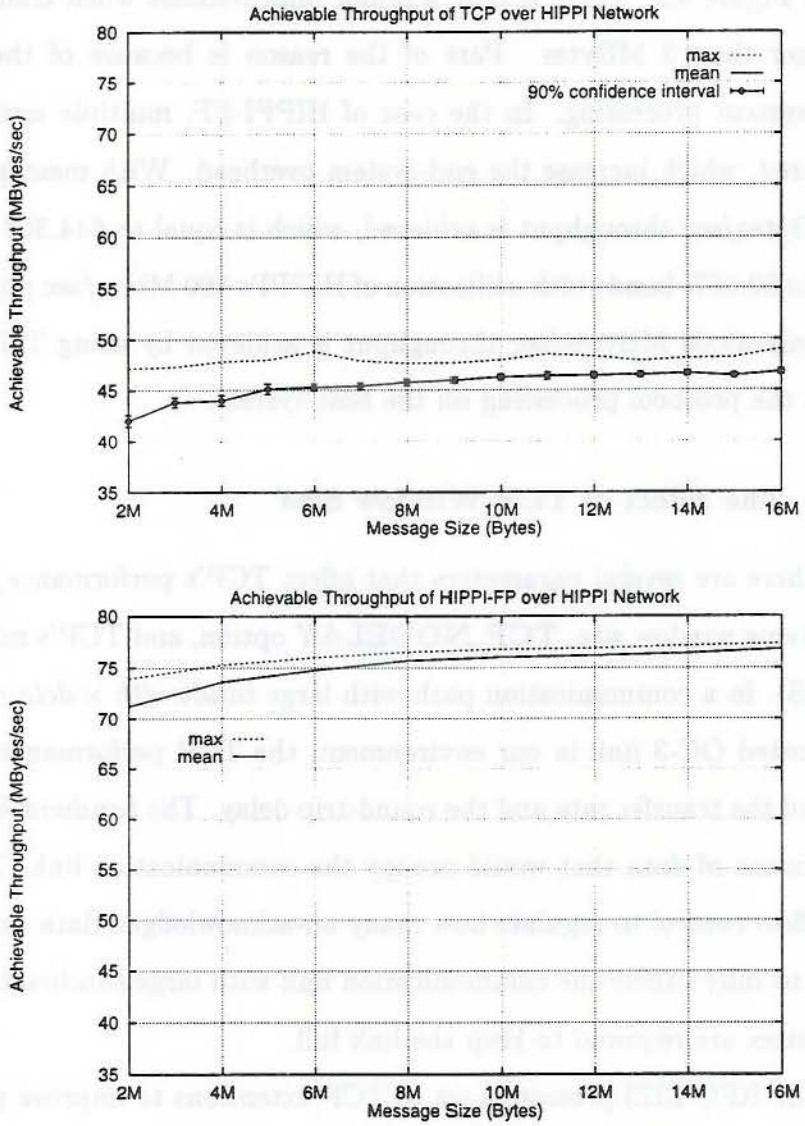

Figure 3.9: Achievable throughput of HIPPI network with large volume of data, Bottom: HIPPI-FP; Top: TCP.

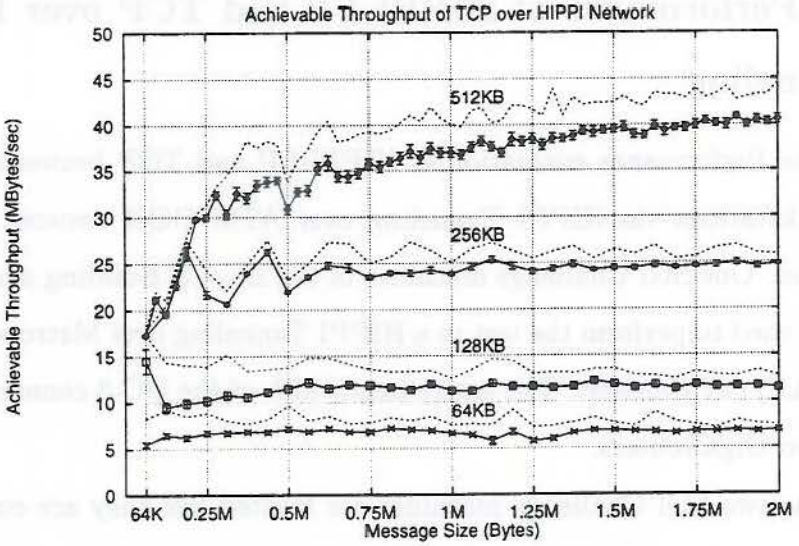

Figure 3.10: The effect of the window size on the TCP performance over the HIPPI network.

the IRIX 5.3 operating system) we used in this study allows the TCP window size to be expended up to 512 KBytes. This means the value of the scale factor can be up to 8.

Figure 3.10 demonstrates the effect of window sizes on the TCP performance over the HIPPI network. From previous measures, the round-trip delay of our HIPPI network at the TCP level is around 2.6 milliseconds. The *bandwidth* $\times$ *delay* product will be 260 KBytes, which means the TCP window size should be larger than 260 KBytes in order to keep the link full. The performance results in Figure 3.10 reflect the effect of window sizes on the performance. For window sizes of 256 KBytes, 128 KBytes, and 64 KBytes, the achievable throughput is bounded by 27 MBytes/sec, 14 MBytes/sec, and 7.5 MBytes/sec.

3.3.3 Performance of HIPPI-FP and TCP over HIPPI Tunneling

The Performance evaluation of HIPPI-FP and TCP between two SGI Challenge workstations via HIPPI Tunneling over ATM OC-3 network is presented in this section. One SGI Challenge machines in the EE/CS Building and another one at MSCI are used to perform the test in a HIPPI Tunneling over Metropolitan Area Network (MAN) environment. The entire bandwidth of the OC-3 connection is allocated for the two GigaRouters.

The two SGI Challenge machines are treated like they are connected to each other via an end-to-end HIPPI connection. The echo-style client and server programs are executed on the two SGI Challenge machines to measure the round-trip latency and end-to-end achievable throughput. We expect a longer round-trip latency in this environment and the maximum achievable throughput will be bounded by the 155.520 Mbits/sec OC-3 link.

Figure 3.11 shows the round-trip latency of short messages, from 32 bytes to 1024 bytes. The round-trip latency is around 3.25 milliseconds for HIPPI-FP over HIPPI Tunneling and 2.5 to 3 milliseconds for TCP over HIPPI Tunneling in this range of message sizes. For HIPPI-FP, the latency is 0.75 milliseconds on average longer than its counterpart in the HIPPI network environment. For TCP, the latency is less than 0.5 milliseconds longer than its counterpart. The longer latency is due to the overhead of HIPPI-ATM converters and the propagation delay. Figure 3.11 also depict the 90% confidence interval as error bars.

Figure 3.12 shows the achievable end-to-end throughput of messages from 1 KBytes to 64 KBytes. As shown in Figure 3.12, the throughput of HIPPI Tunneling also increases dramatically with larger message size. With messages of 64 KBytes, more than 8.5 MBytes/sec throughput can be achieved by using HIPPI-FP and 7.97

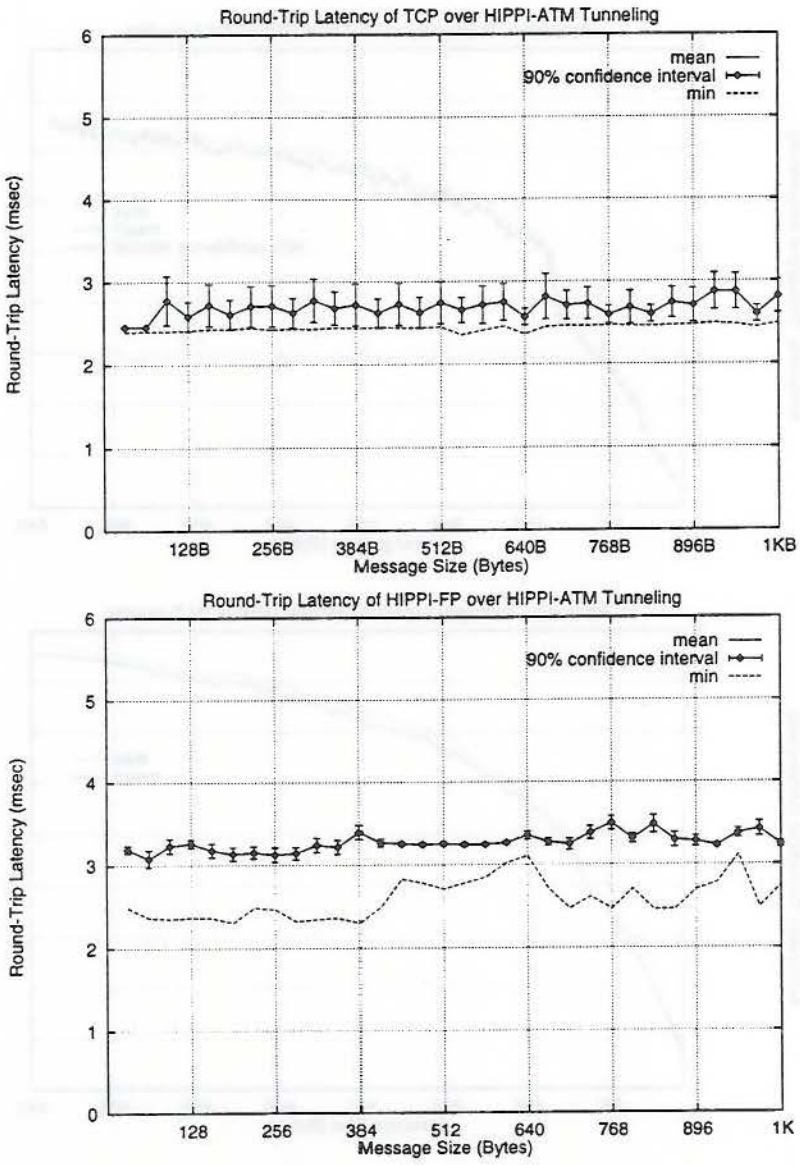

Figure 3.11: Round-Trip Latency of transferring short messages via HIPPI Tunneling over OC-3 ATM link, Bottom: HIPPI-FP; Top: TCP.

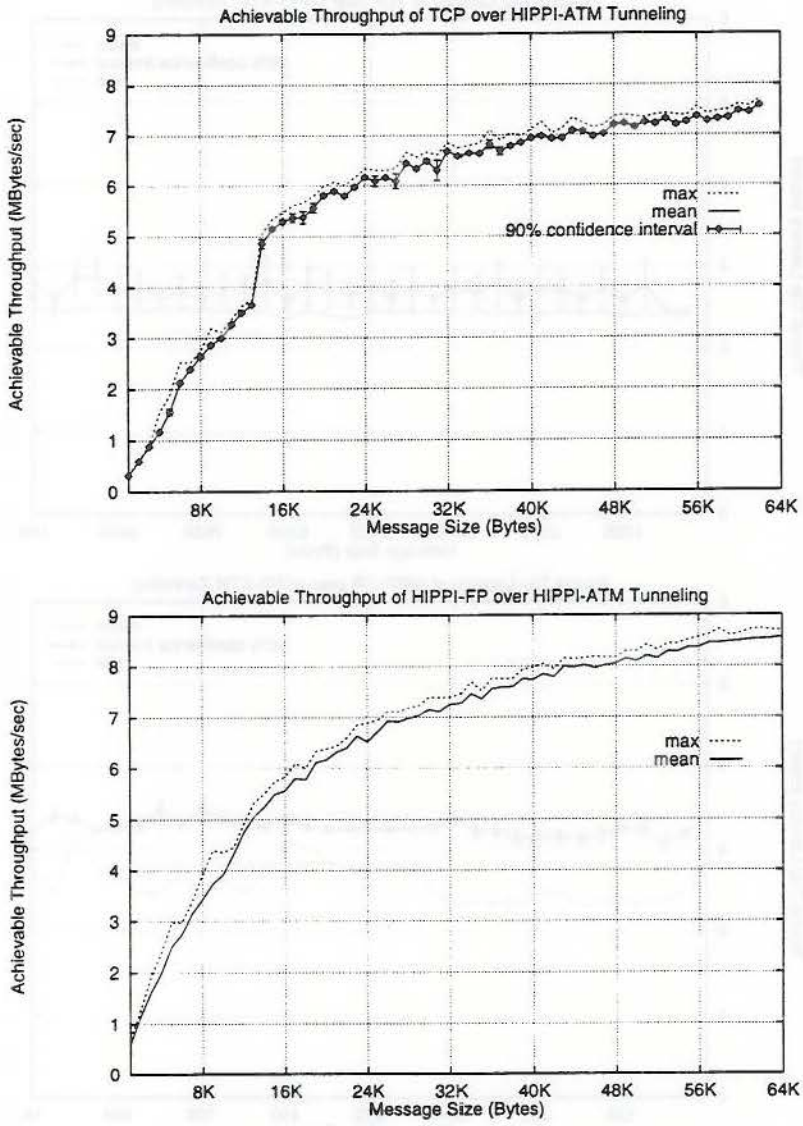

Figure 3.12: Achievable throughput of HIPPI tunneling when transferring messages from 1 KBytes to 64 KBytes, Bottom: HIPPI-FP; Top: TCP.

MBytes/sec for TCP.

To investigate the maximum achievable throughput of HIPPI tunneling, we continue to transfer larger messages, from 64 KBytes to 2 MBytes. Figure 3.13 depicts the achievable throughput of the HIPPI network when transferring messages from 64 KBytes to 2 MBytes. With messages of 2 MBytes, 13.29 MBytes/sec throughput is achieved by using HIPPI-FP and 13.27 MBytes/sec for TCP. For TCP, the throughput is equal to 111.32 Mbits/sec. This represents 82.1% bandwidth utilization of the throughput available to the AAL 5 layer from the OC-3 link (135.632 Mbits/sec). The TCP's performance demonstrates a high degree of channel utilization. As expected, the achievable throughput is bounded by the OC-3 link.

3.3.3.1 The Effect of TCP Window Size

Figure 3.14 demonstrates the effect of window sizes on the TCP performance over the HIPPI Tunneling. From previous measurements, the round-trip delay of the HIPPI Tunneling at the TCP level is around 2.75 milliseconds. The $bandwidth \times delay$ product will be around 53 KBytes, which means the TCP window size should be larger than 53 KBytes in order to keep the link full. However, the performance results in Figure 3.14 show that the communication link can not be fully occupied with window size of 128 KBytes. Figure 3.14 also shows that there is minor difference between window size of 256 KBytes and 512 KBytes. This suggests that the TCP can keep the communication link full when the window size is larger than 256 KBytes. For window sizes of 128 KBytes, and 64 KBytes, the achievable throughput is bounded by 8.35 MBytes/sec and 7 MBytes/sec, respectively.

To further study the effect of window sizes on the TCP performance, we used the HP BSTS to capture the traffic of the OC-3 link. The HP BSTS can capture any ATM cell traverse the OC-3 link and records a time stamp (which is the time

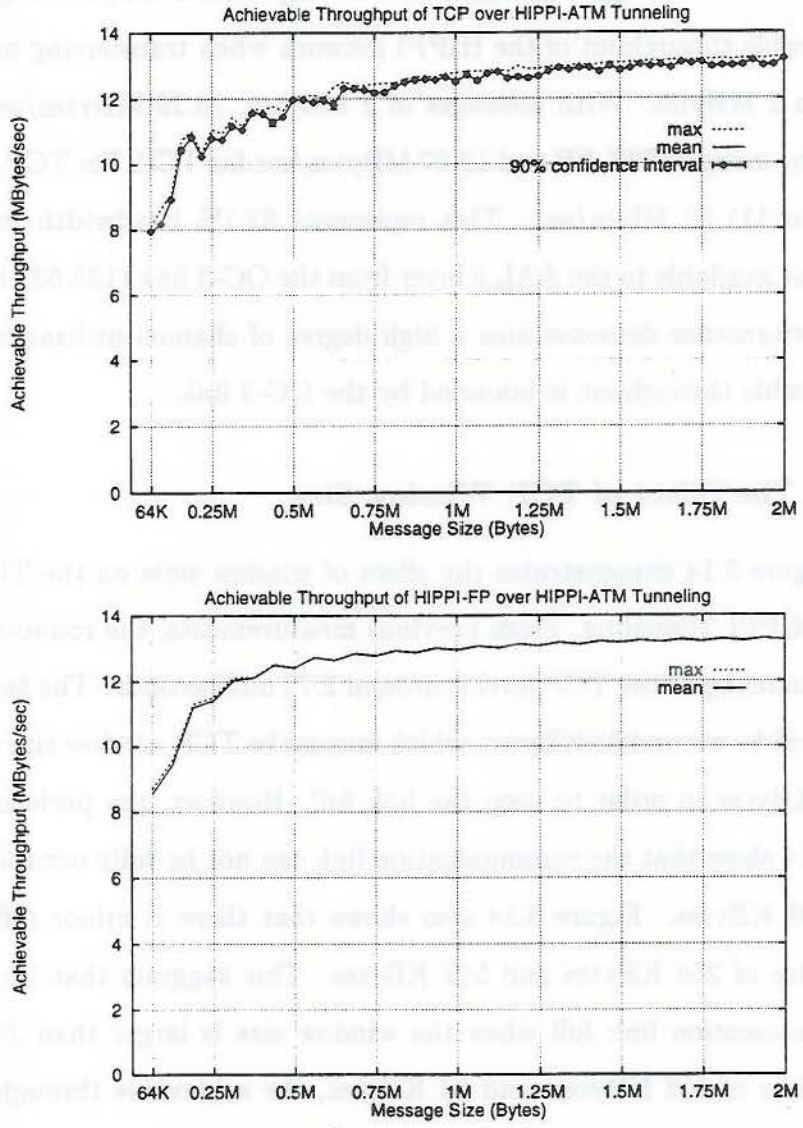

Figure 3.13: Achievable throughput of HIPPI Tunneling when transferring messages from 64 KBytes to 2 MBytes, Bottom: HIPPI-FP; Top: TCP.

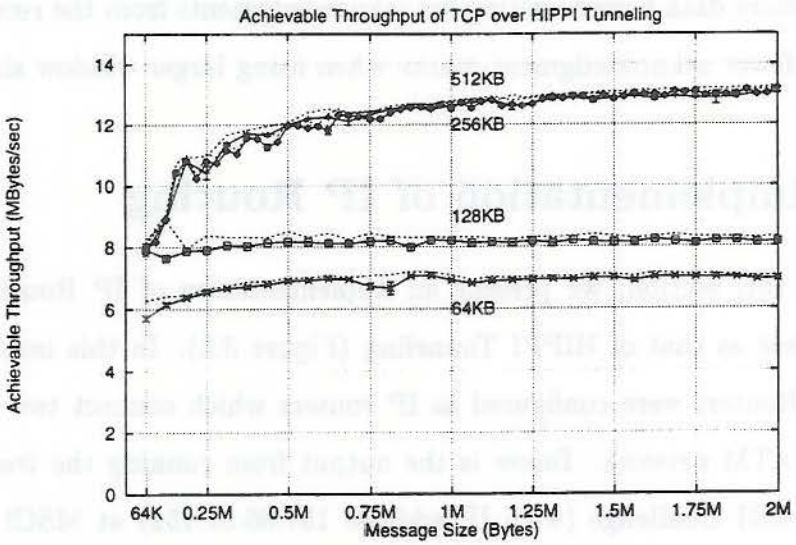

Figure 3.14: The effect of the window size on the TCP performance over HIPPI Tunneling.

when the ATM cell reached the HP BSTS) for each ATM cell. The header of each TCP packet was extracted from the captured data. We used the sequence number and acknowledgment number to study the detailed behavior of the TCP connection. Figure 3.15 is the detailed timing trace of the TCP connection when using TCP window size of 512 KBytes and 64 KBytes. We used the echo-style test program to transfer data of 1 MBytes twice in both cases. Figure 3.15 shows the change of the sequence number and acknowledgment number on the direction from MSCI to EE/CS Building. In both cases, the acknowledge number first increases from 0 to 1048576 (which means the host is receiving data), then stays at 1048576 until the sequence number also increase up to 1048576 (which means the host is sending data). The same pattern repeats twice. Each mark in Figure 3.15 represents one data TCP packet.

It is easy to find out that the sequence number increases faster with larger TCP window sizes (lower part of Figure 3.15). With larger window sizes, the source can

send out more data before waiting for acknowledgments from the receiver. Therefore, there are fewer acknowledgment marks when using larger window sizes.

3.4 Implementation of IP Routing

In this section, we present an implementation of IP Routing on the same environment as that of HIPPI Tunneling (Figure 3.5). In this implementation, the two GigaRouters were configured as IP routers which connect two HIPPI networks with one ATM network. Below is the output from running the *traceroute* program from one SGI Challenge (with IP address 137.66.51.152) at MSCI to another SGI Challenge (with IP address 192.0.10.10) at EE/CS Building:

```
% traceroute 192.0.10.10
traceroute to 192.0.10.10 (192.0.10.10), 30 hops max, 40 byte packets
 1  137.66.51.160 (137.66.51.160)  3 ms  3 ms  3 ms
 2  192.0.2.1 (192.0.2.1)  3 ms  3 ms  2 ms
 3  polar-hip (192.0.10.10)  3 ms  3 ms  6 ms
%
```

The output shows that the two SGI Challenges are three hops away from each other. The connection spreads over three networks.

In our HIPPI networks, the SGI Challenges sent out TCP packets with maximum segment size (MSS) of 61440 bytes (60 KBytes). In our ATM network, the GigaRouters use around 8 KBytes as the size of the maximum transmission unit. Therefore, the GigaRouter need to perform fragmentation and re-assembly operations in addition to routing IP packets between networks.

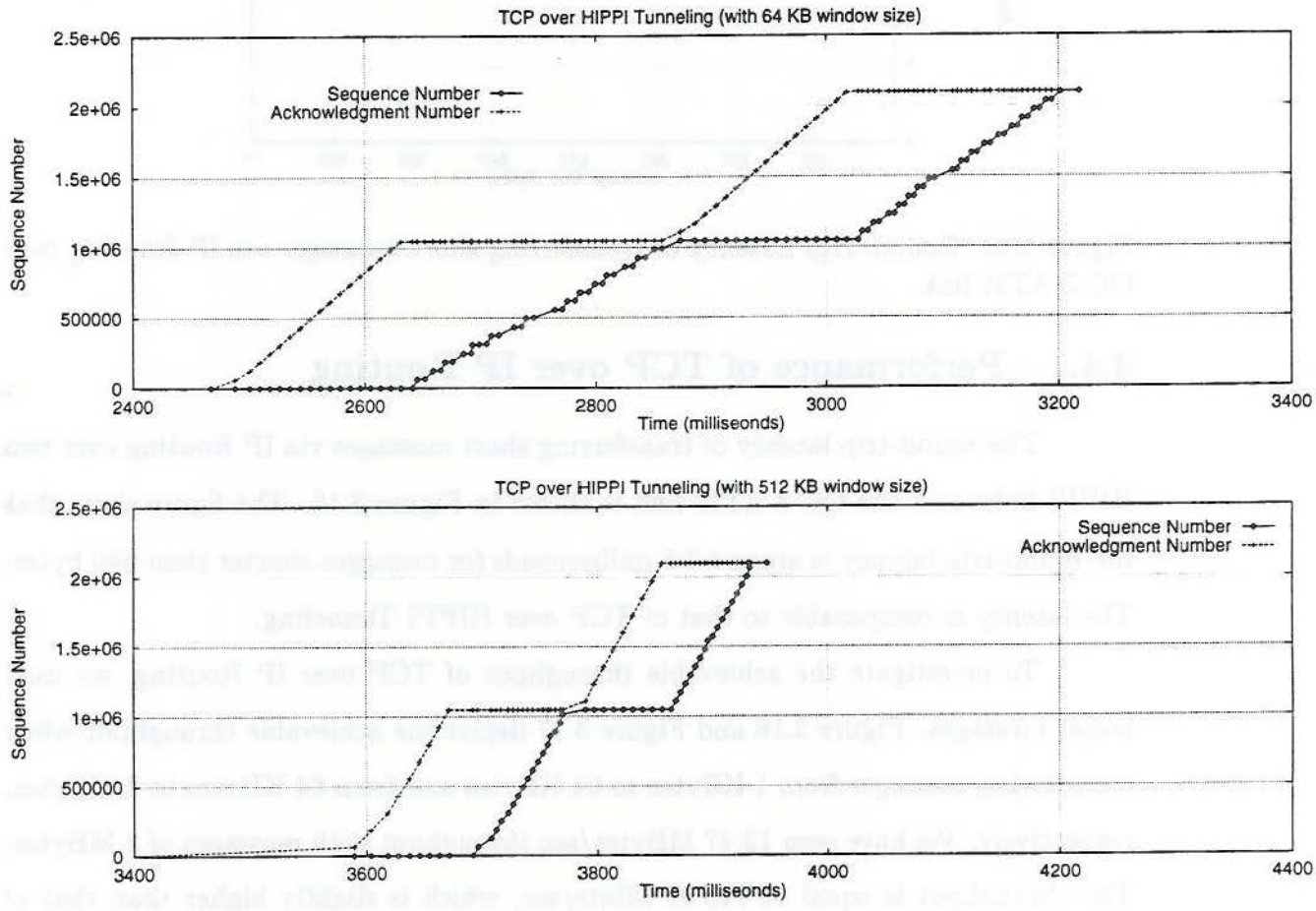

Figure 3.15: Trace of the sequence number and acknowledgment number. Top: using TCP window size of 64 KBytes, Bottom: using TCP window size of 512 KBytes.

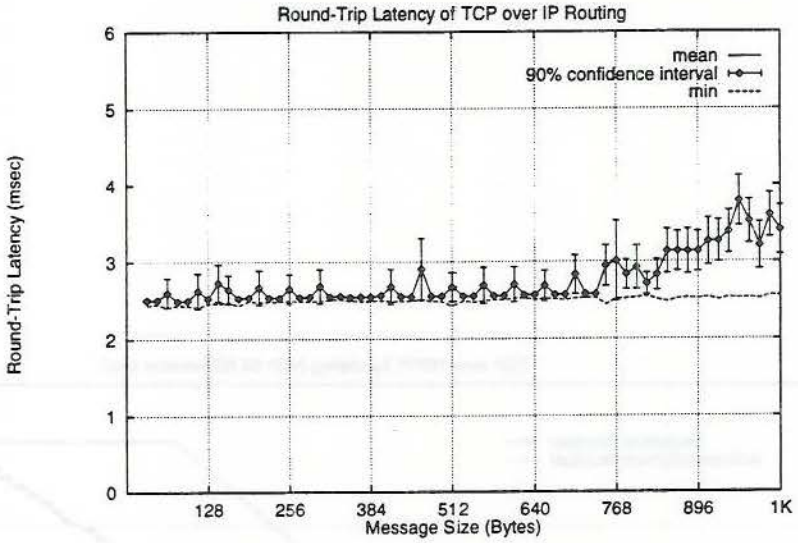

Figure 3.16: Round-Trip Latency of transferring short messages via IP Routing over OC-3 ATM link.

3.4.1 Performance of TCP over IP Routing

The round-trip latency of transferring short messages via IP Routing over two HIPPI links and one OC-3 ATM link is shown in Figure 3.16. The figure show that the round-trip latency is around 2.5 milliseconds for messages shorter than 640 bytes. The latency is comparable to that of TCP over HIPPI Tunneling.

To investigate the achievable throughput of TCP over IP Routing, we used larger messages. Figure 3.16 and Figure 3.17 depict the achievable throughput when transferring messages from 1 KBytes to 64 KBytes and from 64 KBytes to 2 MBytes, respectively. We have seen 13.47 MBytes/sec throughput with messages of 2 MBytes. The throughput is equal to 113.01 Mbits/sec, which is slightly higher than that of transferring messages of 2 MBytes via TCP over HIPPI Tunneling. The throughput was achieved even with the additional overhead on the GigaRouters for the fragmentation and re-assembly operations. The result supports our argument in Section 3.4 that IP Routing could have better channel utilization than HIPPI Tunneling with

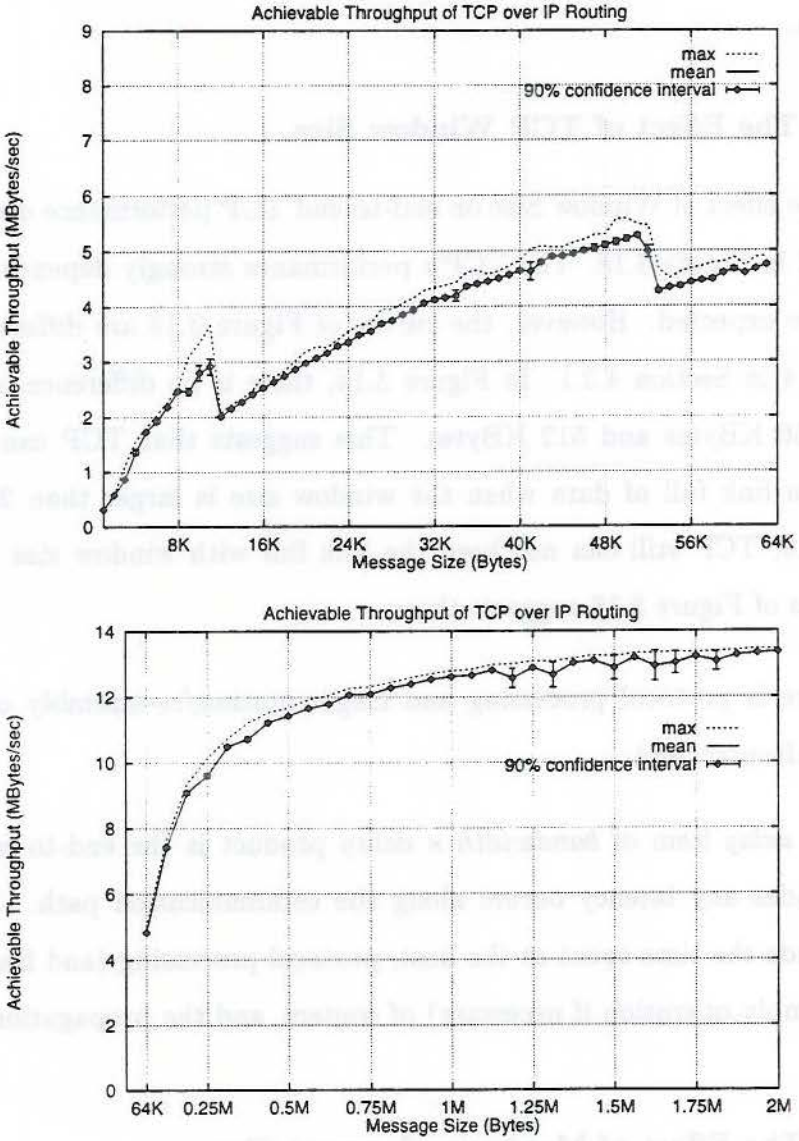

Figure 3.17: Achievable throughput of IP Routing, Top: transferring messages from 1 KBytes to 64 KBytes; Bottom: transferring messages from 64 KBytes to 2 MBytes.

packet size greater than 6000 bytes. In this case, the packet size is around 8 KBytes. The minor difference between the throughput of TCP over IP Routing and TCP over HIPPI Tunneling is because both approaches already reach high degrees of channel utilization.

3.4.1.1 The Effect of TCP Window Size

The effect of Window Size on end-to-end TCP performance over IP Routing is illustrated in Figure 3.18. The TCP's performance strongly depends on the window sizes as we expected. However, the curves of Figure 3.18 are different than that of Figure 3.14 in Section 4.3.1. In Figure 3.14, there is no difference between window sizes of 256 KBytes and 512 KBytes. This suggests that TCP can keep the communication link full of data when the window size is larger than 256 KBytes. In Figure 3.18, TCP still can not keep the link full with window size of 256 KBytes. The curves of Figure 3.18 suggests that:

1. There is protocol processing and fragmentation/re-assembly overhead at the GigaRouter, and

2. The *delay* item of *bandwidth* $\times$ *delay* product is the end-to-end delay which includes any latency occurs along the communication path. These latencies include the time spent at the host, protocol processing (and fragmentation/re-assembly operation if necessary) of routers, and the propagation delay.

3.4.1.2 The Effect of Maximum Segment Size

The performance measurement of the previous section was obtained when there is fragmentation and re-assembly overhead at the GigaRouter. As we mentioned before, the GigaRouter uses around 8 KBytes as the size of the maximum transmission

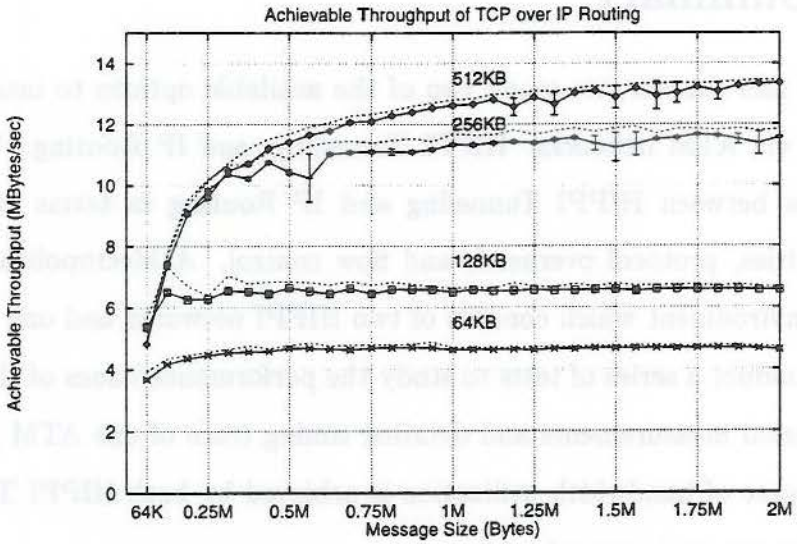

Figure 3.18: The effect of the window size on the TCP performance over IP Routing.

unit. In this section, we reduce the TCP maximum segment size (MSS) from 61440 bytes to 8192 bytes and 4096 bytes to investigate the end-to-end TCP performance without fragmentation and re-assembly overhead. Figure 3.19 shows the effect of MSS sizes on TCP's performance over IP Routing. The achievable throughput of TCP with MSS of 8192 bytes is better than that of TCP with MSS of 4096 bytes. The difference of throughput increases as the message size is getting larger.

The performance data here and the analytical model of protocol overhead in Section 3.4 suggest that larger MSS could have higher degree of channel utilization. However, there will have fragmentation and re-assembly overhead at the router when the MSS of HIPPI networks is larger than the maximum transmission unit of the intermediate network. There is a trade-off between reduction of protocol overhead with larger MSS and avoidance of fragmentation/re-assembly overhead with appropriate MTU.

3.5 Summary

In this chapter, we study two of the available options to interconnect HIPPI networks via ATM networks: HIPPI Tunneling, and IP Routing. We compare the differences between HIPPI Tunneling and IP Routing in terms of their extended connectivities, protocol overhead, and flow control. A Metropolitan Area Network (MAN) environment which consists of two HIPPI networks and one ATM network is used to conduct a series of tests to study the performance issues of these two options. Experimental measurements and detailed timing trace of one ATM analyzer suggest a high degree of bandwidth utilization is achieved by both HIPPI Tunneling and IP Routing in our environment.

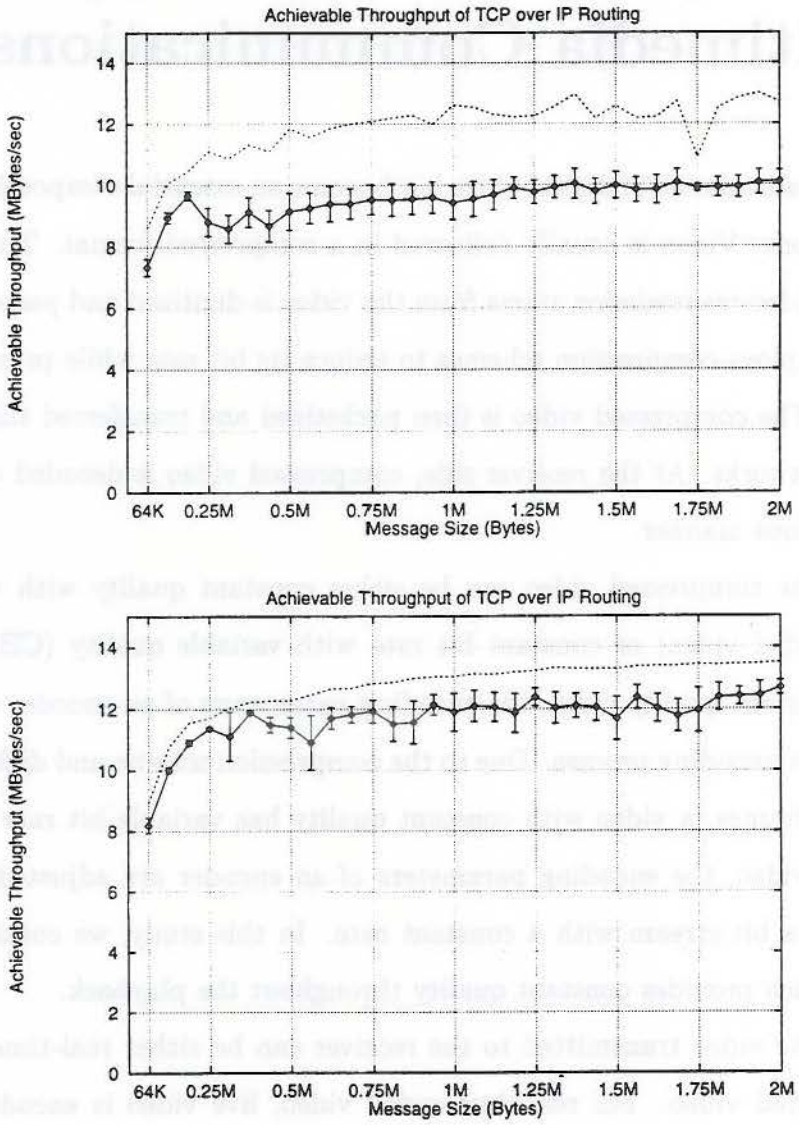

Figure 3.19: The effect of MSS sizes on the TCP performance over IP Routing, Top: MSS size is 4096 bytes; Bottom: MSS size is 8192 bytes.

Chapter 4

High-Speed Network Support for Multimedia Communications

Transmission of digital video has become an essential component of multimedia applications. Video is usually delivered in a compressed format. The entire delivery path of video transmission starts from the video is digitized and passed to an encoder which employs compression schemes to reduce its bit rate while preserving the video quality. The compressed video is then packetized and transferred through communication networks. At the receiver side, compressed video is decoded and displayed in a continuous manner.

The compressed video can be either constant quality with variable bit rate (VBR coded video) or constant bit rate with variable quality (CBR coded video). With constant quality video, the encoding parameters of an encoder remain the same during the encoding process. Due to the compression scheme and different complexity of video frames, a video with constant quality has variable bit rate. With constant bit rate video, the encoding parameters of an encoder are adjusted dynamically to generate a bit stream with a constant rate. In this study, we consider VBR coded video which provides constant quality throughout the playback.

The video transmitted to the receiver can be either real-time coded video or pre-recorded video. For real-time coded video, live video is encoded, transmitted, decoded, and viewed by the user. Example applications are videoconferencing and broadcasting of live programs. Pre-recorded video are digitized and encoded in advance and stored as files in a video server. It provides an on-demand service in which

users can activate video transmission at any time. The transmission of real-time coded video has been studied in [34, 47, 48]. In this study we only consider transmission of pre-recorded video. For a pre-recorded video, its characteristics is completely known a priori. An efficient delivery scheme can be designed to take advantage of this knowledge.

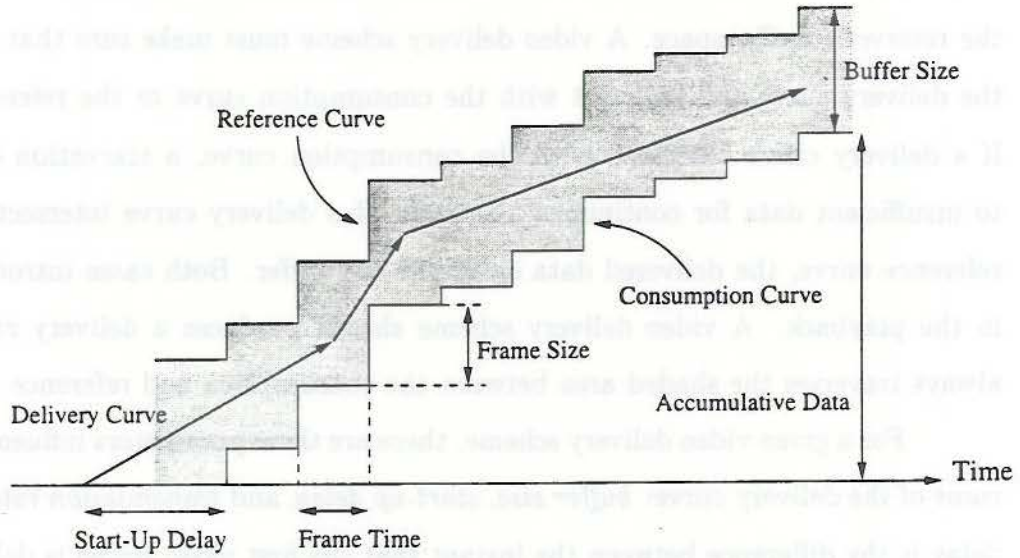

Figure 4.1: A model of video delivery.

A model of video delivery is illustrated in Figure 4.1. The decoder decodes and presents video frames periodically, such as 30 frames per second (fps), to provide continuous playback. The interval for each video frame is called frame time. For 30 fps, the interval is 33.33 ms. The decoder consumes one compressed video frame for each frame time. For VBR coded video, the number of bytes in each frame varies. Assume the decoder consumes one coded video frame instantaneously every frame time, the accumulative amount of data consumed by the decoder can be represented as a *consumption curve* in Figure 4.1. The objective of video delivery is to provide video frames in time for the decoder such that it always has video frames for continuous playback. The accumulative amount of data provided by a video server can be

represented as a *delivery curve*. A possible delivery curve is depicted in Figure 4.1. In this example, the path of the delivery curve always above the consumption curve, which means the coded video frames are delivered in time.

Most of the receivers have limited buffer space to accommodate the difference between the consumption rate and the delivery rate. In Figure 4.1, we also depict a *reference curve* whose vertical distance from the consumption curve is the size of the receiver's buffer space. A video delivery scheme must make sure that at no time the delivery curve will intersect with the consumption curve or the reference curve. If a delivery curve intersects with the consumption curve, a starvation occurs due to insufficient data for continuous playback. If a delivery curve intersects with the reference curve, the delivered data overflows the buffer. Both cases introduce jitters in the playback. A video delivery scheme should produces a delivery curve which always traverses the shaded area between the consumption and reference curve.

For a given video delivery scheme, there are three parameters influence the outcome of the delivery curve: *buffer size*, *start-up delay*, and *transmission rate*. Start-up delay is the difference between the instant that the first video frame is delivered and the instant that it is consumed by the decoder. During this period, a certain amount of data is pre-loaded into the receiver's buffer. The decoder has adequate video frames to start the playback. In Figure 4.1, the transmission rate determines the shape of a delivery curve. The slope of a line segment is the transmission rate used during that period. The relationship among there three parameters is briefly discussed as follow. With a given buffer space, longer start-up delays may allow lower transmission rates to be used in a delivery scheme. With a given start-up delay, higher transmission rates require larger buffer space to accommodate pre-delivered video frames before they can be decoded. Similarly, with a given constant transmission rate, lower start-up delays may reduce the buffer requirement at the receiver.

In this chapter, we study two new CBR transmission schemes, called PCR-assist CBR (PCBR) and PCR-assist Dual-Rate CBR (PDCBR). They utilize the time stamps, called Program Clock References (PCR), inserted by a MPEG-2 encoder to regulate the transmission and to reduce the buffer requirement at the viewer's side. These two schemes keep track of their transmission and dynamically adjust their transmission rates based on the timing information provided by the time stamps. The PCBR scheme was introduced by IBM Rochester [28]. In PCBR scheme, a transmission rate higher than the average rate of a video is used. The time stamps embedded in the video are used to hold up the transmission if it is ahead of the schedule. For PDCBR scheme, two rates are used to adjust its transmission based on the timing information provided by the time stamps. A higher rate is used if the transmission is behind of the schedule, while a lower rate is used if the transmission is ahead of the schedule.

We have developed analytical models of these two schemes. Several video traces are used to compare the transmission rate and buffer requirement of traditional CBR, PCBR, and PDCBR schemes. From the experimental results, PCBR and PDCBR schemes provide more flexible trade-offs between buffer requirement and transmission rate for MPEG streams with high rate variation. The contribution of our study is twofold. First, we have compared the resource requirement (rate and buffer space) of three video transmission schemes. Second, we have studied the relationship among the three parameters (buffer size, transmission rate, and start-up delay) in the two PCR-assist schemes. For example, with a fixed start-up delay, we have studied the minimal transmission rate and buffer requirement. For a given buffer space, we have determined the minimal rate required with an upper bound on the start-up delay. For a given transmission rate, we have studied the minimal start-up delay required with an upper bound on buffer space.

The remainder of this chapter is organized as follows. Section 4.2 discusses

the traditional CBR service which has been studied intensively by researchers. Before describing PCBR and PDCBR schemes, Section 4.3 describes the system model of delivering MPEG-2 over ATM networks. It also covers the PCR-assist mechanism which is used by PCBR and PDCBR to determine the proper transmission rate. We discuss PCBR and PDCBR schemes in detail in Sections 4.4 and 4.5, respectively. The experimental results of these three schemes for several video traces are presented and discussed in Section 4.6. We further compare PCBR and PDCBR schemes in Section 4.7.

4.1 Related Work

Delivering VBR coded video over high speed networks has been studied by a number of researchers [15, 16, 17, 35, 43, 44, 51]. For transmitting pre-recorded VBR coded video (also called stored video) with traditional constant bit rate (CBR) service, McManus and Ross have developed a fundamental relationship between the buffer requirement and transmission rates [44]. They reported that it requires a large buffer at the viewer's side for continuous playback. For example, the minimal buffer required for video *Star Wars* is 23 Mbytes. This makes the pure CBR scheme infeasible under practical consideration since the viewer side may have a decoder with limited memory space.

Other studies have considered piecewise CBR service for the delivery of stored video in order to reduce the buffer or transmission rate requirement [15, 35, 43, 51]. Among them, Salehi et al focus on reduction in variability of transmission rates with a given buffer space [51]. Feng et al consider reducing the number of rate changes during transmission [15]. McManus and Rose determine the transmission schedule which minimizes the buffer and average transmission rate required for a fixed number of transmission intervals [43]. The transmission rate used in each interval

can be different. The schemes proposed in [15, 43, 51] can be implemented on the Renegotiated CBR service proposed by Grossglauser et al [24].

Considering the three parameters mentioned before, the approaches proposed by Salehi et al and Feng et al concentrate on the transmission rate with a given buffer space. The approach proposed by McManus and Rose does not establish a relationship between the buffer and transmission rate requirement.

4.2 Constant Bit Rate (CBR) Transmission

The CBR service is the simplest approach for transmitting video in many applications. In this scheme, a virtual circuit is established with a constant transmission rate. The rate may be equal to or slightly higher than the average rate of a MPEG-2 video stream. If the rate is higher than the average rate, it may requires a large buffer space at the receiver side. Since most MPEG-2 video are VBR coded, at some instant the decoder may consume more data than the server can transmit even with a rate higher than the average rate. In this case, the decoder is starved of video frames during the playback. Therefore, CBR service requires a certain amount of data to be transmitted and stored in the receiver's buffer before the commencement of playback. The fundamental relationship between the start-up delay, transmission rate, and the size of buffer at the receiver side has been studied in [44].

Analytically, the transmission rate can be determined base on the content of a MPEG-2 stream and the length of a start-up delay. For a given start-up delay, the minimum rate required to transmit a MPEG-2 stream can be easily determined using the concept of convex hull as illustrated in Figure 4.2. We assume the network delay is constant and the decoder consumes one coded video frame instantaneously every frame time. The accumulative amount of data consumed by the decoder can be described by a *consumption curve*. It can be represented as a set of points M,

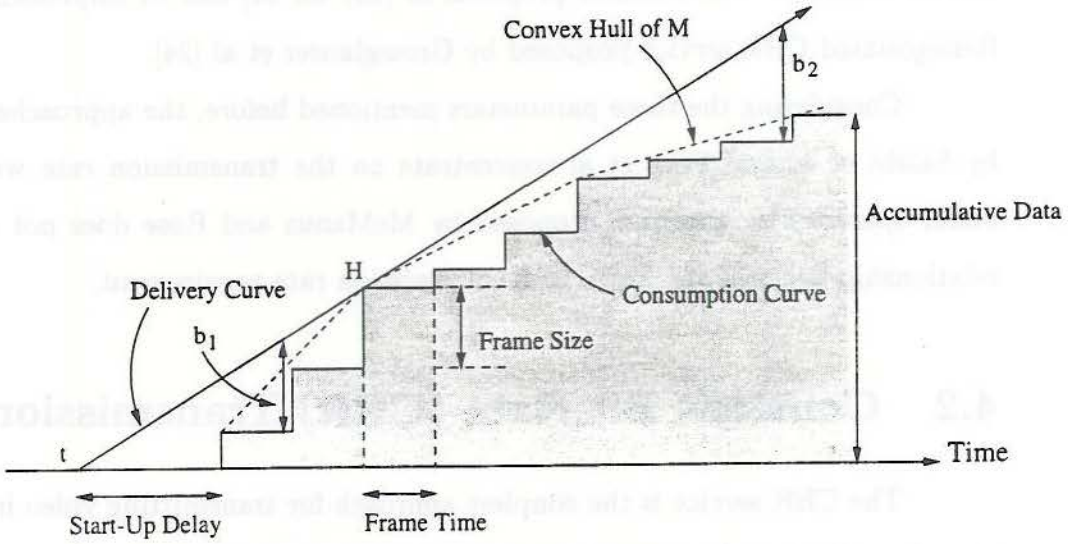

Figure 4.2: The delivery curve of CBR and consumption curve of the decoder.

$$M = \{(n \times F, A_n), ((n+1) \times F, A_n) | 1 \leq n \leq N\}, \tag{4.1}$$

where F is the frame time, N is the total number of frames in a video stream, and A_n is the amount of cumulative data consumed by the client over $[1, n]$ frame time. $A_n = \sum_{i=1}^{n} S_i$ and S_i is the size of frame i. A convex hull, $CH(M)$, of M can be obtained using Jarvis' march algorithm [33]. Figure 4.2 shows a portion of boundary edges of $CH(M)$ which is used to calculate the minimum rate required.

For a given start-up delay P, we can find a line, denoted as the *delivery curve*, connecting point t (whose distance from the first frame time is the start-up delay) and a point of $CH(M)$, say H, with the largest slope. The slope, denoted as R_{CBR}, is the minimum rate required to transmit an MPEG stream without starving the receiver's buffer. The minimum buffer required at the receiver side can then be determined by computing the maximum difference between the *delivery curve* and *consumption curve*. The amount of data that have been received at time $n \times F$ is $(P+n) \times F \times R_{CBR}$.

The amount of data that have been consumed right before time $n \times F$ is A_{n-1}. The maximum difference between the *delivery curve* and *consumption curve* for a given start-up delay P is

$$f_{CBR}(P) = \max_{1 \le n \le N} \left((P+n) \times F \times R_{CBR} - \sum_{i=1}^{n-1} S_i \right). \tag{4.2}$$

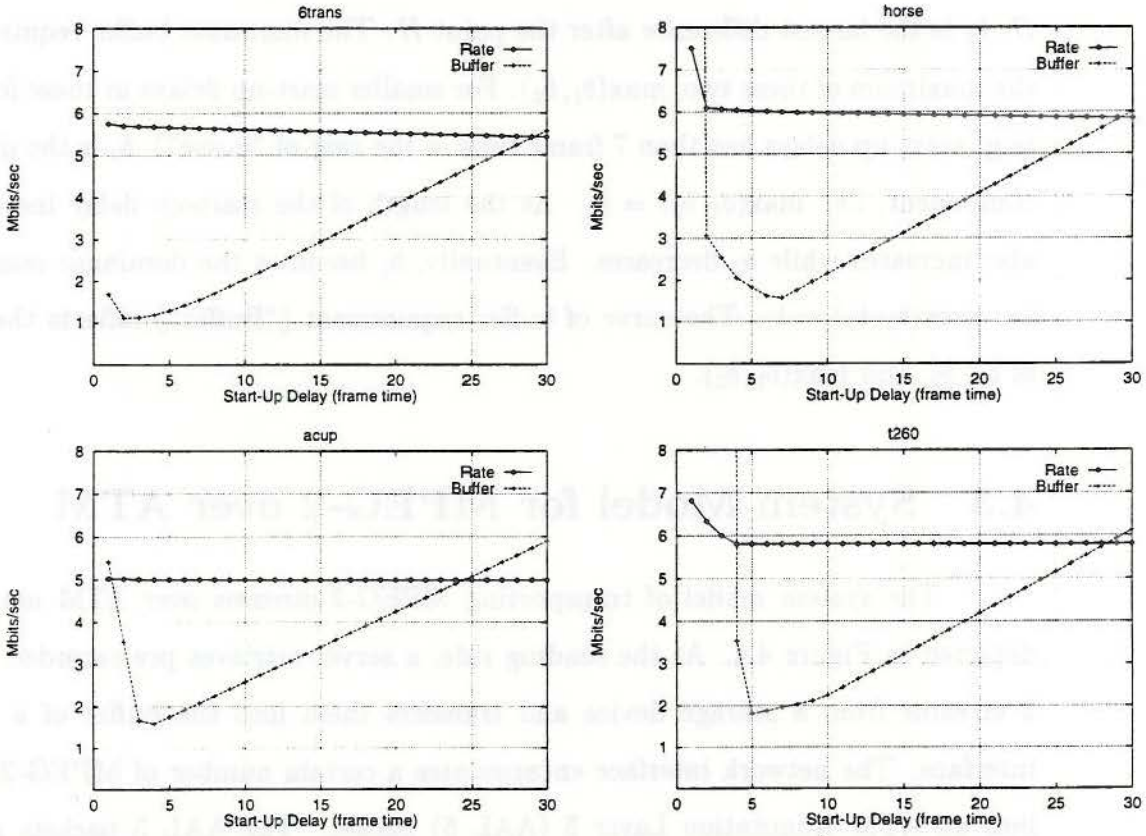

Figure 4.3: Minimum rate and buffer required for different start-up delays. The unit is Mbit/sec for transmission rates, Mbits for buffer sizes.

In Figure 4.3, we used four MPEG-2 traces to study the relationships between transmission rate, buffer requirement, and start-up delay for CBR service. The figure

shows the minimum rate R_{CBR} and buffer $f_{CBR}(P)$ required for different start-up delays: from one frame-time to 30 frame-time. As the length of start-up delay increases, the minimum required rate decreases. This is because the slope from point t to one of the $CH(M)$ points decreases with larger start-up delays. However, the buffer requirement first declines, then increases as the length of start-up delay increases. It is easy to understand this behavior with the illustration of Figure 4.2. Assume b_1 is the largest difference between the delivery curve and consumption curve before the point H, b_2 is the largest difference after the point H. The minimum buffer requirement is the maximum of these two, $\max(b_1, b_2)$. For smaller start-up delays in these four cases (e.g. start-up delays less than 7 frame time in the case of "horse"), b_2 is the dominant component, i.e. $\max(b_1, b_2) = b_2$. As the length of the start-up delay increases, b_1 also increases while b_2 decreases. Eventually, b_1 becomes the dominant component, i.e. $\max(b_1, b_2) = b_1$. The curve of buffer requirement ("Buffer") reflects the change of b_1, b_2, and $\max(b_1, b_2)$.

4.3 System Model for MPEG-2 over ATM

The system model of transporting MPEG-2 streams over ATM networks is depicted in Figure 4.4. At the sending side, a server retrieves pre-recorded MPEG-2 streams from a storage device and transfers them into the buffer of a network interface. The network interface encapsulates a certain number of MPEG-2 packets into an ATM Adaptation Layer 5 (AAL 5) packet. The AAL 5 packets are then segmented into ATM cells and transmitted over a connection to the client. At the client side, the received ATM cells are re-assembled back to AAL 5 packets. MPEG-2 packets are extracted from the payload of AAL 5 packets and sent to a MPEG-2 decoder. In order to accommodate the difference between transmission rate of the server and consumption rate of the client, a buffer is used as a cushion. Usually, the

client can start to decode and playback the MPEG-2 stream right after some data have been received in the buffer. To ensure the continuous playback of a MPEG-2 stream, a certain amount of data will be transmitted and stored in the client's buffer before the commencement of playback. We have defined the time between the start of transmission and the start of playback as *start-up delay*. For the same start-up delay, different transmission schemes may accumulate different amount of data in the buffer.

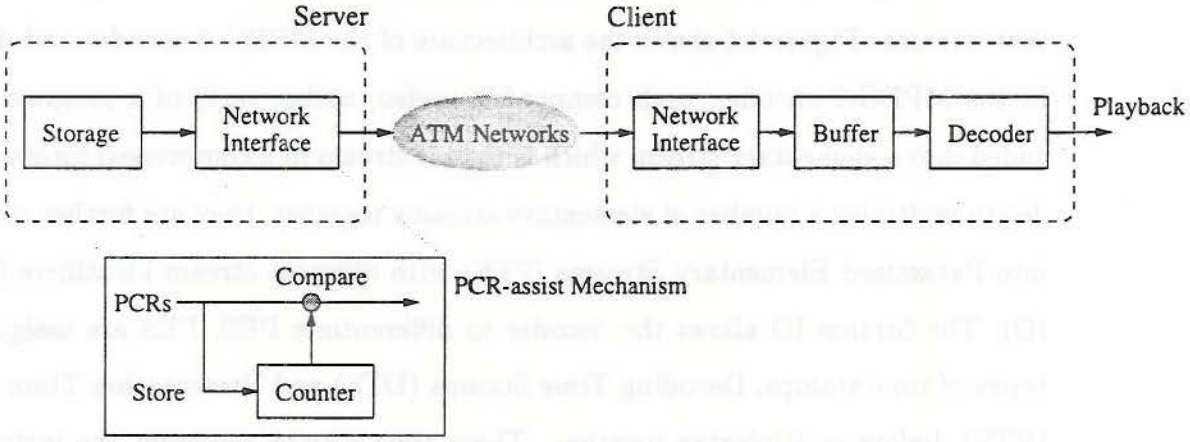

Figure 4.4: The end-to-end model of MPEG-2 over ATM networks.

Different classes of service of ATM can be used to transmit MPEG-2 streams. In this chapter, we focus on CBR, PCR-assist CBR (PCBR) and PCR-assist Dual-Rate CBR (PDCBR). For CBR service, a virtual channel is established between the server and the client with a pre-negotiated constant bandwidth. We assume that MPEG-2 streams are transmitted at a constant cell rate. For PCBR and PDCBR schemes, a special PCR-assist mechanism is used to detect the appearance of PCRs and to control the transmission of ATM cells. We will discuss the mechanism in Section 4.3.3.

4.3.1 MPEG-2 Transport Streams

The Motion Picture Expert Group (MPEG) has standardized compression techniques for video and audio, system streams and transport streams for multiplexing and carrying video, audio, and data in a single time synchronized bit stream. MPEG-2 is a collection of standards which consist of System, Video, Audio, Compliance, and a Digital Storage Media Control Commands (DSM-CC) standard [1, 2, 3]. Among them, the MPEG-2 System standard covers the media multiplexing, media synchronization, and clock synchronization. It organizes the information to be transferred in programs, where a program includes video streams with associated audio and text streams. Figure 4.5 shows the architecture of the MPEG-2 encoder and decoder. In the MPEG-2 encoder, each component (video, audio, text) of a program is first coded into a elementary stream which is the bit stream in a compressed format. In order to multiplex a number of elementary streams together, they are further converted into Packetized Elementary Streams (PES) with different stream identifiers (Stream ID). The Stream ID allows the decoder to differentiate PES. PES are assigned two types of time stamps, Decoding Time Stamps (DTS) and Presentation Time Stamps (PTS), before multiplexing together. These time stamps represent the instant that the associated video frame and audio clip need to be decoded or presented.

Two kinds of System streams are defined by the MPEG-2 standard to meet the needs of various applications. The *Program Stream* is intended for use in a reliable environment such as playback from a local storage device. It consists of large and variable-sized packets. The *Transport Stream* (TS) is intended for use in a lossy environment such as ATM networks. It consists of small fixed-sized packets of 188 bytes, called Transport Packets (TP), to reduce the impact of data loss. For transmission over a network, it is important that the MPEG-2 decoder and the MPEG-2 stream source be synchronized so that the decoder consumes data at the same rate that the

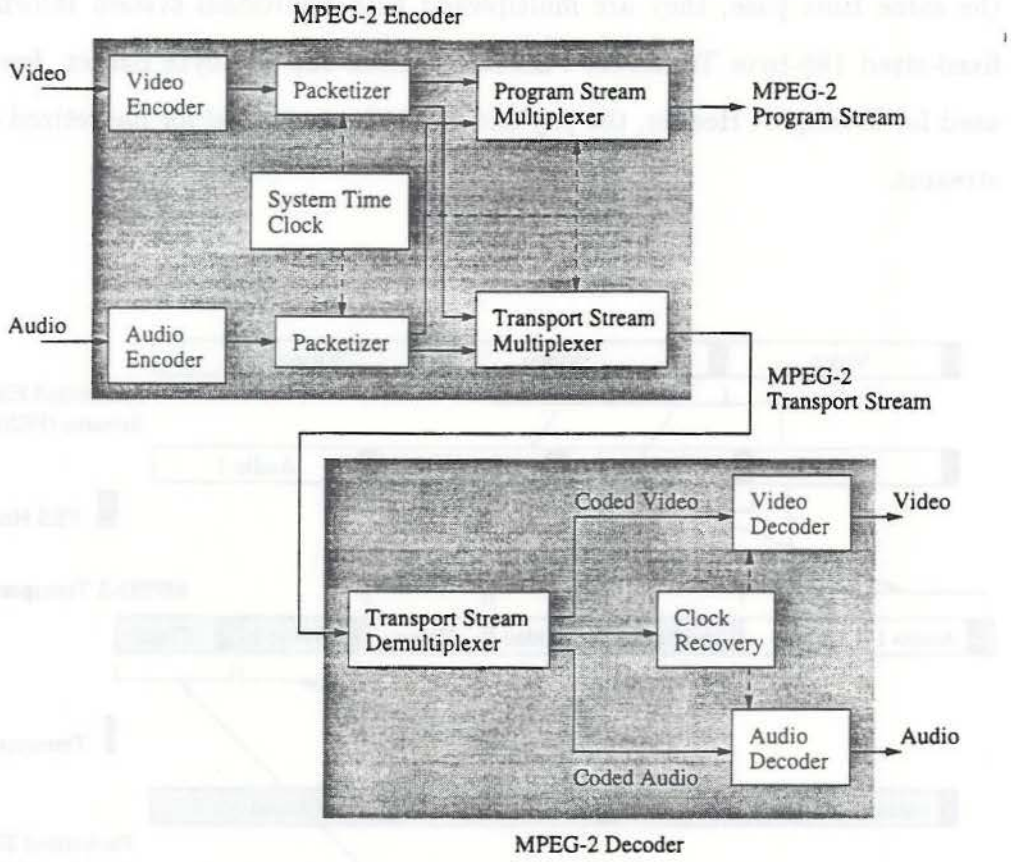

Figure 4.5: MPEG-2 encoder and decoder for Transport Streams.

source sends it. If the decoder is not synchronized to the source, buffer overflows or underflows may occur in the decoder. The clock synchronization is done by using time stamps called *Program Clock References* (PCR) embedded in the MPEG-2 TS. When encoding a program, the MPEG-2 encoder inserts time stamps, which are readings of its clock, into the program periodically (or randomly but keep the spacing of two consecutive PCRs within 100 ms). The MPEG-2 decoder uses received PCRs to synchronize its clock to the program source.

Figure 4.6 shows how a MPEG-2 Transport Stream is constructed from packetized elementary streams of one video stream, two audio streams, and one data stream. Each PES contains a series of variable-sized packets. For those PES share

the same time base, they are multiplexed with additional system information into fixed-sized 188-byte Transport Packets. Within the 188-byte packet, four bytes are used for Transport Header, the payload of 184 bytes is used for packetized elementary streams.

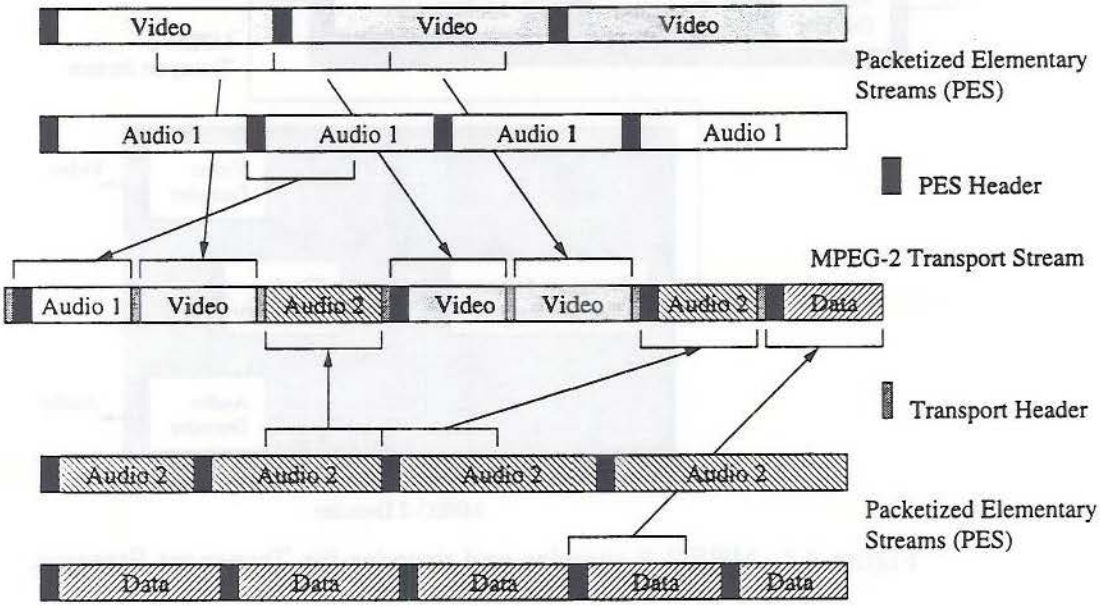

Figure 4.6: Constructing a Transport Stream from packetized elementary streams.

MPEG-2 System standard defines a timing model for media synchronization and clock synchronization. The media synchronization is used to correctly playback a video stream with its associated audio or text. The clock synchronization is used to coordinate the decoder and program source. The clock synchronization is accomplished by sending time stamps in the MPEG-2 stream from the source to the decoder. These time stamps are called Program Clock References (PCR) in MPEG-2

Transport Stream. The PCRs indicate to the decoder what time its clock, called System Time Clock (STC), should be read at the instant the PCRs are received. Both STC and PCR use a 42-bit counter to represent the clock running at a 27 MHz rate. The decoder uses a clock recovery mechanism to synchronize its clock to the program source.

The media synchronization is accomplished by using PCRs, Decoding Time Stamps (DTS) and Presentation Time Stamps (PTS). When a Transport Stream program was encoded, the DTSs and PTSs are embedded in each component. These time stamps indicate the time that a video picture (or audio clip) need to be decoded or presented. During the playback process, the decoder's STC is initialized by the first received PCR. The decoding (or presentation) of a video picture (or audio clip) will be triggered when its DTS (or PTS) matches the system clock. Since all media of a program share the same clock, they will be correctly displayed as they are encoded.

Within the coding of MPEG-2, time stamps are related to the decoding and presentation of video picture and blocks of audio samples. The pictures and blocks are called *Presentation Units*, PU. The encoded data representing the PUs are called *Access Units*, AU. For example, a Video PU (VPU) is a picture, and a Video AU (VAU) is an encoded picture. Some, but not necessarily all, AAUs and VAUs have PTSs associated with them. A PTS indicates the time that the PU (results from decoding the AU which is associated with the PTS) should be presented. Another time stamp, DTS, refers to the time that an AU is to be extracted from the decoder buffer and decoded. Both PTSs and DTSs are 33 bits long. The STC contains two portions, the upper 33 bits incrementing at a 90 KHz rate and lower 9 bits incrementing at 27 MHz. The 33 bits, 90 KHz portion of STC is used for comparison with PTSs and DTSs.

4.3.2 An "Ideal Scheduling"

Since PCRs are readings of the encoder's clock, their values represent the time that they are inserted by the encoder. Assume the first PCR, PCR_1, is inserted at the time represented by its value, $|PCR_1|$. The ith PCR is inserted at the instant that is $|PCR_i| - |PCR_1|$ later than PCR_1. Assume the decoder consumes PCR_1-containing TP at time T_0. If the decoder consumes every PCR_i-containing TP at time $T_0 + |PCR_i| - |PCR_1|$, it playbacks the MPEG-2 TS at the same rate as the encoder encoded it. In addition to clock synchronization, PCRs can also be used as timing reference to schedule the transmission of MPEG-2 TS. Therefore, an "ideal schedule" can be defined as follow: *At any point of the delivery path*[1]*, from the program source to the decoder, the TS packet containing* PCR_i *will be transmitted, forwarded, or consumed at the time that* $|PCR_i| - |PCR_1|$ *later than the TS packet containing* PCR_1.

Intuitively, a variable bit-rate (VBR) transport is suitable to transmit MEPG-2 TS in order to follow the "ideal schedule". Assume the byte-order of ith PCR in a MPEG-2 TS is $b(PCR_i)$. The transmission rate in a PCR interval (the interval between two consecutive PCRs), from PCR_i to PCR_{i+1}, should be the amount of data between them divided by the length of the interval, which is

$$\frac{b(PCR_{i+1}) - b(PCR_i)}{|PCR_{i+1}| - |PCR_i|} \tag{4.3}$$

This means the rates required for each PCR interval are different. However, the VBR service provided by ATM networks is difficult to enforce the "ideal schedule". Since VBR service takes advantage of statistically multiplexing by sharing the bandwidth dynamically among all traffic within a service class. It only guarantees

[1]For pre-recorded MPEG-2 streams, the delivery path includes the storage device and network interface of the server, ATM networks, the network interface and decoder of the client.

statistical quality of service based on a set of traffic descriptors such as peak rate, burst length, and sustained rate. The transmission is even difficult to manage if a traffic shaper is used to regulate the cell transmission.

On the other hand, the constant bit-rate (CBR) service guarantees no or negligible cell loss, delay, and jitter by allocating network resources (link bandwidth and buffer) based on the requested peak transmission rate. Although CBR provides deterministic transmission, it can not be used for the "ideal schedule". Moreover, when transmitting VBR-coded MPEG streams for continuous playback, it often requires a large buffer at the receiving side to accommodate accumulated data due to the difference between the transmission of a server and the data consumption by a decoder [44].

In this chapter, we study two new transmission schemes which employ the timing information from PCRs to provide transmission schedules which are approximations of the "ideal schedule". These two schemes collect timing information with the help of a PCR-assist mechanism discussed in Section 4.3.3.

4.3.3 PCR-assist Mechanism

PCRs are embedded in MPEG-2 Transport Packets to facilitate clock synchronization between the decoder and the program source. The MPEG-2 standard recommends the interval between two consecutive PCRs, called PCR interval, should not be longer than 100 ms. In most implementation, PCRs are periodically inserted by the encoder. PCRs can also be used as timing reference to regulate the transmission of MPEG-2 TPs. Figure 4.4 shows a possible implementation of *PCR-assist mechanism* in the network interface. It is capable of reading PCR values of MPEG-2 TPs while delivering TPs with different ATM services.

At the beginning of video transmission, the first PCR obtained from the storage is stored in a counter which also runs at 27 MHz as the System Time Clock in a MPEG-2 encoder or decoder. Whenever a TP with a PCR is ready to be transmitted, the network interface compares the PCR to the content of the counter. If the PCR is smaller than the counter, the transmission of MPEG-2 TS is behind of its original schedule. Otherwise, the transmission is earlier than its original schedule. In Sections 4.4 and 4.5, we will describe the two PCR-assist transmission schemes which employ the PCR-assist mechanism to adjust the transmission rate.

4.4 PCR-assist CBR

Using CBR service to transmit MPEG-2 streams requires a large buffer and a long start-up delay, especially for VBR-coded MPEG streams [44]. With CBR services, the server keep transmitting MPEG-2 streams at a constant rate. The transmission rate should be high enough in order to prevent starvation occurs in the client's buffer. For VBR-coded MPEG-2 streams, the consumption rate of decoder varies from time to time, depends on the content of the biggest component of a program, video. If the consumption rate can not match with the transmission rate, large amount of data will be accumulated in the client's buffer.

In this section, we describe a new scheme proposed by IBM Rochester [28], called PCR-assist CBR (PCBR), which utilizes the PCRs embedded in MPEG-2 streams to regulate the outgoing traffic. In PCBR, a CBR virtual channel is established with a transmission rate higher than the required average rate of a MPEG-2 TS. The PCR-assist mechanism of the network interface monitors the video stream for presence of PCRs. The value of the PCR in the stream is used to make sure that the PCR-containing packets are sent out at the correct time. The value of the first PCR is stored in a counter of the network interface as shown in Figure 4.4. The counter

running at a rate of 27 MHz. The content of the counter is used to compare with any PCR transmitted through the network interface. Denote the value of ith PCR as $|PCR_i|$ and C_i is the value of the counter at the instant ith PCR appeared, except that C_1 is assigned as $|PCR_1|$. During the transmission, whenever $|PCR_i| > C_i$ for $i > 1$, the delivery is faster than it should be. If $|PCR_i| < C_i$ for any $i > 1$, on the other hand, the delivery is behind of its schedule. In the PCBR scheme, whenever $|PCR_i| > C_i$ happens the network interface will hold up the delivery until C_i reaches $|PCR_i|$. The transmission is idle during this period of time. The objective is to minimize the required buffer space at the client side. If $|PCR_i| < C_i$ for any $i > 1$, the network interface does not change the transmission rate. Since the transmission is already behind of its schedule.

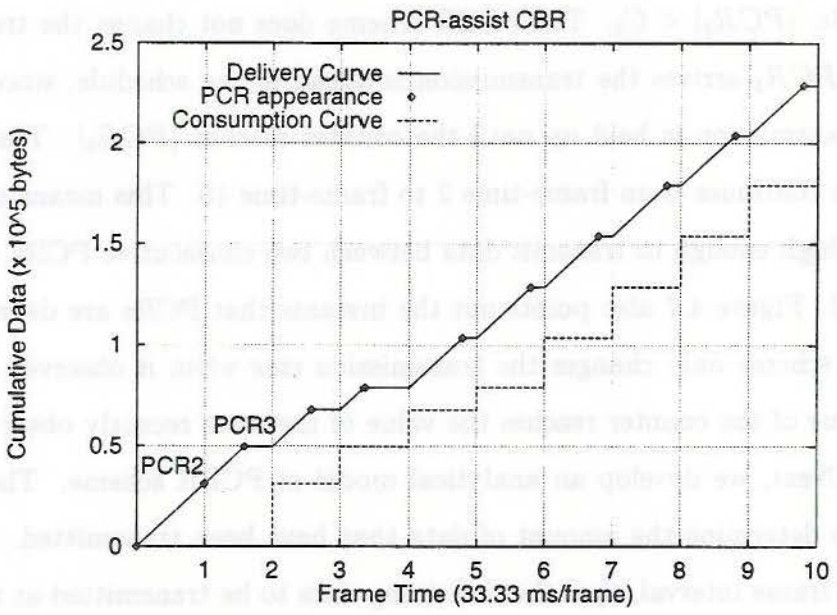

Figure 4.7: The delivery curve of PCR-assist CBR and consumption curve of the decoder.

The behavior of PCBR scheme is illustrated in Figure 4.7. The bottom curve, denoted as a *Consumption Curve*, represents the amount of cumulative data consumed by the decoder[2]. In Figure 4.7, we also assume the network delay is constant and the decoder extracts one coded video frame from the buffer instantaneously every 33.33 ms (i.e. 30 frames per second). The upper curve, denoted as a *Delivery Curve*, is the amount of cumulative data transmitted by the server. For simplicity, we assume PCRs are embedded in the MPEG-2 stream periodically. The interval of any two consecutive PCRs is 33.33 ms, the same as the frame interval. The start-up delay used in Figure 4.7 is two-frame time. The decoder waits for two-frame time before decoding and presenting the MPEG-2 stream. Assume the byte-order of ith PCR in a MPEG-2 stream is $b(PCR_i)$. If the most recently observed PCR is PCR_i. The PCR-assist mechanism will not see the next PCR (byte order $b(PCR_{i+1})$) until the data between them ($b(PCR_{i+1}) - b(PCR_i)$) have been transmitted. In Figure 4.7, when the PCR-assist mechanism sees PCR_2 the transmission is slightly behind the schedule, $|PCR_2| < C_2$. The PCBR scheme does not change the transmission rate. When PCR_3 arrives the transmission is ahead of the schedule, since $|PCR_3| > C_3$. The transmission is held up until the counter reaches $|PCR_3|$. This "go-and-stop" pattern continues from frame-time 2 to frame-time 10. This means the transmission rate is high enough to transmit data between two consecutive PCRs within one PCR interval. Figure 4.7 also points out the instants that PCRs are detected. Note that PCBR scheme only changes the transmission rate when it observes a PCR or when the value of the counter reaches the value of the most recently observed PCR.

Next, we develop an analytical model of PCBR scheme. The model will be used to determine the amount of data that have been transmitted. Assume that in the ith frame interval, $\hat{S}_i$ is the remaining data to be transmitted at the beginning of

[2]Note that even though these curves are derived from real MPEG-2 traces. The curve and data used here are for illustration purpose only.

the interval. S_i is the size of frame i. $\bar{S}_i$ is the amount of data have been transmitted in ith frame interval. F is the frame interval. For a given rate R_{PCBR}, $\bar{S}_i$ and $\hat{S}_i$ can be calculated by the following recurrence equations:

$$\bar{S}_i = \min(R_{PCBR} \times F, \hat{S}_i + S_i) \quad (4.4)$$

$$\hat{S}_{i+1} = \max(\hat{S}_i + S_i - \bar{S}_i, 0) \quad (4.5)$$

$$\hat{S}_1 = 0$$

In Equation 4.4, $\bar{S}_i$ is equal to $R_{PCBR} \times F$ if the transmission rate is not high enough to transmit data of size $\hat{S}_i + S_i$ within one PCR interval. Equation 4.5 calculates the remaining data for the next interval. The minimum buffer required at the client side can be determined by computing the maximum difference between *delivery curve* and *consumption curve*. The amount of data that have been received at time $n \times F$ is $\sum_{i=1}^{n} \bar{S}_i$. The amount of data that have been consumed right before time $n \times F$ is A_{n-1}. The maximum difference between *delivery curve* and *consumption curve* for a given start-up delay P is

$$f_{PCBR}(P) = \max_{1 \le n \le N} \left(\sum_{i=1}^{n+P} \bar{S}_i - \sum_{i=1}^{n-1} S_i \right) \quad (4.6)$$

4.5 PCR-Assist Dual-Rate CBR

In PCBR scheme, the transmission rate is either R_{PCBR} or zero. A virtual channel with a peak transmission rate of R_{PCBR} is established before the delivery. As the experimental results shown in Section 4.6, compared to traditional CBR service, PCBR requires less buffer at the client side with higher transmission rate. One drawback of PCBR scheme is that its go-and-stop delivery curve may intersects with

the consumption curve (i.e. starvation) if there is not enough cushion between them.

To reduce the transmission rate and to reduce the chance of starvation, we propose a more flexible transmission scheme, called PCR-assist Dual-Rate CBR (PD-CBR). In PDCBR scheme, the transmission is switched between two rates, a high and a low rates denoted as R_{high} and R_{low}. The rate used depends on if the transmission is earlier or later than its schedule. If $|PCR_i| > C_i$ for any $i > 1$, the delivery is earlier than it should be. Instead of holding up the delivery as PCBR, PDCBR changes the rate to R_{low} to slow down the accumulation in client's buffer. If $|PCR_i| < C_i$ for any $i > 1$, on the other hand, the delivery is behind its schedule. The rate is changed to R_{high} in order to prevent starvation at the client side. Actually, PCBR is a special case of PDCBR. For PCBR, R_{high} is R_{PCBR} and R_{low} is zero. Note that the rate can only be changed when a PCR appears, which means the amount of data between the previous and the current PCR have been delivered.

Figure 4.8 shows the delivery curve of PDCBR and the consumption curve of the decoder. The consumption curve and the start-up delay are the same as the previous section. In this example, the R_{high} rate is a little bit higher than R_{CBR}, but is lower than R_{PCBR} used in Figure 4.7. R_{low} is a little bit lower than R_{CBR}. From the appearance of PCRs (denoted as points in the figure), it is easy to understand the behavior of PDCBR scheme. Initially, R_{high} rate is used between PCR_1 and PCR_2. At the instant PCR_2 appears, the PCR-assist mechanism detects that $|PCR_2| > C_2$. It switches the rate to R_{low} when transmitting packets containing PCR_2. Rate R_{low} is used until PCR_6 appears (between frame-time 5 and 6). Because when PCR_3, PCR_4, and PCR_5 appear, the transmission is faster than it should be. When the PCR-Assist mechanism observes PCR_6, the transmission is behind of its schedule (i.e. $|PCR_6| < C_6$). The rate is switched back to R_{high} to prevent starvation at the client side. Intuitively, the buffer space required at client side for PDCBR should be smaller than that of CBR, but may be bigger than that of PCBR.

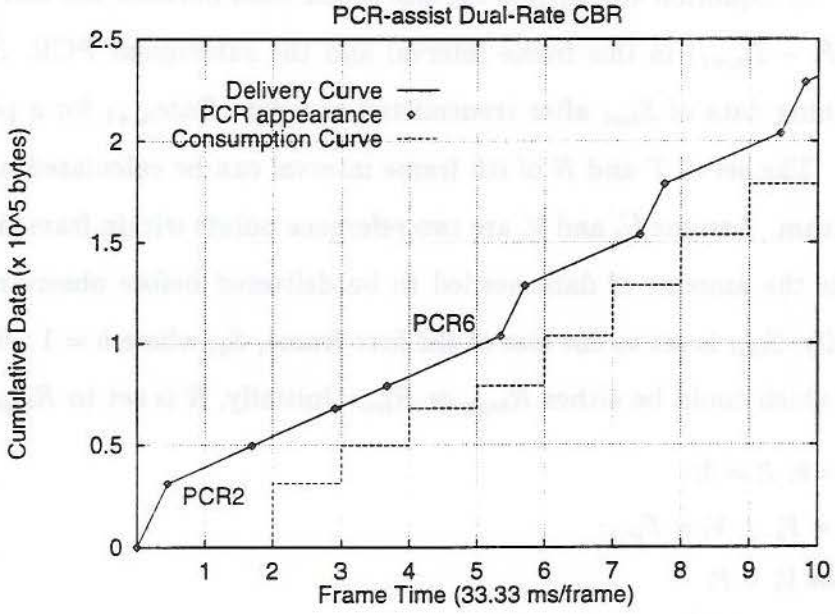

Figure 4.8: The delivery curve of PCR-assist Dual-Rate CBR and consumption curve of the decoder.

An analytical model of PDCBR scheme can be developed as follows. Assume that in ith frame interval (F_{i-1}, F_i), the PCR-assist mechanism observes n_i PCRs. For example, there are two PCRs during the sixth frame interval in Figure 4.8. The instants that PCRs are observed divide the ith frame interval into $n_i + 1$ periods, denoted as $T = \{T_1, ..., T_{n_i+1}\}$. The corresponding transmission rates used in each period are $R = \{Rate_1, ..., Rate_{n_i+1}\}$. The amount of data have been transmitted, $\bar{S}_i$, and the remaining data need to be transmitted at the beginning of next frame interval, $\hat{S}_{i+1}$, are

$$\bar{S}_i = \sum_{k=1}^{n_i+1} T_k \times Rate_k \tag{4.7}$$

$$\hat{S}_{i+1} = \max(S_{last} - T_{n_i+1} \times Rate_{n_i+1}, 0) \quad (4.8)$$

In Equation 4.8, S_{last} is the size of the data between the last PCR (appears at time $F_i - T_{n_i+1}$) in this frame interval and the subsequent PCR. $\hat{S}_{i+1}$ equals to the remaining data of S_{last} after transmitted at a rate $Rate_{n_i+1}$ for a period of T_{n_i+1}.

The set of T and R of ith frame interval can be calculated using the following algorithm. Assume U_i and V_i are two reference points within frame interval (F_{i-1}, F_i). S_{last} is the amount of data needed to be delivered before observing the next PCR. Initially, S_{last} is set to the size of the first frame, S_b, where $b = 1$. *Rate* is the current rate, which could be either R_{high} or R_{low}. Initially, R is set to R_{high}.

1 $T = \emptyset$; $R = \emptyset$;
2 $U_i = F_{i-1}$; $V_i = F_{i-1}$;
3 while $V_i < F_i$
4 $\quad V_i = \min(\frac{S_{last}}{Rate} + U_i, F_i)$;
5 $\quad S_{last} = \max(S_{last} - (V_i - U_i) \times Rate, 0)$;
6 $\quad$ add $Rate$ into R
7 $\quad$ add interval (U_i, V_i) into T
8 $\quad$ if $S_{last} == 0$
9 $\quad\quad$ if $\lfloor \frac{V_i}{\text{PCR Interval}} \rfloor < b$
10 $\quad\quad\quad Rate = R_{low}$;
11 $\quad\quad$ else
12 $\quad\quad\quad Rate = R_{high}$;
13 $\quad\quad b = b + 1$;
14 $\quad\quad S_{last} = S_b$;
15 $\quad$ if $Rate == 0$
16 $\quad\quad Rate = R_{high}$;
17 $\quad\quad V_i = F_i$;
18 $\quad U_i = V_i$;

The algorithm can also be applied to PCBR scheme, since it is a special case of PDCBR. Lines 15 to 17 are used for PCBR scheme. For a given start-up delay P, the maximum buffer required at the client side can be determined by computing the maximum difference between *delivery curve* and *consumption curve* like Equation 4.6 in the previous section.

4.6 Comparison of CBR and PCR-Assist Schemes

As mentioned before, video transmission is influenced by three important parameters: buffer size, transmission rate, and start-up delay. Buffer size and transmission rate are related to the resource requirement at the client side. Start-up delay is related to the service provided to the client by a transmission scheme. These three parameters can be used to study the behavior of a video delivery scheme. In this section, we first compare the transmission and buffer requirements of traditional CBR, PCBR and PDCBR schemes with given start-up delays. The experimental results reveal the resource requirement for a specific waiting time. In Sections 4.7, we will further study the relationship among the three three parameters in the two PCR-assist transmission schemes.

Our methodology is to determine the required transmission rates and buffer sizes by adjusting the start-up delay. For a given value of start-up delay, we compute the minimum rate requirement with a required buffer size to guarantee the jitter-free delivery in each of the following transmission schemes: CBR, PCBR, and PDCBR. The guaranteed delivery ensures that the starvation never happens. For simplicity, we assume the PCR interval is the same as a frame interval, i.e., a PCR occurs every 33.33 ms. Four MPEG-2 trace data which close to CBR-encoded are used in this

analysis. Due to lack of long VBR-encoded MPEG-2 TS available in public domain, we also use six VBR-encoded MPEG-1 and two Motion JPEG [58] (M-JPEG) traces for a complete study. JPEG encoding format was originally designed for encoding and decoding still images. For a sequence of video frames, motion JPEG only uses intra-frame coding scheme. Each video frame can be encoded and decoded independently. Table 4.1 shows the contents of these video sequences.

Table 4.1: Video Contents.

Video Name	Encoding	Content
mtv	MPEG-1	Music Clips
adv	MPEG-1	Advertisement of Graphic Products
silence	MPEG-1	Movie: The Silence of the Lambs
soccer	MPEG-1	Sports: World Soccer Cup 1994 Final: Brazil vs Italy
6trans	MPEG-2	Transport Stream test bit stream
horse	MPEG-2	Transport Stream test bit stream
t260	MPEG-2	Movie: Terminator II
acup	MPEG-2	Sports: American Cup Yacht Race
backdraft	M-JPEG	Movie: Backdraft
fugitive	M-JPEG	Movie: Fugitive

For CBR scheme, given a start-up delay, the calculation of minimum rate requirement with a required buffer size is based on the discussion in Section 4.2. The minimum rate and buffer requirement under CBR scheme will be used as a basis to compare with two PCR-assist transmission schemes. We call the minimum rate requirement as R_{CBR}. As mentioned in [44], if a rate lower than R_{CBR} is used to deliver a video stream, the delivery curve will intersect with the consumption curve, indicating a buffer starvation situation.

For PCBR scheme, given the same start-up delay as in CBR, the minimum required rate must be higher than R_{CBR}. Otherwise, some video data will arrive late. For example, assume R_{CBR} is the rate used by PCBR scheme, the transmission uses a rate of either R_{CBR} or zero (i.e., held up by the network interface) based on PCR

values. As discussed in Section 4.2, R_{CBR} is the rate that the delivery scheme should transmit during the entire session to avoid starvation at the client side. Whenever the transmission is held up by the PCR-assist mechanism, the server will not be able to send any data until the counter reaches the same value as the transmitting PCR. Therefore, a higher rate (R_{PCBR}) than R_{CBR} for the same start-up delay is required for a jitter-free delivery. In the analysis, we set R_{PCBR} as R_{CBR} initially. Then incrementally add 1% of R_{CBR} to R_{PCBR} until the rate is applicable to transmit video stream without causing starvation in client's buffer. The corresponding buffer requirement can then be computed thereafter.

For PDCBR, the determination of high and low rates is needed. However, the number of feasible combinations of high and low rates can be large. Here, we restrict the search space by limiting the high rates from the set $R_H = \{R_{CBR}, 1.01 \times R_{CBR}, 1.02 \times R_{CBR}, \ldots, 1.19 \times R_{CBR}\}$. This means the high rate is determined based on the rate used in CBR scheme. After choosing one high rate, say R_{high} from R_H, the low rate is determined from R_L: $\{0.05 \times R_{high}, 0.10 \times R_{high}, \ldots, 0.95 \times R_{high}\}$. After trying all possible combinations of high and low rates from R_H and R_L, we report the rate combination which demands the least amount of buffer space and use it to compare with other schemes.

4.6.1 Comparison Using MPEG-1 Traces

Table 4.2 shows the encoding information of six VBR-encoded MPEG-1 trace data[3] which are from [22, 50]. Figures 4.9 and 4.10 show the buffer requirement (the left column) and rate requirement (the right column) versus start-up delays in six MPEG-1 trace data for CBR, PCBR, and PDCBR schemes. There are three curves in each figure of buffer requirement. From top to bottom, they are the buffer

[3]The trace "adv" is available via anonymous FTP from ftp://tenet.berkeley.edu/pub/dbind

requirements of CBR, PDCBR, and PCBR schemes. For each figure of required transmission rates, there are four curves. They are the rate used by PCBR, the high rate used by PDCBR, the rate used by CBR, and the low rate used by PDCBR (also from top to bottom). Note these orders are consist with all the figures. From the experimental results they indeed exhibit insight of each delivery scheme.

Table 4.2: Statistical Data of MPEG-1 Streams.

Video Name	mtv	adv	silence	soccer
Video Length	27.78 min	11.33 min	27.78 min	27.78 min
Picture Size	384 × 288	160 × 120	384 × 288	384 × 288
Picture Pattern	$IBB(PBB)^4$	$IBBPBB$	$IBB(PBB)^4$	$IBB(PBB)^4$
No. of Video Frames	40000	16317	40000	40000

- **Trade-offs between buffer and rate requirements.** CBR service requires lower rates (the third curve) at the cost of higher buffer requirement (the first curve). PCBR and PDCBR schemes, on the opposite, require higher transmission rates with less buffer requirement. PDCBR as expected demands less rate (the second curve) than it is in PCBR scheme (the first curve). For the case of trace "adv" with the start-up delay of 300 frames, two PCR-assist schemes can reduce client buffer requirement from 4.75 Mbytes down to 0.675 Mbytes by allocating network bandwidth of 0.547 Mbits/sec up from 0.516 Mbits/sec. This translates into a reduction of 6.5 times memory requirement at the cost of 6% more transmission rate. The trade-offs between buffer requirement and transmission rate exist in all six traces with various degrees of significance.
- **Impact of start-up delay.** For CBR scheme, while the start-up delay is increasing, the buffer requirement is decreasing. However, the trend is different in two PCR-assist schemes. For example, in delivery of trace "soccer" as shown

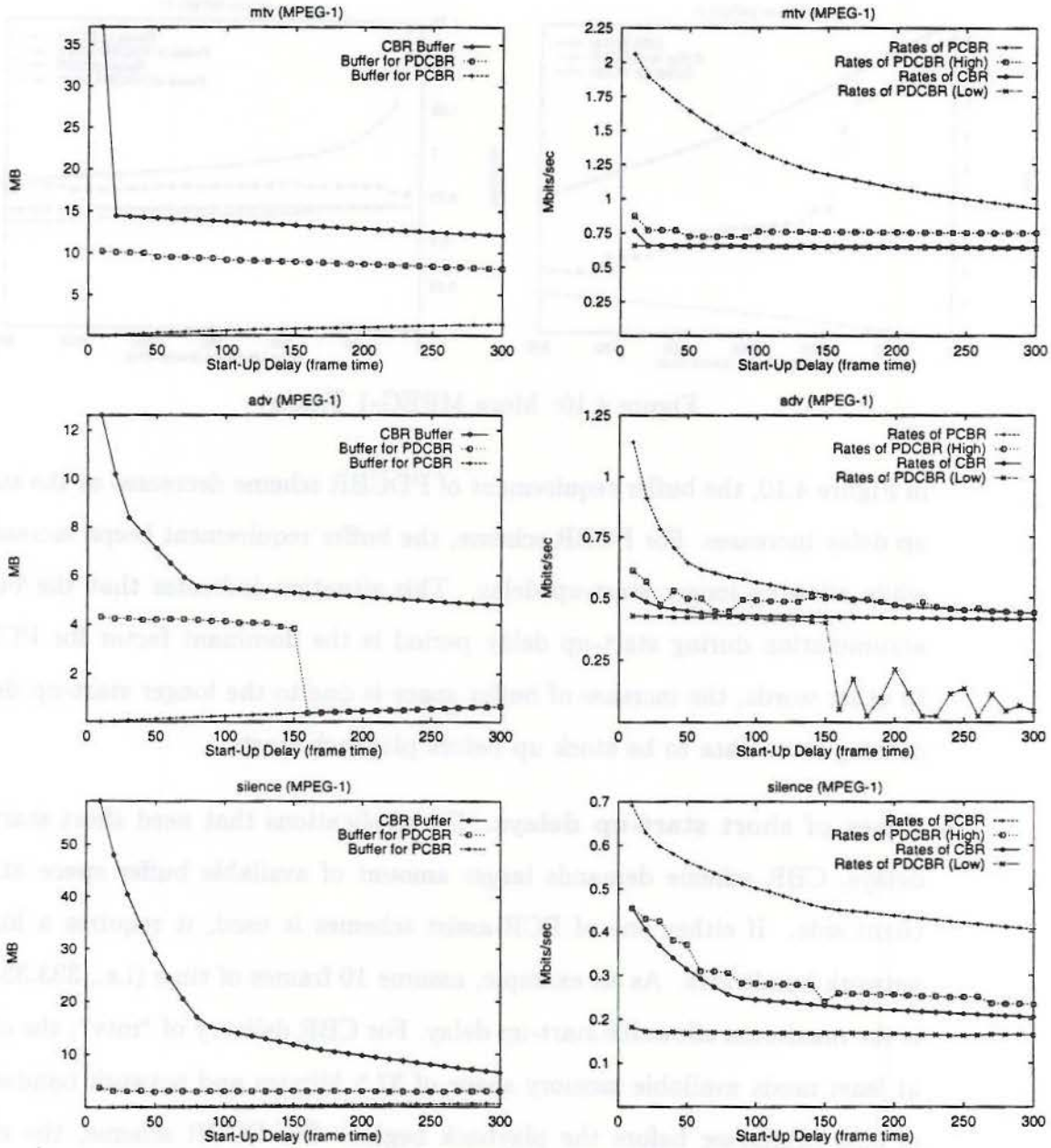

Figure 4.9: Comparison of CBR, PCBR, and PDCBR using MPEG-1 Traces.

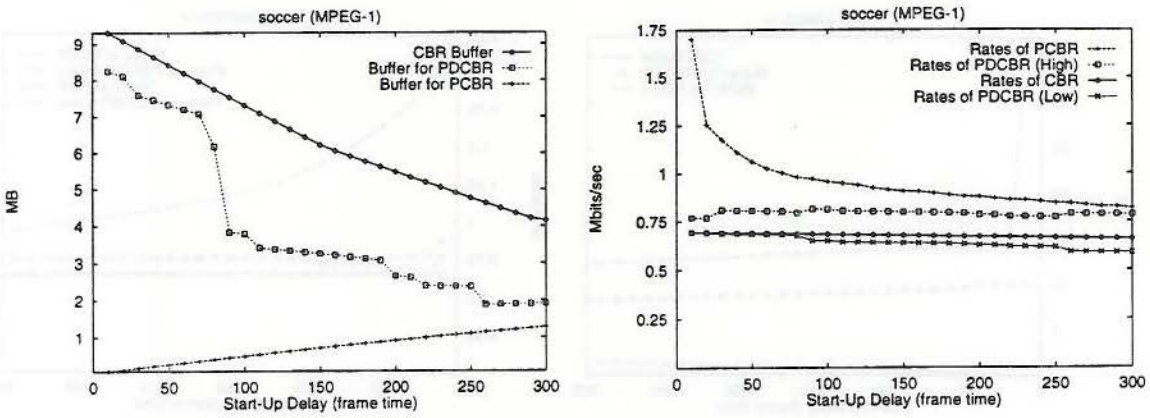

Figure 4.10: More MPEG-1 Traces.

in Figure 4.10, the buffer requirement of PDCBR scheme decreases as the start-up delay increases. For PCBR scheme, the buffer requirement keeps increasing while allowing longer start-up delay. This situation indicates that the buffer accumulation during start-up delay period is the dominant factor for PCBR. In other words, the increase of buffer space is due to the longer start-up delay, causing more data to be stock up before playback starts.

- **Cases of short start-up delays.** For applications that need short start-up delays, CBR scheme demands larger amount of available buffer space at the client side. If either one of PCR-assist schemes is used, it requires a higher network bandwidth. As an example, assume 10 frames of time (i.e., 333.33 ms) is the maximum allowable start-up delay. For CBR delivery of "mtv", the client at least needs available memory space of 37.5 Mbytes and network bandwidth of 0.96 Mbits/sec before the playback begins. For PCBR scheme, the client memory requirement reduces to 0.107 Mbytes and the rate increases to 2.57 Mbits/sec.

4.6.2 Comparison Using MPEG-2 Traces

The experimental results of Section 4.6.1 show the resource requirement of three delivery schemes for MPEG video with high rate variation. In this section, we study their behavior when transmitting video with low rate variation. Table 4.3 shows the encoding information of four MPEG-2 trace data. These traces were encoded with less rate variation. Therefore, they are very close to CBR-encoded MPEG streams. For example, Figure 4.11 displays frame sizes of "t260" trace by averaging every 10 frames. The fluctuations of frame sizes remain minimum.

Table 4.3: Statistical Data of MPEG-2 Streams.

Video Name	6trans	horse	t260	acup
Video Length	20.161 sec	30.13 sec	28.953 min	3.9 min
Picture Size	352 × 480	704 × 480	352 × 480	352 × 480
Picture Pattern	$IBB(PBB)^4$	$IBB(PBB)^4$	IBB	$IBB(PBB)^3$
No. of Video Frames	602	897	52077	7021

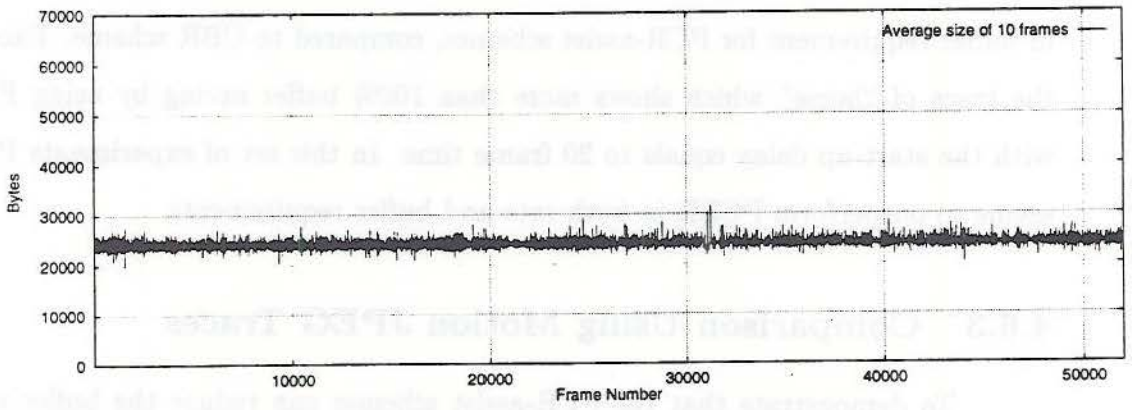

Figure 4.11: t260 (MPEG-2) Trace

Figures 4.12 and 4.13 illustrate buffer (left column) and rate (right column) requirements of CBR, PCBR, and PDCBR schemes. The start-up delays of all figures

range from 1 to 30 frame time. There are three curves in each figure of buffer requirement. For most of the cases from the top most, they are the buffer requirements of CBR, PCBR, and PDCBR schemes. For each figure of the required transmission rates, there are four curves. They are the rates used by PCBR, the high rate used by PDCBR, the rate used by CBR, and the low rate used by PDCBR (also from the top most).

As can be observed from these results, they show the same behavior as in VBR MPEG-1 cases regard to the buffer and rate requirements. CBR transmission demands lower rates while requires a larger buffer space. The two PCR-assist transmission schemes need higher rates but consume smaller buffer spaces. The transmission rate of PDCBR scheme is less than that used in PCBR scheme. However, the degree of rate or buffer gain or lose is not in the same order of MPEG-1 traces. The less rate variation embedded in MPEG-2 traces may contribute to this effect. For instance, the required rates are about the same for all traces in all three transmission schemes for start-up delays longer than 10 frame time. The amount of buffer requirement is not so widely different either. We observed a reduction of 5% to 20% in buffer requirement for PCR-assist schemes, compared to CBR scheme. Except for the trace of "horse" which shows more than 100% buffer saving by using PDCBR with the start-up delay equals to 20 frame time. In this set of experiments PDCBR seems to outperform PCBR in both rate and buffer requirements.

4.6.3 Comparison Using Motion JPEG Traces

To demonstrate that the PCR-assist schemes can reduce the buffer requirement for any VBR-coded video, we further use Motion JPEG video to study their performance. MPEG encoded streams can achieve greater compression ratio because of the use of intra-frame and inter-frame encoding and the use of bi-directional

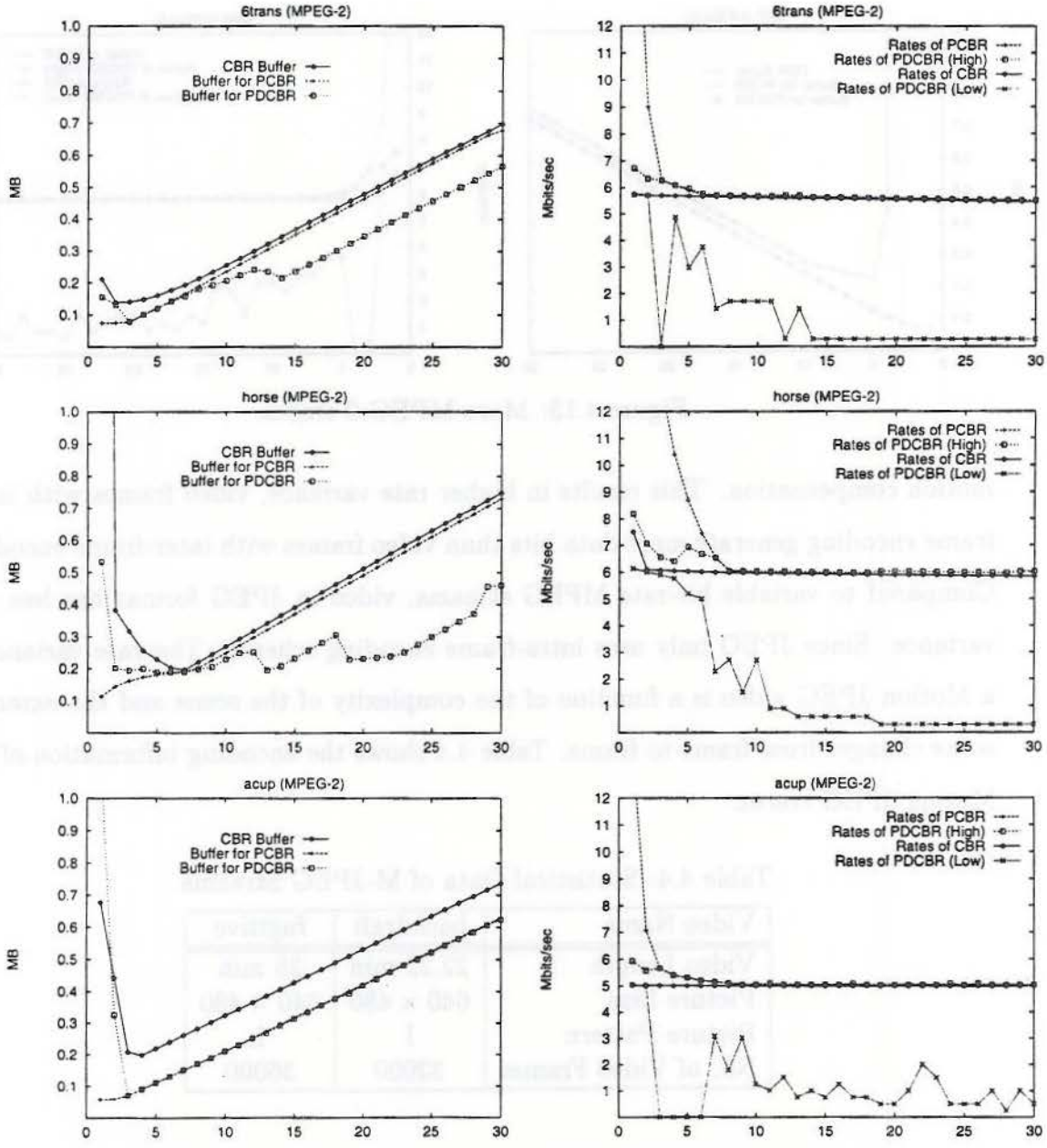

Figure 4.12: Comparison of CBR, PCBR, and PDCBR with MPEG-2 traces.

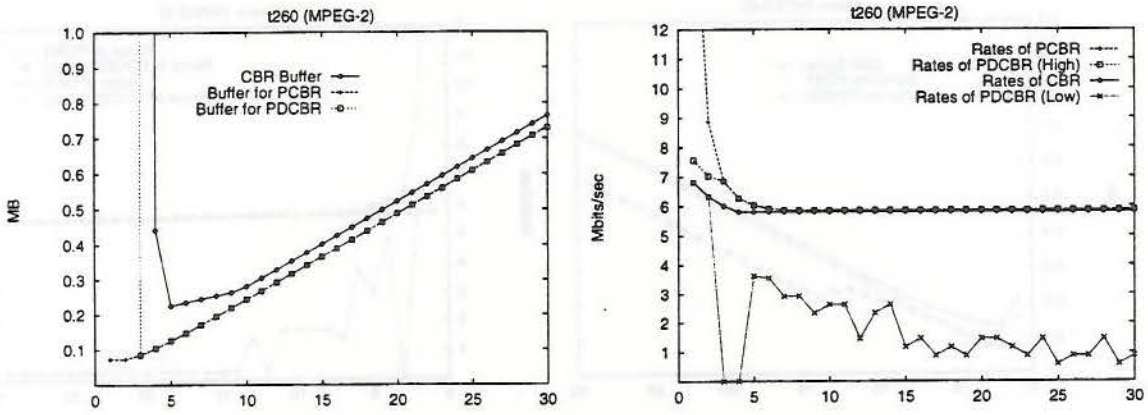

Figure 4.13: More MPEG-2 traces.

motion compensation. This results in higher rate variance, video frames with intra-frame encoding generate more data bits than video frames with inter-frame encoding. Compared to variable bit-rate MPEG streams, video in JPEG format has less rate variance. Since JPEG only uses intra-frame encoding scheme. The rate variance of a Motion JPEG video is a function of the complexity of the scene and the extent of scene changes from frame to frame. Table 4.4 shows the encoding information of two Motion JPEG traces.

Table 4.4: Statistical Data of M-JPEG Streams.

Video Name	backdraft	fugitive
Video Length	22.22 min	25 min
Picture Size	640 × 480	640 × 480
Picture Pattern	I	I
No. of Video Frames	32000	36000

Figure 4.14 shows buffer (left column) and rate (right column) requirements of the three transmission schemes. The start-up delays of all figures also range from 1 to 30 frame time. We observed similar performance trend as in previous sections. The experimental results demonstrate that PCBR and PDCBR schemes require much less

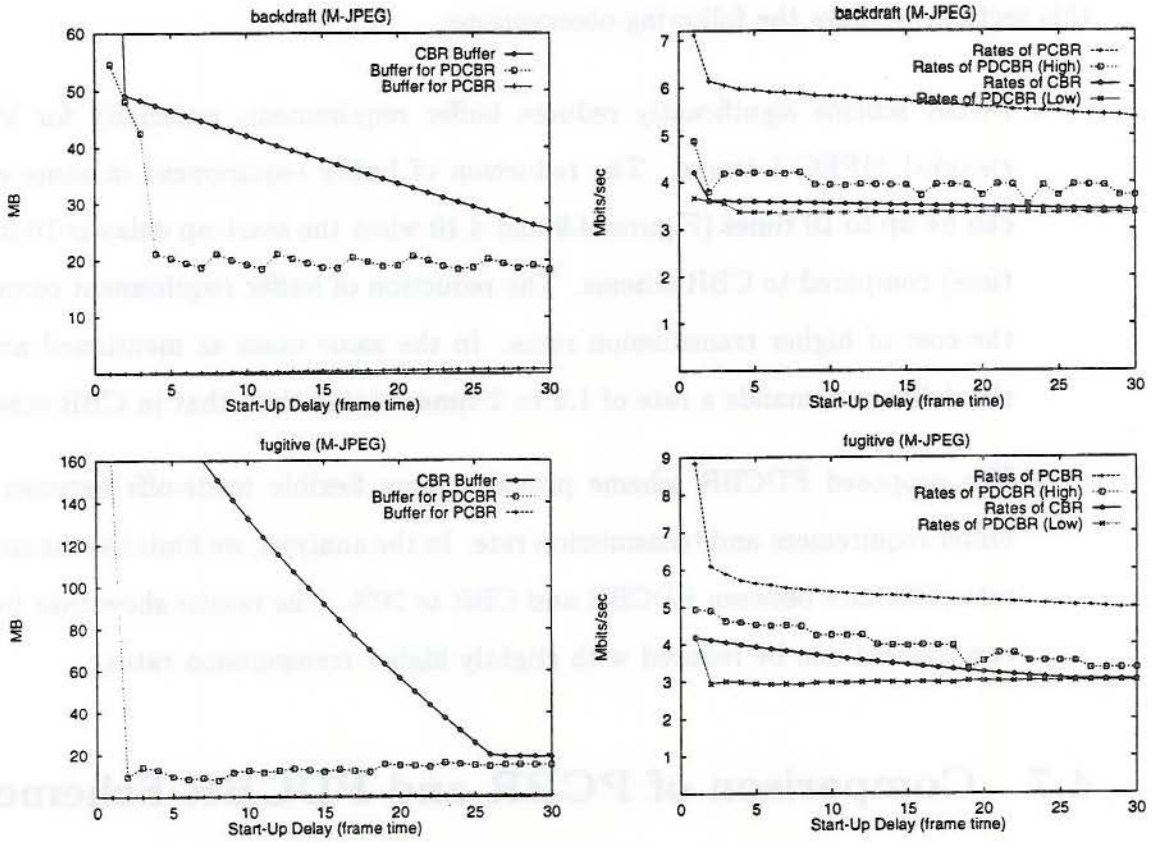

Figure 4.14: Comparison of CBR, PCBR, and PDCBR using M-JPEG traces.

buffer space for any video stream with rate variance, not only for MPEG streams. Figure 4.14 also indicates that the two Motion JPEG traces require more buffer space and transmission rates than the previous two cases (MPEG-1 and MPEG-2 cases). This is because the traces used here have larger picture sizes and the motion JPEG encoding scheme has less compression ratio than MPEG schemes. The average bit of traces "backdraft" and "fugitive" are 3.3 and 3.0 Mbit/sec, respectively.

4.6.4 Observations

Six MPEG-1 (VBR-encoded), four MPEG-2 (close to CBR-encoded) and two Motion JPEG traces are used in the analysis. Based on the experimental results in

this section, we have the following observations:

- PCBR scheme significantly reduces buffer requirement, especially for VBR-encoded MPEG-1 traces. The reduction of buffer requirement in some cases can be up to 10 times (Figures 4.9 and 4.10 when the start-up delay is 10 frame time) compared to CBR scheme. The reduction of buffer requirement comes at the cost of higher transmission rates. In the same cases as mentioned above, the delivery demands a rate of 1.5 to 2 times higher than that in CBR scheme.

- The proposed PDCBR scheme provides more flexible trade-offs between the buffer requirement and transmission rate. In the analysis, we limit the maximum rate difference between PDCBR and CBR to 20%. The results show that buffer requirement can be reduced with slightly higher transmission rates.

4.7 Comparison of PCBR and PDCBR Schemes

In Section 4.6, we compared the buffer and rate requirements of the three transmission schemes. The methodology is to determine the required transmission rates and buffer sizes by adjusting the start-up delay. For some environments, such as video on demand or video transmission through residential networks, the user may have fixed buffer space or fixed link bandwidth. For example, most of the setop boxes which are used to receive and decode video streams have limited memory space due to cost consideration. The communication links between setop boxes and video server may go through cable TV networks or ISDN connections. These links also have restricted bandwidth for video transmission. In Sections 4.7.1 and 4.7.2, we study the performance of both PCBR and PDCBR schemes in these environments.

4.7.1 Fixed Buffer Sizes

With fixed buffer sizes at the receiver side, the objective of this section is to compare the transmission rates required by PCBR and PDCBR schemes in order to provide a continuous playback. The transmission scheme should not either overflow or underflow the receiver's limited buffer space during the entire session of video transmission. Assume the accumulative amount of data consumed by the decoder can be described by a consumption curve (as illustrated in Figure 4.1). The consumption curve can be represented by a discrete function $C(n)$,

$$C(n) = \sum_{i=1}^{n} S_i, \tag{4.9}$$

where S_i is the size of video frame i. Given a fixed buffer size B, there is a *reference curve* which can be denoted as $R(n) = C(n) + B$. A transmission scheme should provides a feasible delivery curve, which represents the accumulative amount of video data delivered to the receiver, between the consumption curve and reference curve.

With a given buffer space, there is also an upper bound for the start-up delay. Because the amount of data pre-loaded into the client's buffer during the period of start-up delay can not excess the capacity of the buffer. In this study, we minimize the required transmission rate by adjusting the start-up delay within the range allowed by the fixed buffer space. Longer start-up delays (more pre-loaded data) may result in lower transmission rates.

Figure 4.15 shows the required transmission rates of PCBR and PDCBR schemes with different buffer sizes. Four MPEG-1 and two Motion JPEG traces are used to compare PCBR and PDCBR schemes. The experimental result with trace "mtv" uses a buffer size of 2 MBytes. The result with trace "adv" uses a buffer size of 1 MBytes. The results from the rest of traces use a buffer size of 4 MBytes. Since different traces have different rate variance. For each trace, we conducted the

performance evaluation with various buffer sizes. We present those results that can demonstrate the difference between PCBR and PDCBR schemes.

There are three curves in each figure. From top to bottom, they are the transmission rate required by the PCBR scheme, the high rate and the low rate used by the PDCBR scheme. For a given buffer size, the required transmission rates are obtained by adjusting the start-up delay. The start-up delay is controlled by the amount of data pre-fetched into the receiver's buffer. It is the time required to pre-fetched a certain amount of data before the playback. Therefore, for those points closer to the left side of each figure, video transmission has smaller start-up delays because it only needs to wait for smaller amount of data to be re-fetched. For those points closer to the right side of each figure, video transmission needs to wait for larger amount of data be pre-fetched. In the former case, video transmission has less data in decoder's buffer for cushion.

From Figure 4.15, we observed that with larger start-up delays PCBR and PDCBR schemes require similar transmission rates. However, the PDCBR scheme requires lower transmission rates than PCBR with smaller start-up delays. This means the PCBR scheme requires higher transmission rates than PDCBR when the user demands low delays. Because of the "go-and-stop" transmission pattern of PCBR scheme, it requires higher transmission rate to avoid starving decoder's buffer (i.e. avoid intersection between the delivery curve and the consumption curve).

4.7.2 Fixed Transmission Rates

For a given transmission rate for PCBR scheme or a given pair of high rate and low rate for PDCBR scheme, there will be a fixed delivery curve. In order to provide jitter-free video transmission (avoid intersection between the delivery curve and the consumption curve) for continuous playback, we need to include appropriate

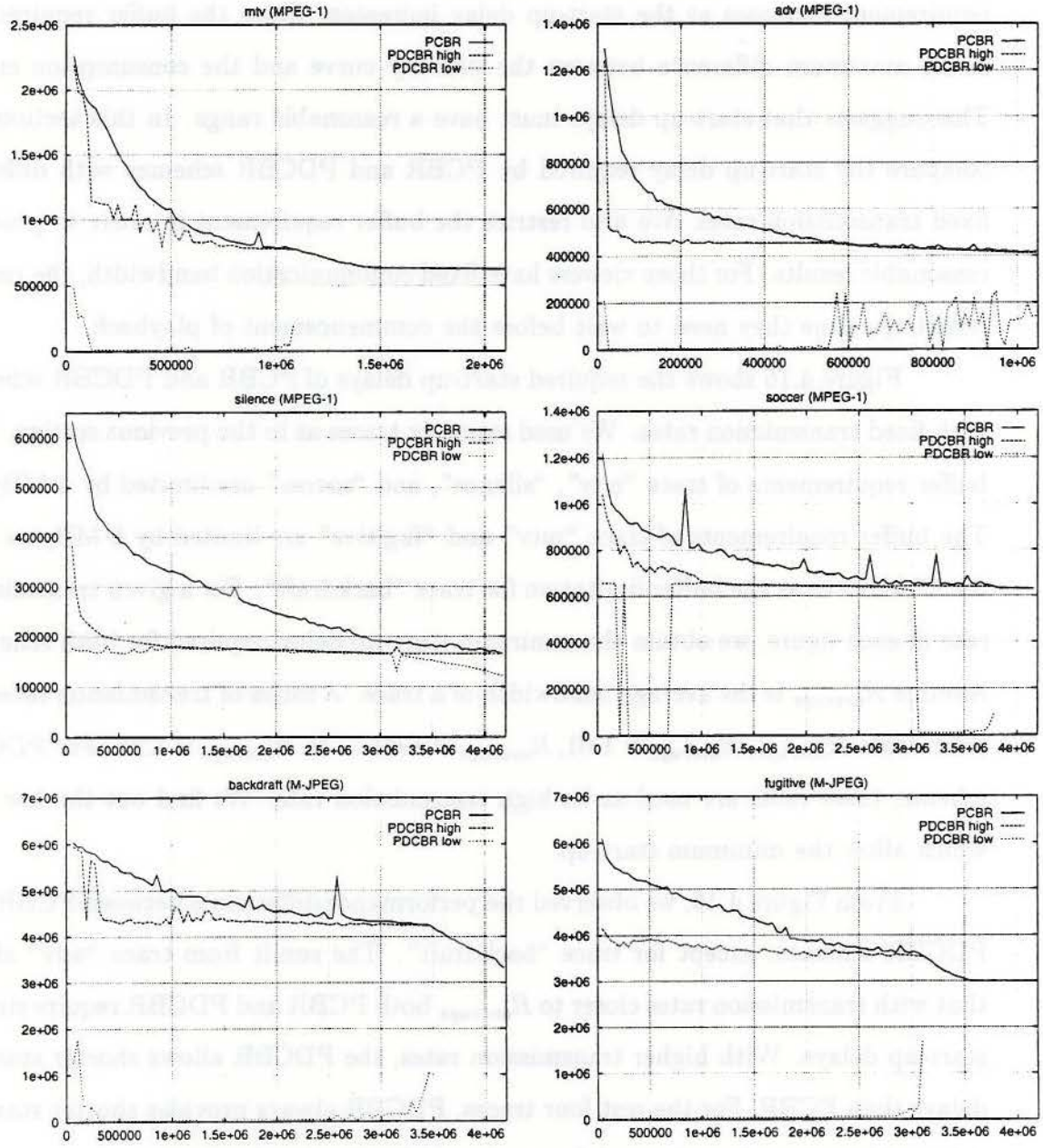

Figure 4.15: Required transmission rates with fixed buffer sizes.

start-up delays as the Figure 4.2 in Section 4.2 suggests. On the other hand, the buffer requirement increases as the start-up delay increases. Since the buffer requirement is the maximum difference between the delivery curve and the consumption curve. This suggests that start-up delays must have a reasonable range. In this section, we compare the start-up delay required by PCBR and PDCBR schemes with different fixed transmission rates. We also restrict the buffer requirement in order to produce reasonable results. For those viewers have fixed communication bandwidth, the results reflect the time they need to wait before the commencement of playback.

Figure 4.16 shows the required start-up delays of PCBR and PDCBR schemes with fixed transmission rates. We used same six traces as in the previous section. The buffer requirements of trace "adv", "silence", and "soccer" are limited by 4 MBytes. The buffer requirements of trace "mtv" and "fugitive" are limited by 8 MBytes. We use 16 MBytes as the buffer limitation for trace "backdraft". For a given transmission rate in each figure, we obtain the minimum start-up delay required for both schemes. Assume $R_{average}$ is the average bandwidth of a trace. A series of transmission rates are used, from $R_{average}, R_{average} \times 1.01, R_{average} \times 1.02, ...$ to $R_{average} \times 1.24$. For PDCBR scheme, these rates are used as its high transmission rate. We find out the low rate which allow the minimum start-up.

From Figure 4.16, we observed the performance differences between PCBR and PDCBR schemes, except for trace "backdraft". The result from trace "adv" shows that with transmission rates closer to $R_{average}$ both PCBR and PDCBR require similar start-up delays. With higher transmission rates, the PDCBR allows shorter start-up delays than PCBR. For the rest four traces, PDCBR always provides shorter start-up delays. Note that the unit used for start-up delay is second.

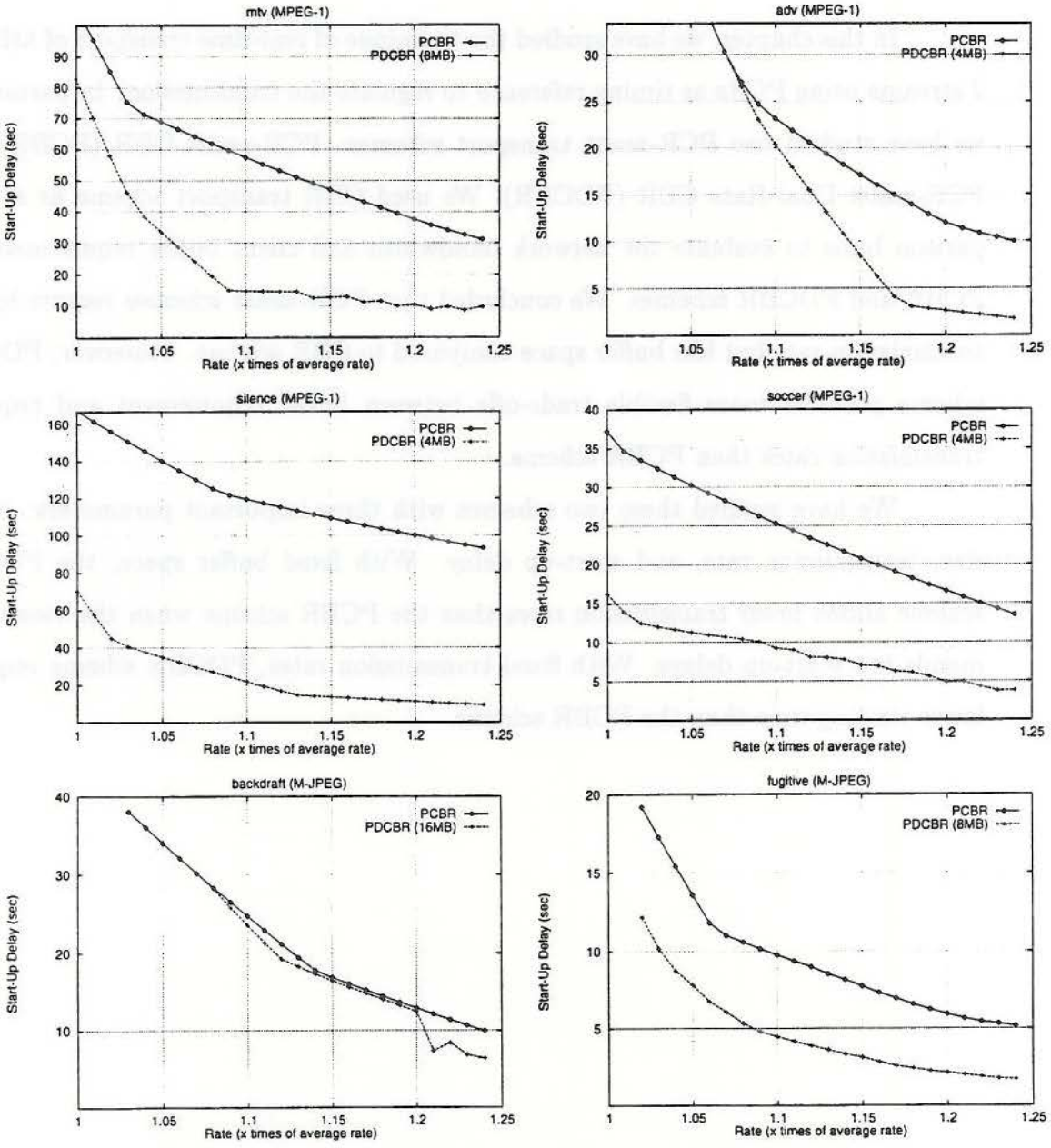

Figure 4.16: Required start-up delays with fixed transmission rates.

4.8 Summary

In this chapter, we have studied the technique of real-time transport of MPEG-2 streams using PCRs as timing reference to regulate the transmission. In particular, we have studied two PCR-assist transport schemes: PCR-assist CBR (PCBR) and PCR-assist Dual-Rate CBR (PDCBR). We used CBR transport scheme as a comparison basis to evaluate the network bandwidth and client buffer requirement for PCBR and PDCBR schemes. We concluded that PCR-assist schemes require higher transmission rate but less buffer space compared to CBR scheme. Moreover, PDCBR scheme provides more flexible trade-offs between buffer requirement and required transmission rates than PCBR scheme.

We have studied these two schemes with three important parameters: buffer size, transmission rate, and start-up delay. With fixed buffer space, the PDCBR scheme allows lower transmission rates than the PCBR scheme when the viewer demands low start-up delays. With fixed transmission rates, PDCBR scheme requires lower waiting time than the PCBR scheme.

Chapter 5

Conclusion

Switch-based high speed networks possess superior features, such as high data transfer rates, low latency, scalability, and support for multiple classes of service, than legacy networks. For application developers, the challenge lies on how to utilize the high performance communication provided by the switch-based high speed networks. For system designer, the challenge lies in how to deliver these features to application level by reducing overheads from hardware and software components. In this thesis, we have concentrated our effort on the high speed network support for two important classes of applications in network-centric computing: network computing and multimedia communications.

Our contribution is summarized as follows.

- *High speed network support for cluster computing.* In cluster computing environment, distributed applications are implemented by partitioning the computation into tasks and assigning them to processes which collaborated with each other based on a message passing model. The speed of message passing is critical to the performance of cluster computing. We have carefully examined a popular cluster computing environment and enhanced its communication performance by reducing the protocol processing overhead and employing features provided by the underlying high speed networks. The protocol processing overhead is reduced by bypassing higher layer protocols and using low level application programming interfaces. The communication performance is further improved with features of the underlying networks, such as multicasting communication

and credit-base flow control.

Based on the experimental results from the same cluster computing environment on local ATM and HIPPI networks, we have improved the achievable throughput while preserving low communication latency. This demonstrates the feasibility and potential of network computing on clusters of workstations.

- *High speed network support for meta-computing.* Meta-computing is one possible extension of cluster computing. The advantage of meta-computing is allowing researchers to utilize geographically distributed computing resources connected via local and wide area networks to solve large problems. We have studied two feasible approaches to facilitate the internetworking of local HIPPI networks and wide area ATM networks for meta-computing. They are HIPPI Tunneling and IP Routing. The former provides interconnection at the physical layer and the later supports internetworking at the network layer. We have compared these two approaches in terms of their network connectivities, protocol overheads, and flow control. The impact of flow control of upper layer protocol on the performance is also presented. The unique feature of our study is that both approaches were implemented and evaluated in the same network infrastructure.

- *High speed network support for multimedia communications.* Network delivery for continuous media is an inherently difficult problem due to the time-sensitive nature of the data and its rate variance. Transmitting compressed variable bit rate video with traditional constant bit-rate (CBR) service for continuous playback requires a large buffer at the viewer's side. We have studied two new CBR transmission schemes which utilizing the timing information embedded in the MPEG-2 stream and a hardware-assist mechanism at the network interface. The two schemes, called PCR-assist CBR (PCBR) and PCR-assist Dual-Rate CBR (PDCBR), reduce buffer requirement by using the timing information to

regulate their transmission. We have compared their performance with traditional CBR service with real video traces.

Bibliography

[1] ISO/IEC 13818-1. "Information Technology - Generic Coding of Moving Pictures and Associated Audio, Part 1: Systems, Recommendation ITU H.222.0". International standard, International Standardization Organization, November 1994.

[2] ISO/IEC 13818-2. "Information Technology - Generic Coding of Moving Pictures and Associated Audio, Part 1: Video, Recommendation ITU H.262". International standard, International Standardization Organization, November 1994.

[3] ISO/IEC 13818-3. "Information Technology - Generic Coding of Moving Pictures and Associated Audio, Part 3: Audio". International standard, International Standardization Organization, November 1994.

[4] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and V. Sunderam. "A User's Guide to PVM (Parallel Virtual Machine)". Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, Oak Ridge, TN, July 1991.

[5] M. Blumrich, C. Dubnicki, E.W. Felten, and K. Li. "Virtual Memory-Mapped Network Interfaces". *IEEE Micro*, pages 21–28, Feb. 1995.

[6] J. Boudec. "The Asynchronous Transfer Mode: A Tutorial". *Computer Networks and ISDN Systems*, 24:279–309, 1992.

[7] R. Bulter and E. Lusk. "User's Guide to the P4 Programming System". Technical Report ANL-92/17, Argonne National Laboratory, 1992.

[8] N. Carriero and D. Gelernter. "Linda in Context". *Communications of the ACM*, 32(4):444–458, April 1989.

[9] J.D. Cavanaugh. "Protocol Overhead in IP/ATM Networks". Technical report, Minnesota Supercomputer Center, Inc., Aug. 1994.

[10] Sheue-Ling Chang, David H.C. Du, Jenwei Hsieh, Mengjou Lin, and Rose P. Tsang. "Enhanced PVM Communications over a High-Speed Local Area Network". *IEEE Parallel and Distributed Technology*, pages 20–32, Fall 1995.

[11] B. Chinoy and K. Fall. "TCP/IP and HIPPI Performance in the CASA Gigabit Testbed". In *USENIX High-Speed Networking Symposium*, page 45, Aug. 1994.

[12] C. Dubnicki, L. Iftode, E.W. Felten, and K. Li. "Software Support for Virtual Memory-Mapped Communication". In *Proc. of the 1996 International Parallel Processing Symposium*, pages 372–381, April 1996.

[13] Dave Dunning and Greg Regnier. "Virtual Interface Architecture". In *Proc. of the Hot Interconnect Symposium V*, August 1997.

[14] D.E. Culler et. al. "Parallel Computing on the Berkeley NOW". In *Proc. of the 9th Joint Symposium on Parallel Processing*, Kobe, Japan, 1997.

[15] W. Feng, F. Jahanian, and S. Sechrest. "Optimal Buffering for the Delivery of Compressed Prerecorded Video". In *Proc. of the IASTED/ISMM International Conference on Networks*, January 1995.

[16] W. Feng and J. Rexford. "A Comparison of Bandwidth Smoothing Techniques for the Transmission of Prerecorded Compressed Video". In *IEEE Infocom '97*, page 58, April 1997.

[17] W. Feng and S. Sechrest. "Critical Bandwidth Allocation for Delivery of Compressed Video". *Computer Communications*, 18(10):709–717, October 1995.

[18] Inc. Fore Systems. *"ForeRunner SBA-200 ATM SBus Adapter User's Manual"*. Fore Systems, Inc., 1993.

[19] Message Passing Interface Forum. *"MPI: A Message-Passing Interface Standard, Version 1.1"*, June 1995.

[20] The ATM Forum. "Audiovisual Multimedia Services: Video no Demand Specification 1.0". Technical report, ATM Forum Technical Committee, December 1995.

[21] D. Le Gall. "MPEG: A Video Compression Standard for Multimedia Applications". *Communications of the ACM*, 34(4):46–58, April 1991.

[22] M. Garrett and W. Willinger. "Analysis, Modeling and Generation of Self-Similar VBR Video Traffic". In *ACM SIGCOMM*, pages 269–280, London, England UK, August 1994.

[23] G.A. Geist and V.S. Sunderam. "Network-Based Concurrent Computing on the PVM System". *Concurrency: Practice and Experience*, 4(4):293–311, June 1992.

[24] M. Grossglauser, S. Keshav, and D. Tse. "RCBR: A Simple and Efficient Service for Multiple Time-Scale Traffic". In *Proc. ACM SIGCOMM*, Cambridge, MA, 1995.

[25] Serial HIPPI Implementors Group. *"Serial HIPPI Specification, Revision 1.0"*, May 1991.

[26] Jenwei Hsieh, David H.C. Du, Norman J. Troullier, and Mengjou Lin. "Enhanced PVM Communications over HIPPI Networks". In *Proceedings of The Second International Workshop on High-Speed Network Computing (HiNet '96)*, Honolulu, April 1996.

[27] J.P. Hughes and W.R. Franta. "Geographical Extension of HIPPI Channels via High Speed SONET". *IEEE Network*, pages 42–53, May/June 1994.

[28] IBM. *"IBM ATM 155+ SAR Module, Functional Description: CHARM 1.5"*. IBM, October 1996.

[29] NetStar Inc. *"GigaRouter System Description, Revision 2"*. NetStar Inc., Dec. 1994.

[30] Silicon Graphics Inc. *"IRIS HIPPI API Programmer's Guide"*. Silicon Graphics Inc., April 1994.

[31] Tandem Computer Inc. "ServerNet". *Enterprise Systems Journal*, June 1995.

[32] V. Jacobson, R. Braden, and D. Borman. *"TCP Extensions for High Performance, Request for Comment (RFC) 1323"*, May 1992.

[33] R.A. Jarvis. "On the Identification of the Convex Hull of a Finite Set of Points in the Plane". *Information Processing Letters*, 2:18–21, 2 1973.

[34] H. Kanakia, P.P. Mishra, and R. Reibman. "An Adaptive Congestion Control Scheme for Real Time Packet Video Transport". *IEEE/ACM Transactions on Networking*, 3(6):671–682, December 1995.

[35] G. Karlsson. "Asynchronous Transfer of Video". *IEEE Communications Magazine*, pages 118–126, August 1996.

[36] M. Kawarasaki and B. Jabbari. "B-ISDN Architecture and Protocol". *IEEE Journal on Selected Areas in Communications*, 9(9):1405–1415, Dec. 1991.

[37] A. Kolawa. "The Express Programming Environment". In *Workshop on Heterogeneous Network-Based Concurrent Computing*, Tallahassee, Oct. 1991.

[38] Los Alamos National Laboratory. "High Performance Networking at Long Distances". http://juggler.lanl.gov/lanp/gateway-info.html.

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

[40] M. Lin, J. Hsieh, D.H.C. Du, and J. MacDonald. "Performance of High-Speed Network I/O Subsystems: Case Study of a Fibre Channel Network". In *IEEE Proceedings of Supercomputing 1994*, Washington D.C., Nov. 1994.

[41] M. Lin, J. Hsieh, D.H.C. Du, J. Thomas, and J. MacDonald. "Distributed Network Computing Over Local ATM Networks". *IEEE Journal on Selected Areas in Communications*, 13(4):733–748, May 1995.

[42] R.J. Manchek. "Design and Implementation of PVM Version 3". Master thesis, University of Tennessee, Knoxville, May 1994.

[43] J.M. McManus and K.W. Ross. "Pre-recorded VBR Sources in ATM Networks: Piecewise-Constant-Rate Transmission and Transport". Technical report, Department of Computer Systems Engineering, University of Pennsylvania, September 1995.

[44] J.M. McManus and K.W. Ross. "Video-on-Demand Over ATM: Constant-Rate Transmission and Transport". *IEEE Journal on Selected Areas in Communications*, 14(6):1087–1098, August 1996.

[45] Sun Microsystems. *"Network Programming Guide"*, March 1990.

[46] Hewlett-Packard France Rami el Sebeiti. "Personal communications", 1995.

[47] A.R. Reibman and Berger A.W. "Traffic Descriptors for VBR Video Teleconferencing Over ATM Networks". *IEEE/ACM Transactions on Networking*, 3(3):329–339, June 1995.

[48] D.J. Reininger, D. Raychaudhuri, and J.Y. Hui. "Bandwidth Renegotiation for VBR Video Over ATM Networks". *IEEE Journal on Selected Areas in Communications*, 14(6):1076–1086, August 1996.

[49] J. Renwick and A. Nicholson. *"IP and ARP on HIPPI, Request for Comment (RFC) 1374"*, Oct. 1992.

[50] O. Rose. "Statistical Properties of MPEG Video Traffic and Their Impact on Traffic Modeling in ATM Systems". In *Proc. of the 20th Annual Conference on Local Computer Networks*, Minneapolis, MN, 1995.

[51] J.D. Salehi, Z.L. Zhang, J. Kurose, and D. Towsley. "Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing". In *Proc. ACM SIGMETRICS*, Philadelphia, PA, May 1996.

[52] M. Snir, P. Hochschild, D.D. Frye, and K.J. Gildea. "The Communication Software and Parallel Environment of the IBM SP2". *IBM Systems Journal*, 34(2):205–221, Feb. 1995.

[53] IEEE Computer Society. *"IEEE Standard for Scalable Coherent Interface (SCI)"*, 1993.

[54] C.A. Thekkath, H.M. Levy, and E.D.Lazowska. "Separating Data and Control Transfer in Distributed Operating Systems". In *Proc. of the 6th International Conference on Architectural Support for Programming Language and Operating Systems*, October 1994.

[55] D. Tolmie and J. Renwick. "HIPPI: Simplicity Yields Success". *IEEE Network*, page 28, Jan. 1993.

[56] T. von Eicken, A. Basu, V. Buch, and W. Vogels. "U-Net: A User-Level Network Interface for Parallel and Distributed Computing". In *Proc. of the 15th ACM Symposium on Operating Systems Principles*, Dec. 1995.

[57] T. von Eicken, D.E. Culler, S. Goldstein, and K. Schauser. "Active Messages: a Mechanism for Integrated Communication and Computation". In *Proc. of the 19th Annual International Symposium on Computer Architecture*, pages 256–67, May 1992.

[58] Gregory K. Wallace. "The JPEG Still Picture Compression Standard". *Communications of the ACM*, 34(4):31–44, April 1991.

[59] Elisabeth Wechsler. "Industry Harnesses Clustered Workstations To Squeeze Extra Cycles". *NAS News*, 2(9), March/April 1995.

[60] ANSI X3.183-1991. "High-Performance Parallel Interface: Mechanical, Electrical, and Signaling Protocol Specification (HIPPI-PH)". Technical report, American National Standard Institute, Inc., June 1991.

[61] ANSI X3.210-1992. "High-Performance Parallel Interface: Framing Protocol (HIPPI-FP)". Technical report, American National Standard Institute, Inc., Feb. 1992.

[62] ANSI X3.218-1993. "High-Performance Parallel Interface: Encapsulation of ISO 8802-2 (IEEE Std 802.2) Logical Link Control Protocol Data Unit (802.2 Link Encapsulation), (HIPPI-LE)". Technical report, American National Standard Institute, Inc., June 1993.

[63] ANSI X3.222-1993. "High-Performance Parallel Interface: Physical Switch Control (HIPPI-SC)". Technical report, American National Standard Institute, Inc., July 1993.

[64] ANSI X3.230-1994. "Fibre Channel - Physical and Signaling Interface (FC-PH)". Technical report, American National Standard Institute, Inc., 1994.

[65] ANSI X3T11. "High-Performance Parallel Interface - Mapping to Asynchronous Transfer Mode". Technical report, American National Standard Institute, Inc., Feb. 1995.

[66] K.C. Young, C.A. Johnston, D.J. Smith, J.W. Mann, J.J. DesMarais, M.Z. Iqbal, J.C. Young, K.A. Walsh, and W.S. Holden. "A HIPPI/ATM/SONET Network Interface for the Nectar Gigabit Testbed". In *LEOS 1993 Summer Topical Meeting Digest on Gigabit Networks*, page 14. IEEE Lasers and Electro-Optics Society, July 1993.