Quality of Service and Asynchronous Transfer Mode in IP Internetworks

by
Bruce Albert Mah

B.S. (University of California at Berkeley) 1991
M.S. (University of California at Berkeley) 1993

A dissertation submitted in partial satisfaction of the requirements for the degree of
Doctor of Philosophy
in
Computer Science
in the
GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Domenico Ferrari, Chair
Professor Randy H. Katz
Professor Ronald Wolff

Fall 1996

The dissertation of Bruce Albert Mah is approved:

_____

Chair Date

_____

Date

_____

Date

University of California at Berkeley

1996

Quality of Service and Asynchronous Transfer Mode in IP Internetworks

Abstract

Quality of Service and Asynchronous Transfer Mode in IP Internetworks

by

Bruce Albert Mah

Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Domenico Ferrari, Chair

The deployment of Asynchronous Transfer Mode (ATM) networks is a recent development in the field of computer communication. When we attempt to use these networks as a part of the global Internet, running the Internet Protocol (IP), we see a number of differences between the data forwarding models of ATM (virtual circuits supporting performance guarantees) and IP (datagrams, usually best-effort). In our research, we have evaluated different policies for IP-over-ATM networks to bridge the gaps between these two networks and to make them function more efficiently together.

The differences between IP and ATM raise three issues. First is the question of how Internet applications can take advantage of ATM quality of service facilities, without support from other portions of the Internet. A second issue is that of determining which IP conversations should be multiplexed onto a single ATM virtual circuit. Last is the problem of virtual circuit management, which determines when ATM connections should be established and torn down.

We have examined different quality of service, multiplexing, and virtual circuit management policies, and evaluated their relative merits from the standpoint of the performance of typical Internet applications. Our evaluation used a simulation of a large IP internetwork with a wide-area ATM backbone and a synthetic workload modeling the traffic generated by common Internet applications. For this purpose, we implemented a new network simulator, the Internet Simulated ATM Networking Environment (INSANE).

Our results show that the use of different scheduling algorithms and QOS parameters can be used to express preference for certain applications, although some care must be taken to avoid starvation effects. The use of jitter controlling schedulers in the ATM network can be efficacious in reducing packet loss in long TCP bulk transfers. We see that multiplexing can improve application performance due to a reduced need to set up ATM virtual circuits, although interactions with some network service disciplines can negate these effects. Finally, we show that caching idle virtual circuits for reuse is, in general, beneficial for both network and application performance.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would first like to thank my advisor, Professor Domenico Ferrari, for his guidance and encouragement through the past seven years (and three degrees!). I am truly fortunate to have been one of his students; he gave me the intellectual freedom to define the path of my research and the support to help me see this work to completion. I also appreciate the extra effort required for him to advise and assist me remotely, from half-way around the world.

Professor Randy H. Katz served on my thesis committee and was the chair of my qualifying exam committee. He was also my local "sounding board" for ideas, and I have greatly benefitted from his wisdom and advice.

Finally, thanks are due to Professor Ronald W. Wolff, for serving on my thesis committee (again, across an ocean!) and qualifying exam committee. Also, my gratitude to Professor Jean Walrand, for serving on my qualifying exam committee.

I am also indebted to Elan Amir, Kimberly Keeton, and Venkata N. Padmadabhan for their helpful comments on various portions of this work.

Also important were the valuable insights and inputs from the members of the Tenet Group at the University of California at Berkeley. In particular, Anindo Banerjea, Ramón Cáceres, Riccardo Gusella, Mark Moran, Dinesh Verma, and Hui Zhang helped to initiate me into the world of networking research. Thanks to S. Keshav and Steve McCanne for serving as (perhaps unknowing) inspirations to try for higher standards of excellence. I have benefitted from many conversations with Amit Gupta, Wendy Heffner, Edward Knightly, and Colin Parris.

I'd like to acknowledge the assistance of the members of the XUNET II project at AT&T Bell Laboratories. In particular, thanks to Sandy Fraser and Ravi Sethi for their commit-

ments to the XUNET summer program, to Chuck Kalmanek for his advice and encouragement, and to S. Keshav and Pat Parseghian for their patience and for sharing their technical expertise.

Funding for this research was provided by AT&T Bell Labs, Corporation for National Research Initiatives, the United States Department of Energy (DE-FG03-92ER-25135), Digital Equipment Corporation, Hitachi, International Computer Science Institute, Mitsubishi Electric Research Laboratories, the National Science Foundation and the Advanced Research Projects Agency (NCR-8919038), and Pacific Bell. I also wish to thank the National Science Foundation for their support in their form of an NSF Graduate Fellowship.

The network simulations described in this work were performed using computers belonging to the Network of Workstations (NOW) project at the University of California at Berkeley. I am grateful to this group for the use of their resources, and especially to Eric Fraser, Douglas Ghormley, and Amin Vahdat for their technical assistance.

On a more personal note, I'd like to thank all of my friends for their encouragement and advice (as well as their tolerance!), especially Hari Balakrishnan, Judy Chan, Kimberly Keeton, Jeanna Neefe Matthews, Venkata N. Padmanabhan, and Srinivasan Seshan.

Last (but certainly not least), thanks to my parents, Albert and Phyllis Mah, for all their support, especially during my college and graduate career.

1 Introduction

1.1. Motivation

This research brings together two very different networking “worlds”. One is that of Asynchronous Transfer Mode (ATM) networks. This technology is designed to carry a wide variety of audio, video, and data traffic, and is presently growing in popularity. Network users are turning to ATM as a solution for applications that require high bandwidth or guaranteed performance.

The other world is that of the nearly-ubiquitous global Internet. The explosive growth of the Internet, both in terms of its logical size and the amount of traffic it carries, is well-known. More than a single network, it is a heterogeneous collection of networks incorporating a wide range of technologies from dial-up modems to high-speed fiber-optic links.

An important, practical problem is to make IP and ATM networks function together in an efficient and effective manner, which draws on the strengths of both. However, this goal raises several technical challenges, brought about by basic differences in the architectures of these two types of networking systems.

There are three basic differences that create problems when we try to make these two dissimilar types of networks interoperate. The first arises from the different data forwarding models of IP (connectionless) versus that of ATM (connection-oriented). Because the Internet has no connections in its network layer, ATM portions of the Internet need to infer the appropriate times to establish and close connections.

Another difference is the types of quality of service support provided by the networks. ATM allows users to specify the quality of service (for example, a minimum throughput or a maximum delay) they desire from the network. An ATM service can guarantee per-

formance for all data sent on a connection, until the connection is closed. By contrast, the Internet has no support for end-to-end quality of service; all packets are treated equally. Although efforts are underway to provide differential treatment for different types of traffic in the Internet, this support is not widespread, let alone ubiquitous.

A final difference is the nature of packets supported by the two networks. In ATM, packets are small and fixed-size. A segmentation and reassembly protocol needs to be used in order to support the larger, variable-sized packets that are normally assumed by the Internet protocol stack.

These differences motivate three critical design issues for the integration of IP and ATM, issues that are the focus of this work. First, Internet applications should be able to take advantage of the quality of service features of an ATM subnet, even though surrounding portions of the Internet may not be able to provide such support.

Another issue is that of multiplexing. Some efficient policy is required to select the packets that share a given ATM virtual circuit. These packets could all belong to the same IP conversation, or parts of different ones.

The final issue relates to the management of virtual circuits. Because IP has no connections at its network layer, some set of policies and mechanisms are required to infer appropriate times for ATM virtual circuits to be established and closed.

1.2. Dissertation Overview

In this dissertation, we investigate the effects of different policies for using Asynchronous Transfer Mode (ATM) networks to carry packets for the Internet Protocol (IP). We focus on evaluating alternatives for these policies with respect to the end-to-end performance that they deliver to applications commonly seen on the Internet.

In Chapter 2, we present some background on both the IP and ATM protocol stacks. We also provide an introduction to the three IP-over-ATM design issues we investigated in this study.

Chapter 3 discusses the methodology we used to evaluate different alternatives in IP-over-ATM designs. We include a description of a new network simulator, the Internet Simulated ATM Networking Environment (INSANE), and show how we used it to measure the performance of several simulated Internet applications in a heterogeneous network of more than a thousand hosts.

We present of the use of different IP-over-ATM quality of service policies in Chapter 4, along with the implications for end-to-end Internet application performance. We examined four different ATM service disciplines and experimented with a variety of policies to use these mechanisms for giving preference to different types of applications. We show that a static priority scheme (if used carefully) can be effective providing differential treatment to different Internet applications. The policing provided by performance-guaranteed ATM services can prevent bulk transfers from monopolizing the network, however the subsequent benefits derived by other applications are uncertain.

In Chapter 5, we discuss the impact of three IP-over-ATM multiplexing policies. These policies vary the granularity with which IP packets are mapped onto virtual circuits in an ATM network. We see that aggregating several IP conversations onto a common virtual circuit improves the performance of short transfers. Longer conversations may fare better if carried by their own virtual circuits, although this result is dependent on the scheduling policy used over the ATM network.

Chapter 6 shows the effects of different policies for managing virtual circuits in an IP-over-ATM network. We look, in particular, at three different policies for setting up and tearing down virtual circuits being used to carry Internet traffic. We show that the performance of applications using switched virtual circuits can be improved by caching idle connections for reuse by subsequent conversations.

Finally, in Chapter 7, we present our conclusions and some thoughts for future work in this area.

Our network simulations relied on having reasonably accurate models of the network activity generated by common Internet applications. In two cases, we needed to develop

new models, because existing ones were either non-existent or not applicable. We discuss this work in two appendices. In Appendix A, we present an empirical model of traffic generated by the HyperText Transport Protocol (HTTP), as used by World Wide Web applications.

Appendix B describes a similar model for Internet video traffic, based on a trace taken from the MBONE video applications `vic` and `vgw`.

2 Background

Using IP and ATM together presents some interesting challenges because they differ in fundamental ways, from their respective models of data forwarding (connectionless vs. connection-oriented) to support for the preferential treatment of packets (no support vs. the potential for hard guarantees). This chapter provides background information on these two different types of networks, and some of the issues that are involved when attempting to make them interoperate.

In Section 2.1, we present a brief background of the Internet Protocol. Asynchronous Transfer Mode networks are described in Section 2.2. Section 2.3 describes some work in the provision and use of quality of service in packet-switching networks such as ATM. In Section 2.4, we describe some current approaches to running IP over ATM networks. In Section 2.5, we discuss some remaining, fundamental problems and show how they motivate the issues that are the subjects of our research. Finally, in Section 2.6, we present some design alternatives for solutions to these problems.

2.1 Internet Protocol

In the current Internet, the network layer protocol used for forwarding data across and between subnets is the *Internet Protocol* (IP) [Postel81a]. IP is a connectionless protocol; packets (also referred to as *datagrams*) are sent and received independently of each other. This approach allows a host to send packets to a destination without first needing to set up state information in intermediate routers along the path to be taken by data.

IP is almost entirely independent of the technology used to transport packets. It makes few assumptions about the nature of individual subnets used to forward data. IP packets can traverse many subnets without either the senders or receivers being aware of the details or

types of the networks encountered along the path between them. Although IP generally does not deliberately drop (discard) packets, no attempt is made to provide reliability (in other words, to guarantee that data sent will be correctly received).[1]

Internet applications typically do not use IP directly. Rather, they employ higher-layer protocols, which provide services more appropriate to the needs of the applications. A commonly-used transport protocol, layered on top of IP, is the *Transport Control Protocol* (TCP) [Postel81b]. TCP provides reliable, in-order delivery of streams of bytes. It is connection-oriented; thus the endpoints of a TCP conversation must handshake to establish connection state before any data packets are sent. Because TCP is implemented using IP's unreliable datagram service, it uses acknowledgments, timeouts, and retransmissions to ensure that data is correctly received by the destination. TCP also implements algorithms for flow control and congestion control [Jacobson88].

The other transport protocol commonly used with IP is the *User Datagram Protocol* (UDP) [Postel80]. UDP furnishes a service very similar to that of "raw" IP; it provides the unreliable delivery of datagrams to user programs. Applications using UDP typically either implement their own reliability protocol (for example, as done in Sun RPC [Sun Microsystems88]) or do not require reliable transport of data (as in the case of typical applications transporting audio and video).

TCP, UDP, and IP, together with other higher-layer protocols not discussed here, collectively form the Internet protocol suite, pictured in Figure 2-1. Additional details on the Internet protocol suite can be found in [Stevens94].

The Internet Engineering Task Force is currently designing a successor to IP, known as *IP Version 6* or *IPv6*. IPv6 is a network-layer protocol which addresses the primary limitations of IP, while retaining much of the same basic protocol architecture [Deering96]. Among the new features of IPv6 are an expanded address space (128-bit addresses vs. 32-bit IP addresses), ease of route aggregation for scalability, a redesigned packet header for effi-

1. One instance of a situation in which packets are deliberately dropped is a firewall, which enhances network security by restricting the packets that can be forwarded to or from a network.

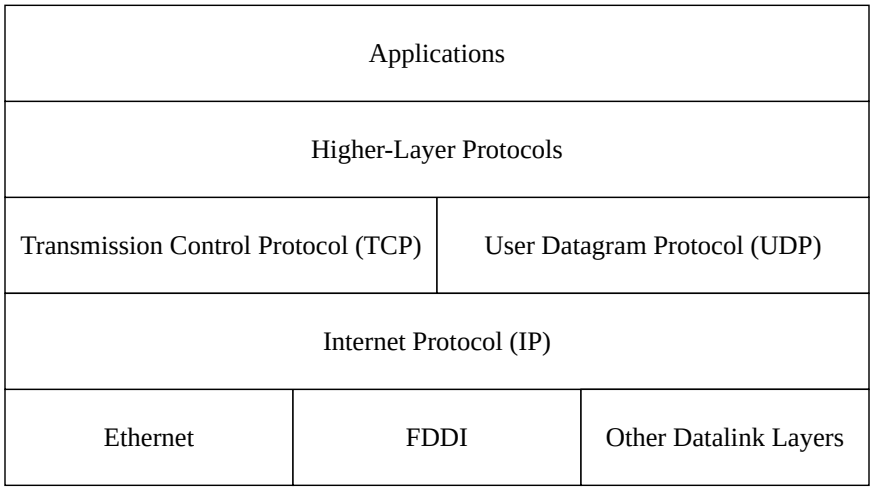


Figure 2-1. The Internet Protocol Suite and Datalink Layers.

cient packet processing, and explicit support for security and authentication. In the context of IPv6, the original Internet Protocol is referred to as *IP version 4*, or *IPv4*. For the most part, IPv6 is similar enough to IPv4 that most of this work (which deals with IPv4) is applicable to the newer protocol as well.

2.2 Asynchronous Transfer Mode

Asynchronous Transfer Mode (ATM) is a new network technology designed for "integrated services" networks capable of carrying multimedia data as well as conventional computer data traffic [ATM Forum95]. ATM is a connection-oriented service that transfers small, fixed-sized packets called *cells* through a switch-based network. Although it makes no promises of reliable delivery, cells that are actually delivered are guaranteed to be in-order.

ATM's small, fixed-size cells (48 payload bytes) are hardly suitable for network protocols such as IP, which are designed to use larger, variable-sized frames. To address the requirements of such network applications, an *ATM Adaptation Layer* (AAL) protocol fragments larger, variable-sized packets into cells for transmission and reassemble them upon arrival at their destinations. The AAL typically used to transport IP packets is known as *AAL 5* [Heinanen93]. (Other AAL protocols serve a variety of purposes, such as timing and syn-

chronization of continuous media data. Not all of them provide fragmentation and reassembly.)

Because ATM is connection-oriented, it requires a signalling protocol to set up the forwarding tables in the network switches along the path to be taken by data. ATM signalling also needs to reserve network resources for guaranteed-performance connections. In ATM User-Network Interface (UNI) 3.1-compliant networks, this connection establishment is done using the *Q.2931* signalling protocol. Although ATM networks do not provide reliable delivery of data, their signalling protocols can be greatly simplified if they can be assured of reliable transport of signalling messages. In the UNI 3.1 standard, this functionality is performed by a specialized *Signalling ATM Adaptation Layer* (SAAL) [ATM Forum95].

ATM uses a Virtual Circuit Identifier (VCID) in the ATM cell header to identify the connection to which each cell belongs. VCIDs are local to a link; thus, the cells for a given connection may have different VCIDs as they traverse the ATM network. Assignment of VCIDs to a connection is a part of the setup done by ATM signalling.

The ATM protocol stack according to [ATM Forum95] is shown in Figure 2-2. We note that ATM networks not complying with this standard will have a different (but probably similar) protocol stack. For example, the XUNET II experimental ATM testbed network described in [Fraser92] implements its own adaptation layer, similar to AAL 5. Its signalling is provided by a proprietary signalling protocol, which incorporates its own reliability functionality.

With the appropriate scheduling disciplines in the network switches and support in the signalling software, ATM networks have the potential to provide real-time[2] performance guarantees, such as bounds on bandwidth and packet loss. These performance guarantees are likely to be necessary for many network applications, such as digital audio and video [Ferrari90]. The ATM Forum has defined a number of traffic classes, performance param-

2. By “real-time”, we refer to a service that can provide mathematically provable bounds or guarantees on performance.

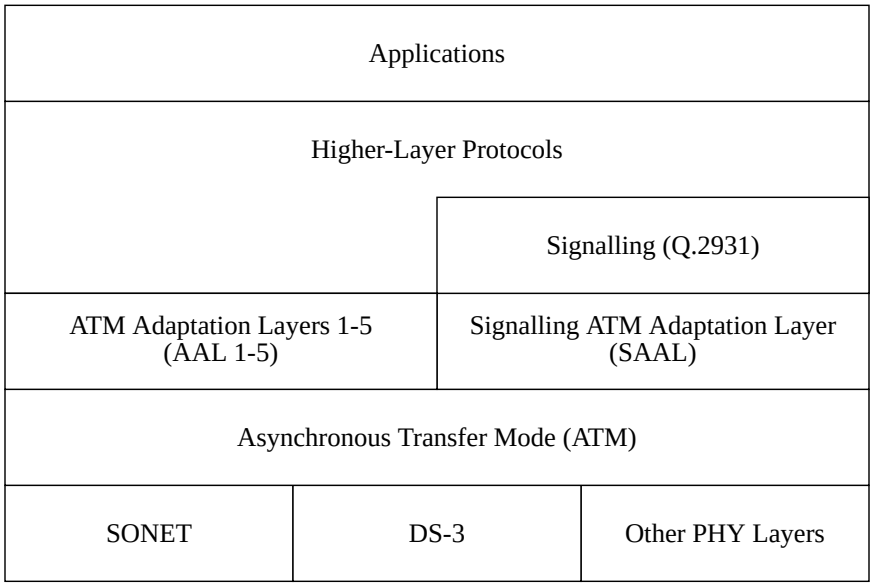


Figure 2-2. ATM Protocol Stack and Physical Layers.

eters, and interfaces for the support of different qualities of service. Unfortunately, no set of specific algorithms for supporting these services has been defined.

Therefore, we chose to study a hypothetical ATM networking environment that does not conform to ATM Forum standards, but has a set of well-known algorithms for providing performance guarantees. We rely only on the following abstract properties of ATM networks:

- ATM is a switch-based network, with point-to-point links between switches. By contrast, many popular LANs such as Ethernet are multiple-access, shared-media networks.

- ATM is connection-oriented (as opposed to datagram-based). There is some signalling protocol capable of setting up virtual circuits and (if necessary) performing the necessary admission control tests. In place of Q.2931 signalling, our simulated network uses a protocol based heavily on the *Real-Time Channel Administration Protocol* (RCAP), the signalling protocol used in the Tenet Real-Time Protocol Suite [Mah93].

- ATM can support performance guarantees. Although we make no assumptions for the mechanism for providing guarantees, we do assume a well-defined interface. Our network uses different versions of Rate-Controlled Static Priority Queueing (RCSP) as scheduling mechanisms (along with appropriate admission control tests) to provide performance guarantees [Zhang93a].

Although a fairly recent innovation, ATM is currently gaining popularity. However, it is uncertain whether or not ATM will become a ubiquitous, dominant network technology. The existing installed base of Local Area Networks (LANs) such as Ethernet is considerable; replacing these networks with ATM will be costly, and in many cases unnecessary. For the foreseeable future, it appears that large-scale connectivity will continue to involve multiple, heterogeneous networks, and appropriate internetwork layer protocols. It seems likely that ATM networks will be used as wide-area backbones, connecting existing, non-ATM LANs.

2.3 Quality of Service in Packet-Switching Networks

The issue of providing quality of service support in packet-switching networks (such as ATM) has received considerable attention. By “quality of service”, we specifically refer to the differential treatment of packets in the network, usually depending on the requirements of the network applications or on administrative constraints. For example, packet video applications need some minimum throughput in order to deliver a usable picture quality. Certain types of remote control applications require a bound on network delay in order to make operation feasible and tolerable for the end user.

Our views on quality-of-service and guaranteed service are based on the Tenet approach to real-time communication in packet-switching networks. The Tenet approach provides strict, mathematically-provable performance guarantees that will hold even under worst-case conditions of network load [Ferrari94]. This method places several requirements on the network. First, the data forwarding entities (e.g. routers and switches) must use suitable queueing and scheduling algorithms when processing network packets. A large class of scheduling disciplines fit this requirement, including Earliest Due Date [Ferrari89], Hierarchical Round Robin [Kalmanek90], and Rate-Controlled Static Priority [Zhang93a].

The second requirement is that network sources must be able to characterize their traffic characteristics (e.g. peak and average sending rate) and their performance requirements (e.g. delay or delay jitter). This information will typically be provided to the network at connection setup time. If a connection is accepted by the network, the connection establishment is treated as a contract between the application and the network, whereby the network agrees to provide the requested performance as long as the application adheres to its traffic characteristics.

Finally, there must be a procedure for performing admission control when connections are established. An admission control procedure determines, based on the current state of allocated network resources, whether or not the network can accept a new connection and still meet all of its promised guarantees. Note that the admission control tests must take into account the worst-case traffic patterns, subject to the sources' traffic characteristics, and ensure that all guarantees will still apply at all times.

The realization of the Tenet approach in this work is based on *Rate-Controlled Static Priority* (RCSP) queueing in the output ports of ATM switches. Conceptually, an RCSP queue consists of two parts, a static priority scheduler and a rate controller (see Figure 2-3). The static priority scheduler buffers cells before they can be sent to the output. Each cell is assigned a priority, based on the connection to which it. Whenever the output link is free, the scheduler transmits the highest-priority cell it has buffered.

The rate controller regulates the flow of cells into the static priority scheduler. Due to variable queueing delays caused by queueing, the spacing of cells along a given connection may become distorted as the cells travel along the path of a connection. This effect, known as *jitter*, requires extra buffering in downstream queues in order to absorb bursts. The purpose of the rate controller is to either completely or partially reconstruct the input traffic pattern before cells are allowed into the scheduler, thus reducing the buffer requirements. This reconstruction is accomplished by delaying selected cells for a connection until those cells conform to the original traffic specifications supplied by the source at connection setup time.

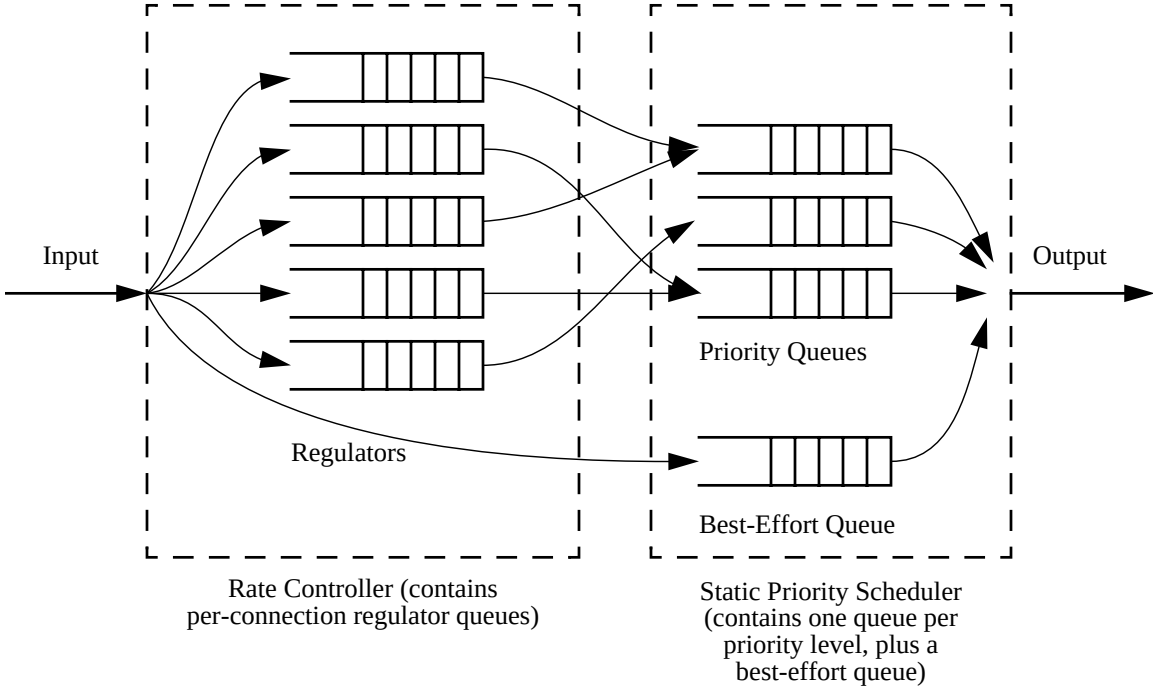
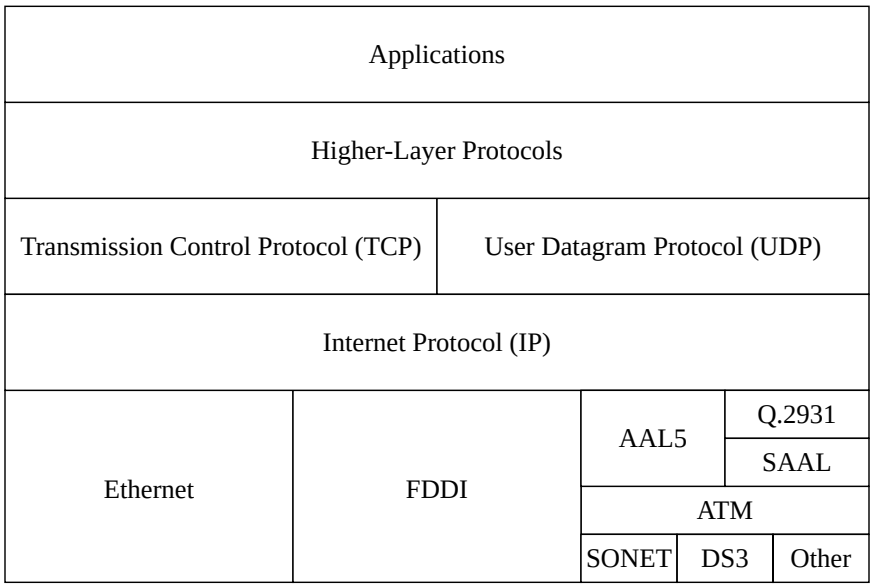


Figure 2-3. Block Diagram of a Rate-Controlled Static Priority Scheduler. The per-connection queues in the rate controller “shape” the traffic so that it corresponds to the source’s traffic specification. Packets are then admitted to the scheduler, which transmits data to the output by priority level.

One potential disadvantage imposed by the use of a rate controller is that the RCSP queue becomes non-work-conserving; in other words, the output link may remain idle, even when there are cells queued in the rate controller. This effect may lead to some unnecessary delays for cells. In our study of different QOS policies, we have experimented with replacing some of the RCSP queues in the ATM network with work-conserving variants.

2.4 Current IP over ATM Approaches

As ATM and the Internet will likely co-exist in the foreseeable future, it is desirable that hosts using either (or both) of these types of networks be able to communicate. One approach to interoperability is for IP to use an ATM network (with an appropriate adaptation layer) as a datalink layer, in the same way that Ethernet and FDDI are commonly used. Conversely, the ATM protocol stack views IP as an application. The use of these two protocol stacks together in this way is commonly referred to as *IP over ATM*. The resulting protocol stack is shown in Figure 2-4.



igure 2-4. IP over ATM Protocol Stack. Different ATM networks will likely have similar adaptation layer and signalling components replacing AAL5, Q.2931, and SAAL in the above protocol stack.

There are several approaches to IP over ATM, each of which defines a slightly different relationship between the ATM network and the IP internetwork. Our research fits well within the framework of three popular models being advanced within the networking community. Although they differ in details, our work treats them almost identically.

The first model is the *Classical Model of IP over ATM* [Laubach94], proposed by the IP over ATM working group within the Internet Engineering Task Force (IETF) [3]. In this paradigm, an ATM network can support one or more IP subnets (referred to as *Logical IP Subnets* or *LISs* in the literature).[4] Hosts and routers belonging to the same subnet can exchange data directly, using virtual circuits to forward IP packets across the ATM network. Two hosts belonging to different subnets (but attached to the same ATM network) can only communicate via a router that is a member of both subnets. Figure 2-5 illustrates the flow of data in this scenario. While classical IP over ATM is potentially inefficient in

3. We note that the IP over ATM and Routing Over Large Clouds working groups merged in May 1996. The combined group, referred to as Internetworking Over Non-Broadcast Multiple Access (ION), continues the work of both of its ancestors [ION96].

4. Scalability and efficiency concerns may make it desirable to divide the hosts attached to an ATM network into several subnets if the ATM network is large.

that a path between ATM-connected hosts may require forwarding through a router, it has the advantage of preserving the original semantics of IP subnets.

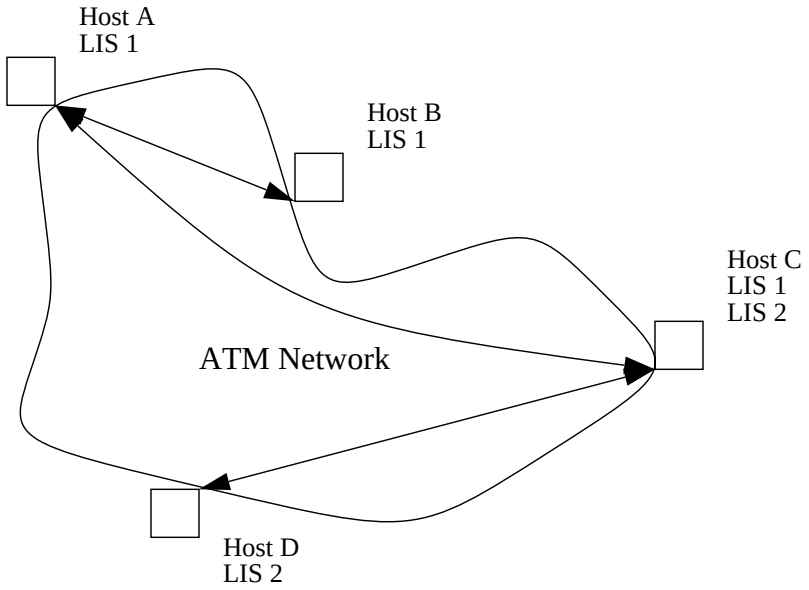


Figure 2-5. Classical IP Model and Logical IP Subnets. Host A can communicate directly ith host B because the two are a part of the same subnet. However, hosts A and D belong to different subnets, and must route their communication through an intermediate router (host C), even though the two are attached to the same ATM network.

Another approach, taken by the *Routing Over Large Clouds* (ROLC) working group of the IETF, seeks to remove the potential inefficiency of the classical model. In the ROLC model, hosts attached to the same ATM network can communicate directly, even if they do not belong to the same LIS. Since part of the original IP routing model dictates that hosts on different subnets must communicate via a router (rather than directly), this method forces changes to the way that IP routing and forwarding is performed. A *Next-Hop Routing Protocol* (NHRP) [Luciani96] communicates the routing information necessary to send data between subnets directly across the ATM network.

A third paradigm, proposed by the ATM Forum, is *LAN Emulation* (commonly abbreviated LANE or LE) [LANE95]. LANE's approach is to make an ATM network appear as a IEEE 802-compliant local-area network. Thus the units of data that are transmitted are

IEEE 802 frames, rather than IP packets. This paradigm is quite similar to Classical IP, but supports multiple network protocols (such as IPX or AppleTalk) in addition to IP.

For simplicity's sake, we chose the Classical IP approach as our model of IP over ATM. However, all three approaches can be viewed identically, for the purposes of this research. Their most important commonality is that ATM-attached hosts transmit IP packets to other hosts across ATM virtual circuits. When the cells making up an IP packet exit the ATM network, the IP packet is reassembled, and forwarding of the packet continues according to standard IP routing (if necessary). Thus, an ATM network is treated as a link layer by IP and the larger Internet.

There exist other schemes for using IP and ATM together, with various degrees of compatibility with the designs explored in this work. For example, various cell-based routers (plus attendant algorithms) have been proposed and described, for example [Parulkar95], [Ipsilon96], and [Newman96c]. These devices are hybrid devices, part IP router and part ATM switch. Essentially, they perform packet forwarding for IP packets along virtual circuits established between adjacent switches. When instructed to, they can establish end-to-end (or hop-by-hop) virtual circuits; packets are then forwarded on a per-cell basis, without the need for IP protocol processing at every intermediate hop.

TCP and UDP over Lightweight IP (TULIP) and *TCP and UDP over a Nonexistent IP Connection* (TUNIC) define new methods for eliminating some or all of the Internet protocol headers across an ATM network [Cole96]. [Newman96a] and [Newman96b] offer similar, more concrete protocols, in the same vein. We believe that our techniques could potentially extend to these environments as well.

IP routers supporting tag switching [Rekhter96] use identifiers similar to ATM VCIDs to streamline packet forwarding. Unlike VCIDs, tags are considered to be merely an optimization (avoiding the IP routing table lookups normally performed for each packet). Another difference is that allocation of tags is intended to be driven by routing changes, rather than the transmission of data. Subject to these differences, the ideas we explore in this work are probably applicable.

2.5 IP-over-ATM Design Issues

The current approaches to IP-over-ATM described in Section 2.4 are primarily concerned with providing basic connectivity. As such, they deal largely with issues such as routing and address resolution, which address the fact that ATM is a non-broadcast, multiple access network. However, they do not address certain design issues arising from basic differences in the nature of IP and ATM networks.

We note three fundamental differences between IP and ATM networks. First are conflicting connection models. IP is connectionless; every packet is sent on an individual basis and is delivered to its destination independently of every other packet. By contrast, ATM is connection-oriented. It requires that network state be set up for a stream of packets before that data can be transmitted.

Another difference is the quality of service models of the two networks. The ATM signalling protocol used to establish connections also allows users to specify the performance they require from the network (for example, a minimum throughput or a maximum loss rate). If the network can support a new connection and its performance requirements, it can guarantee that the set of packets sent on a connection will be given the required quality of service. IP, on the other hand, has no end-to-end quality of service features (the RSVP protocol and integrated services work are intended to address this shortcoming, but such support is not currently widespread).

The third contrast is the nature and size of packets supported by IP and ATM. In ATM networks, cells are small and fixed-sized. IP, by contrast, is designed around medium-sized, variable-length packets.

These three basic differences motivate three design issues, which are the subject of this research. Our first issue deals with extending the QOS features of an ATM network to IP applications. Although IP in its current form has no provision for QOS support, the underlying ATM subnet has the capability to offer performance guarantees. We would like, therefore, for Internet applications to gain some of the benefits of ATM performance guarantees, without end hosts or applications necessarily being aware of this capability.

Another issue concerns the degree of multiplexing to be used on ATM virtual circuits. At one extreme, ATM-attached routers could provide unique virtual circuits for individual network conversations (such as single TCP connections). At the other extreme, they could use virtual circuits as trunks carrying traffic for many conversations passing between a pair of routers. One could easily imagine hybrid schemes as well, which might use dedicated virtual circuits for some traffic and route all other packets via default trunk-like connections.

A third design consideration is that of virtual circuit management. Because IP is connectionless, there is no explicit notification as to when underlying virtual circuits should be set up or torn down. Some heuristic must be used to infer appropriate times for these actions. Different schemes involve using permanent virtual circuits established at network startup time, or switched virtual circuits that are set up on demand and torn down when idle. Specific design choices dictate idle timeout values and whether or not virtual circuits can be cached for reuse by other IP conversations.

ATM-attached hosts and routers implement policies to address each of these issues. To a certain extent, these policies can be implemented (and investigated) separately from each other. Thus three design issues can be viewed as three parameter axes, denoting a space of possible policies. Points within this design space correspond to design policies, made up from a combination of QOS, multiplexing, and virtual circuit management policies. The axes for this space are shown graphically in Figure 2-6.

2.6 IP-over-ATM Policies

In this section, we briefly summarize the alternatives we investigated for QOS, multiplexing, and virtual circuit policies. We then fit them into the framework of the “policy space” described in Section 2.5.

To address the quality-of-service issue, we investigated a number of policies, which used various scheduling disciplines in the ATM network to express precedence for certain Internet applications. We list the various scheduling disciplines in Table 2-1 (more details can

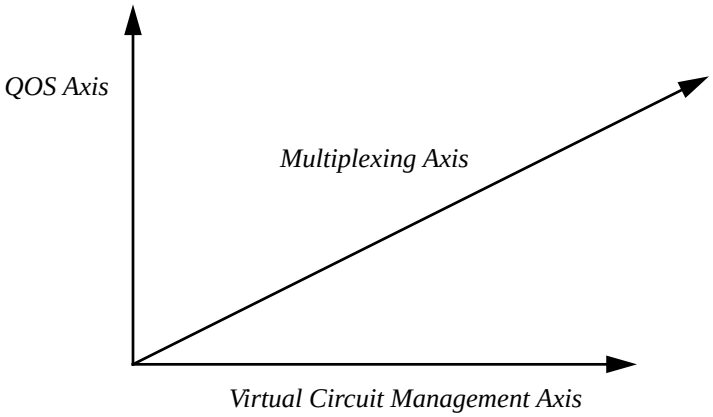


Figure 2-6. IP-over-ATM Policy Space Axes. Each axes represents a set of design alternatives to address a specific design issue. With certain limits, these policies can be varied and investigated independently.

be found in Table 4-5 and Table 4-6). These schedulers control the transmission of cells from the output queues of ATM switches.

Name	Scheduler	Remarks
`noqos`	First-Come-First-Served	A best-effort service only, with all cells treated identically.
`sp`	Static Priority	Simple static priority scheme, no rate control.
`wc`	Work-Conserving Rate-Controlled Static Priority	RCSP variant. Work-conserving refers to the fact that this type of queue will always transmit a cell if one is available.
`nwc`	Non-Work-Conserving Rate-Controlled Static Priority	RCSP variant implementing rate jitter control. Due to jitter control, this scheduler is non-work-conserving.

Table 2-1. Scheduling Disciplines.

Our investigation of multiplexing led us to examine three policies, which perform increasing degrees of aggregation of traffic onto ATM virtual circuits. We summarize them in Table 2-2.

Name	Policy
`conv`	Virtual circuit per IP conversation (e.g. TCP connection or UDP flow)
`app`	Virtual circuit per application type per host pair.
`router`	Virtual circuit per pair of routers, carrying all traffic passing through the pair of routers, regardless of source or destination host.

Table 2-2. Multiplexing Policies.

Finally, we investigated the effects of three different virtual circuit management policies, as summarized in Table 2-3.

Name	Policy
`pvc`	Permanent Virtual Circuits. These connections are never torn down.
`svc`	Switched Virtual Circuits with 10-second static timeouts. Virtual circuits idle for longer than this amount of time are torn down.
`svccache`	Switched Virtual Circuits with 10-second static timeouts. Virtual circuits idle for longer than this amount of time are cached for future re-use. After 300 seconds (5 minutes) of idle time, they are torn down.

Table 2-3. Virtual Circuit Management Policies.

We describe the various IP-over-ATM policies by the names of the QOS, multiplexing, and virtual circuit management policies that make them up. For example, an experimental configuration of `sp-telnet-conv-svccache` indicates a scenario where a static priority scheme was used to give preference to telnet applications, with multiplexing done on a per-conversation basis, and with switched virtual circuits cached for reuse.

The specific instances of the three types of policies form the set of possible values along each of the axes in the IP-over-ATM policy space. Points in this space correspond to specific IP-over-ATM policies that we evaluated in the course of this work. We show them in Figure 2-7.

We observe that these axes are not entirely independent. For example, in an IP-over-ATM service using PVCs, it is impractical to set QOS parameters for an unknown workload traversing a fixed set of virtual circuits. Moreover, the sheer number of virtual circuits required for a complete PVC mesh likely forces a multiplexing policy of one virtual circuit per router pair. Thus, the only PVC policy we consider is a QOS-oblivious (`noqos`), virtual circuit per router (`router`) pair design.

In a similar vein, when performing `router` multiplexing, it is impossible to assign a meaningful QOS to each virtual circuit, because the nature of the aggregate traffic between the routers is unknown. Therefore, all of the policies that use a multiplexing policy of one virtual circuit per router pair do not use any of ATM's QOS features.

We point out certain interesting points in this policy space:

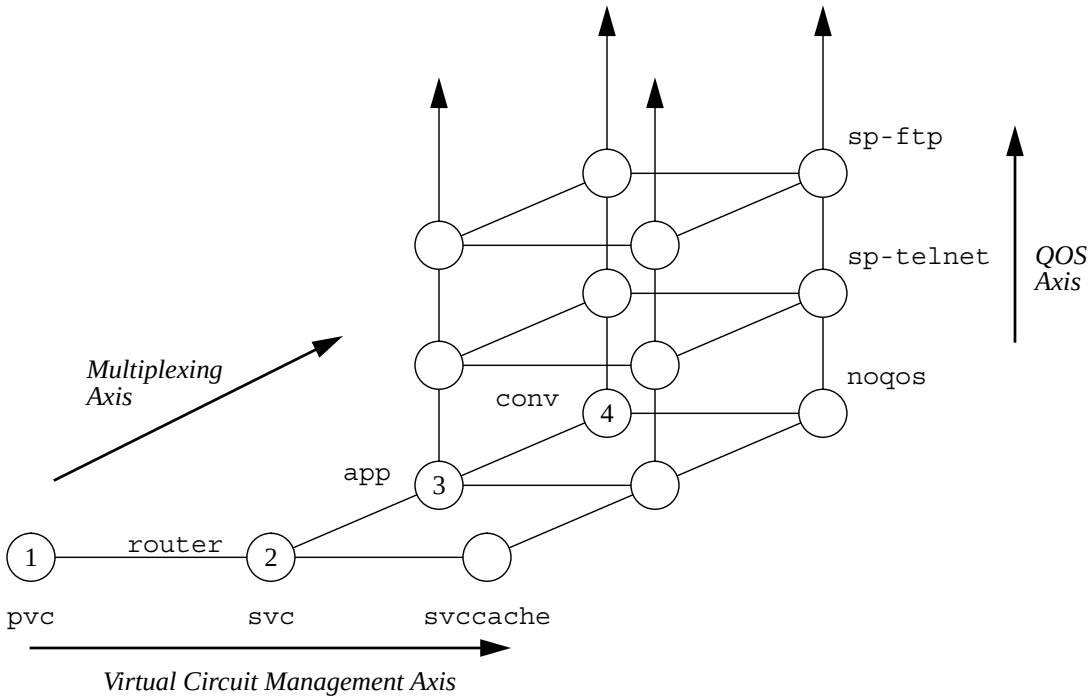


Figure 2-7. IP-over-ATM Policy Space. Circles represent specific design points in this space. The arrows parallel to the QOS axis reflect the fact that this study examined many more QOS policies than could easily be depicted in this diagram.

- The `noqos-*-pvc-router` policy (labeled “1” in Figure 2-7) represents the simplest possible IP-over-ATM design. It was used by XUNET II, a wide-area ATM testbed, in its default configuration [Fraser92]. Other ATM network testbeds (such as BAGNET [Johnston95]) have used this policy, in situations when a lack of interoperability makes providing switched virtual circuits infeasible.

- The `noqos-*-svc-router` policy (denoted by “2” in Figure 2-7) is used by several commercial ATM LANs employing SVCs, including the FORE Systems ATM LAN described in [Biagioni93].

- We have implemented enhancements to the XUNET II IP service to cover various SVC policies (labeled “2”, “3”, and “4”). More details on this implementation can be found in [Mah94a].

There are, of course, other considerations which must be addressed in the design of an IP-over-ATM system. One example is the relationship between IP and ATM routing (addressed in various ways by the various proposals listed in Section 2.4). An investigation of such issues, however, is outside the scope of this work.

3 Methodology

This chapter presents our methodology for evaluating the performance of different IP-over-ATM policies. We used simulations to look at the impact of using various IP-over-ATM policies in a heterogeneous IP internetwork with a wide-area ATM backbone. Specifically, we focused on the operation and performance of common Internet applications running in this environment. By exploring the space of possible IP-over-ATM policies, we were able to analyze the effects of different quality-of-service, multiplexing, and virtual circuit management policies separately, as well as their interactions. Our simulations utilized a new network simulation tool, the Internet Simulated ATM Networking Environment (INSANE).

3.1 Introduction

Fairly early in our experiment planning, we realized that we would need to rely on network simulation for our evaluation. One reason was that there were almost no IP-over-ATM networks available for experimentation that fit our needs. We initially planned to use XUNET II [Fraser92], but it was decommissioned during the course of our investigations. Even if a suitable network were available, implementation artifacts and background network traffic would have introduced uncontrollable factors whose effects might be difficult to isolate.

We briefly considered the use of analytic techniques. However, we felt that the workings and dynamics of a large IP internetwork would be too difficult to analyze in a tractable fashion.

A simulation, by contrast, was ideal for several reasons. First, it allowed us to capture the behavior of hardware, protocols, and applications very closely. We were able to run our experiments in a controlled setting, as we had complete control over the environment.

Finally, because (in many cases) a simulation is a single program on a single computer, testing and analysis was much easier compared to working in a distributed network.

We chose to simulate a wide-area IP internetwork, in which an IP-over-ATM subnet is used as a backbone. Other networking technologies are used at local sites. We felt that this topology was consistent with our assumption that ATM will be used primarily for wide-area, long-haul networks.

Upon this network, we imposed a workload that approximated traffic generated by contemporary Internet applications. Individually, we simulated instances of common applications (such as Web browsers and mail servers) using empirical-based models that mimicked the traffic patterns produced by real programs. Our complete workload was generated by instantiating a number of these applications on hosts throughout the network. The specific applications we examined in this study are listed in Table 3-1.

Application	**Description**	**Traffic Type**
telnet	Remote login	Interactive
FTP	File transfers	Bulk transfers
HTTP	World Wide Web	Bulk transfers, somewhat interactive
audio	Digital audio	Continuous media (constant bitrate)
video	Digital video	Continuous media (variable bitrate)
SMTP	Electronic mail	Bulk transfers (background load only)
NNTP	Network news	Bulk transfers (background load only)

Table 3-1. Internet Applications.

A critical choice in any network evaluation is the set of performance metrics. In our case, we chose to look at application-layer performance. We assumed that changes to the IP-over-ATM backbone would manifest themselves in end-to-end application measures, such as file transfer completion times and packet loss rates. We felt that examining these metrics would be important because they reflected effects visible to end users, unlike more abstract measures such as average queue lengths.

We ran our workload and performed our measurements over a number of configurations, covering a large number of points in the IP-over-ATM policy space discussed in Chapter 2. We were able to investigate the effects of individual types of policies (for example, varying

multiplexing policies) by isolating their effects on the various application performance metrics. For example, to see the effects of different multiplexing policies, we would compare the performance of setups differing only in their multiplexing policies.

As a mechanism to perform our experiments, we designed and built the Internet Simulated ATM Network Environment (INSANE). It contains models of many entities encountered in the environment discussed above, simulating functionality from ATM cell transport to TCP dynamics and application-layer workloads. It is well-adapted to investigating performance in large networks.

In Section 3.2, we show the network environment we used for our evaluation. We describe the workload we imposed on our network in Section 3.3. Section 3.4 discusses our evaluation criteria and Section 3.5 discusses the actual experiments in some detail. In Section 3.6, we present some details on INSANE, the network simulator we constructed for this study. Finally, in Section 3.7, we present a few notes on our experiences with running and using the INSANE simulator.

3.2 Network Topology

The topology of our simulated network is loosely based on that of XUNET II, a wide-area ATM network testbed largely sponsored by AT&T Bell Laboratories [Fraser92]. XUNET II connected a number of universities and research laboratories in the continental United States from 1990–1996. At its peak, its backbone consisted of DS3 (45 Mbps) and 622 Mbps links between four universities and four government and industrial research laboratories. Routers at each of the sites forwarded IP packets between the ATM backbone and local subnets. XUNET II's physical topology is shown in Figure 3-1.

Our simulated ATM backbone connected six local area networks, representing six of the eight XUNET II sites. The main difference between our backbone and that of XUNET II was that we constructed the former to have exactly one bottleneck link. Figure 3-2 shows the configuration of the ATM backbone network used for our experiments.

Each of the six campus sites consisted of two servers and two hundred workstations connected to an idealized shared-media 100 Mbps LAN. The servers represented FTP and

Figure 3-1. XUNET II Backbone Topology.

WWW servers seen on the Internet today. The workstations were assumed to be desktop machines, each primarily used by a single person. The servers and workstations could communicate directly using the LAN for local communication, or to machines at other sites through a router, also attached to the LAN. Figure 3-3 shows the configuration of a typical site.

The simulated wide-area ATM links each had a bandwidth of 1.5 Mbps, the same line rate provided by T1 circuits. The delays along these links were representative of cross-country links; the one-way delay between the switches on the long-distance bottleneck link was 30 ms while the one-way latencies between all other pairs of adjacent switches were each 5 ms. Although higher-speed links are in common use today, we feel that our results would likely scale up to faster link speeds and larger amounts of aggregate traffic.

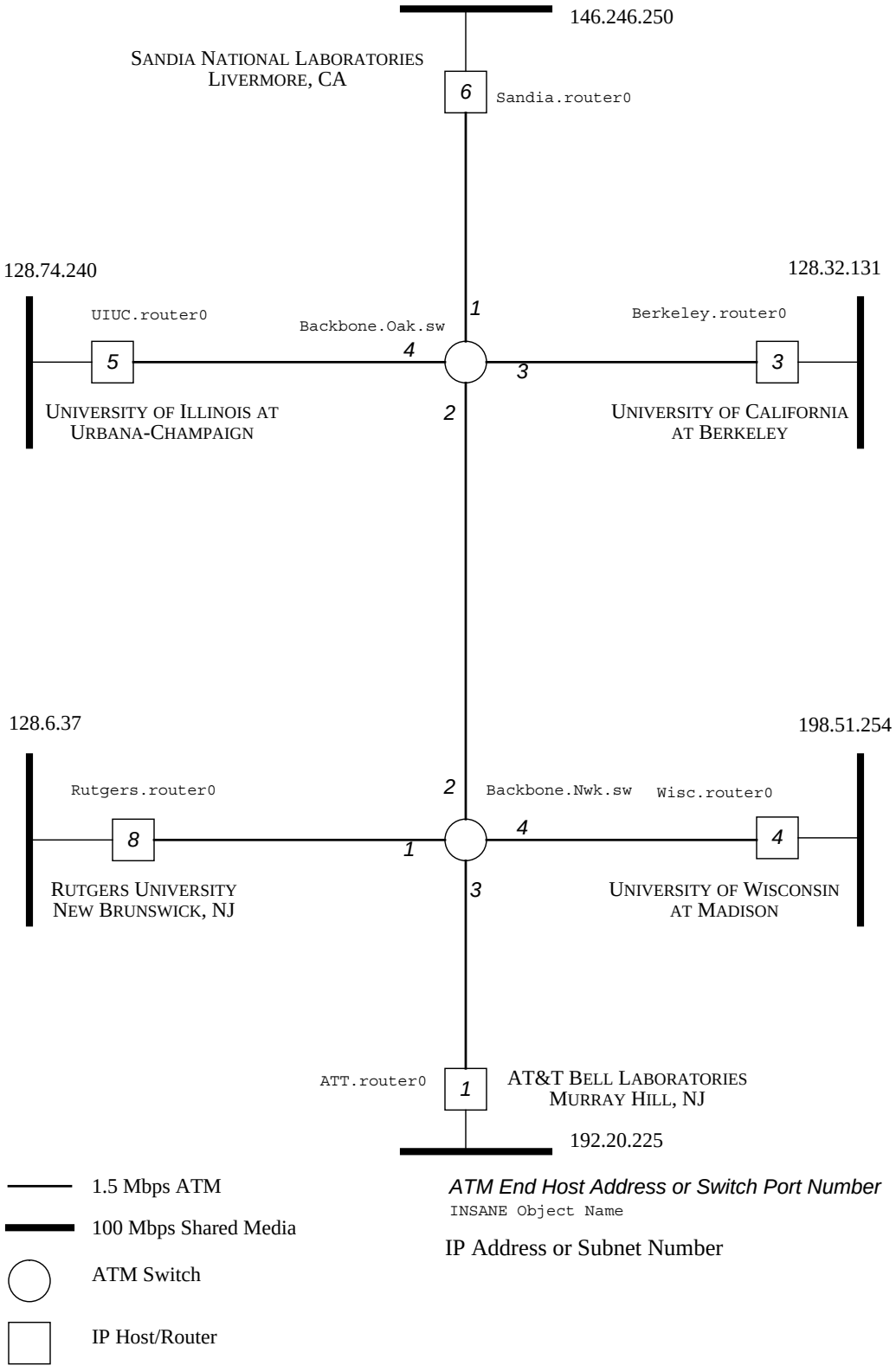


Figure 3-2. Logical Topology of Simulated ATM Backbone Network.

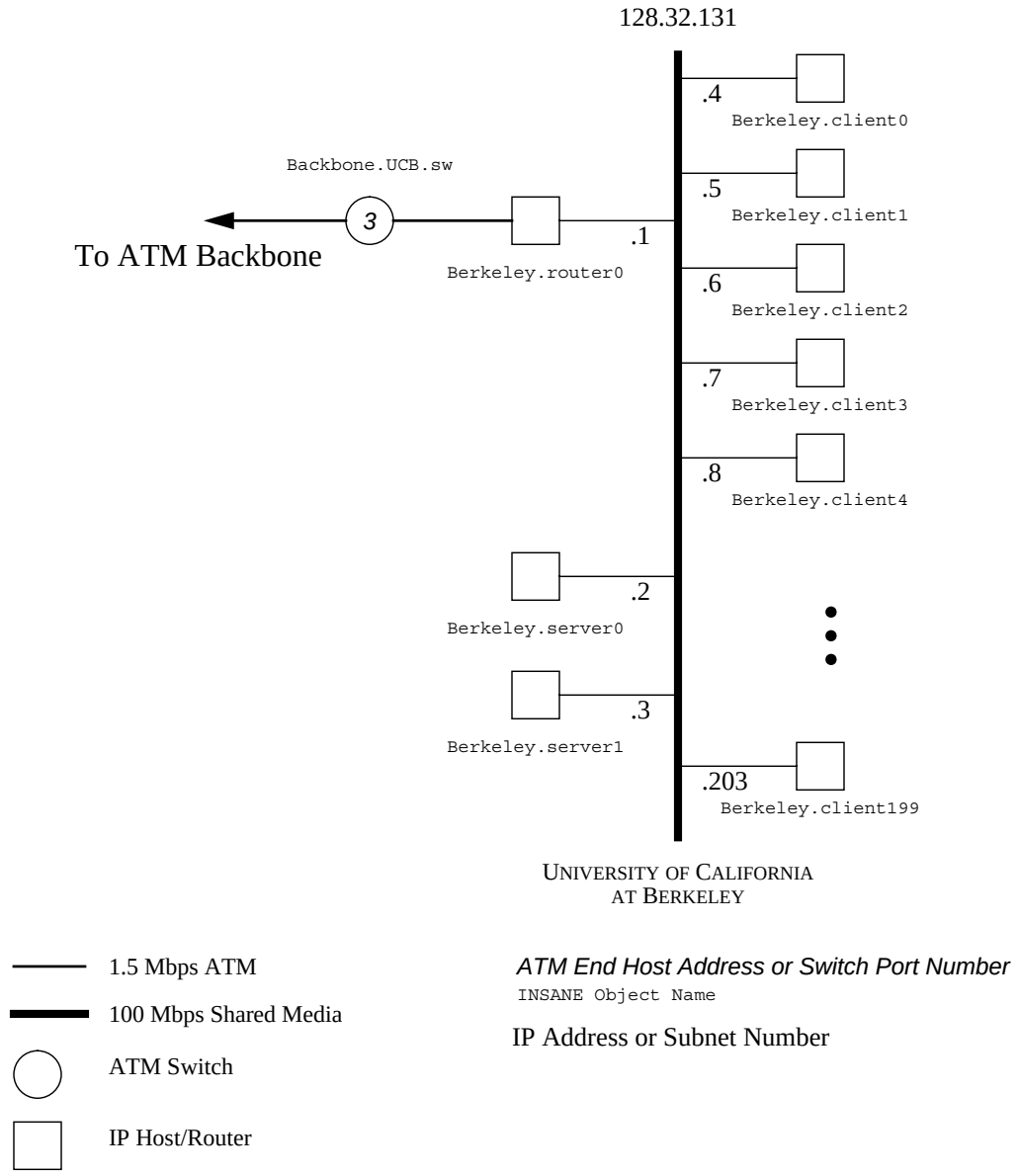


Figure 3-3. Logical Topology of a Typical Local Site.

3.3 Workload

We designed the network workload to approximate the traffic produced by contemporary (1996) Internet applications. To do this, we simulated the network activity of several common Internet applications, and placed instances of these applications onto various hosts in the network during the course of a simulation run. In this section, we describe the

various simulated applications provided by INSANE, as well as the composite workload created out of instances of the individual applications.[1]

3.3.1 Telnet

The *telnet* application provides a means for users to login to computers across an IP network [Postel83]. Part of the traffic generated by this application consists of single bytes (a user's keystrokes) sent in one direction along a TCP connection. The remote computer responds to these data with replies of various lengths (a keystroke echo generates only a single byte in return, but the user typing the final key of a UNIX shell command may generate a "reply" consisting of many bytes of data).

The behavior of the simulated telnet application is controlled by the empirical probability distributions of `tcplib`, a set of traffic models designed for use in network simulators [Danzig91]. For the telnet application, these distributions specify the interarrival times of keystrokes, the size of responses, and the total duration of conversations. Although another remote login application (`rlogin` [Kantor91]) is also commonly used, the traffic generated by the two is similar enough that we felt that simulating only one was sufficient.

3.3.2 File Transfer Protocol

The *File Transfer Protocol* (FTP) provides a service for copying files between computers [Postel85]. It has been one of the most common Internet tools for the distribution of data and software. Each FTP session consists of multiple TCP connections: a *control connection* and one or more *data connections*. User authentication, requests for files, and other commands (plus their responses) are sent via the control connection. Each file is sent via its own TCP data connection.

The number and size of files to be transferred is determined via empirical distributions, again derived from `tcplib`. We note that in our model, all file transfers are downloads

1. We recall that some of the applications use TCP to provide reliable delivery of data. In addition to guaranteeing delivery, TCP also attempts to minimize congestion in the Internet by regulating the amount of data that each connection can send into the network. To simulate these effects on network traffic patterns, we needed to implement the TCP congestion control, error recovery, and connection management algorithms on each host.

(from server to client). We believe that this behavior is representative of most file transfers done on the Internet.

3.3.3 Hypertext Transfer Protocol

The *Hypertext Transfer Protocol* (HTTP) is used for communication between clients (also popularly known as *browsers*) and servers on the World Wide Web [Berners-Lee96]. It is a request-response protocol, in which a client opens a TCP connection to a server and requests a file; the server then replies with the file, using the same TCP connection. The Web uses a model in which each document (also referred to as a *page*), consists of one or more files to be transferred, each using its own TCP connection. For example, a page might consist of text, plus three images, in a total of four files to be transferred. Unlike FTP, there is no control connection; each individual file transfer is self-contained.

As no model of HTTP network traffic was available at the time of this study, we constructed our own model consisting of empirical distributions based on traffic traces. This approach is similar in spirit to that used by `tcplib`. Our model includes the sizes of client requests and replies, as well as the number of files transferred per Web page, and the "think time" between pages. More details about our model, and the process of data collection, can be found in Appendix A.

3.3.4 Simple Mail Transfer Protocol

We simulated one source of "background" network traffic using an extremely course simulation of the *Simple Mail Transfer Protocol* (SMTP) [Postel82]. SMTP uses TCP to handle delivery of electronic mail on the Internet. Each SMTP transfer consists of two phases: a request-response phase consisting of machine identification and option setting, followed by some (sometimes short) one-way bulk data transfers, each containing an email message. Multiple email messages can be sent in each TCP connection.

In INSANE, we only simulate the actual transfer of the email messages (taken from the appropriate `tcplib` distribution), but not any of the handshaking that occurs beforehand. We felt that for our purposes, a more precise simulation was unneeded and would unnecessarily increase the complexity of the simulation.

3.3.5 The Network News Transfer Protocol

Another background traffic source is furnished by servers running the *Network News Transfer Protocol* (NNTP) [Kantor86]. NNTP is a protocol that uses TCP to deliver newsgroup articles between news servers at different sites (they are then stored for retrieval by local news clients). News servers typically transfer articles at selected times (for example, every fifteen minutes), with each server communicating with a fixed set of peers.

INSANE's simulation of NNTP consists of news server processes periodically exchanging batches of news articles. Both the number of articles per batch and the sizes of individual articles are taken from `tcplib` distributions. We model only traffic between news servers, as we feel that client-to-server traffic will be strictly local and not require wide-area access.

3.3.6 Audio

A recent, growing trend is the live transmission of multimedia data across the Internet. To examine this growing class of applications, we included two simulated multimedia applications in our traffic mix. The first sends digital audio data. One example of this type of application is the Video Audio Tool (`vat`) [Jacobson96], which is used for multiparty audio conferencing over the Internet MBONE [Macedonia94], a virtual network used to provide IP multicast services to large portions of the Internet.[2]

Our simulated audio application sends voice-grade audio, using fixed-size packets at a constant throughput of 64 Kbps. The sender is an on-off source, with the on- and off-times controlled by two of `tcplib`'s empirical distributions used for characterizing packet voice traffic.

3.3.7 Video

Another class of multimedia applications sends variable bitrate compressed video. An example of this type of application is `vic`, a video conferencing tool used on the MBONE [McCanne95, McCanne96b]. We used packet traces to derive a simple empirical model of

2. We note that although many current multimedia tools are designed for multicast (one-to-many) dissemination, our simulated applications and network currently support unicast (one-to-one) transmission only.

`vic`'s behavior, when used for teleseminar multicasts. `vic` operates in two states: a conditional replenishment state to update pictures during periods of motion, and a background update state to ensure that over a long time period, all senders eventually get a complete frame image. Our model captures the transitions between these two states, as well as the packet sizes and interarrivals within those two states. More details of our model can be found in Appendix B.

3.3.8 Composite Workload

To provide a complete traffic model to our simulated network, we needed to inject a mix of application traffic. This aspect of the workload can be characterized by a set of arrival processes of new applications starting up on various hosts throughout the network. It is impossible to give a single characterization for all Internet sites—[Cáceres91] showed that Internet traffic varies widely among several sites studied and [Paxson94b] documents the change and growth of Internet traffic over time at a single site. Our workload presents what we feel is a "reasonable" workload for an Internet site, based on existing traffic studies about the contributions of various Internet applications to wide-area network traffic. Table 3-2 summarizes the arrival processes used to regulate the creation of instances of applications at each of the network sites in our simulation scenario.

	Initiated By	**Connection To**	**Type of Startup Arrival Process**	**Mean Conversation Interarrival Time, Per Site (seconds)**
telnet	workstations	workstations	Poisson	10
FTP	workstations	servers	Poisson	15
HTTP	workstations	servers	Poisson	5
SMTP	servers	servers	Poisson	4
NTTP	servers	servers	Uniform	225
Audio	workstations	workstations	Poisson	600
Video	workstations	workstations	Poisson	600

Table 3-2. Application Workload for a Single Site.

For our TCP-based applications, we first created application server processes on various end hosts, which existed for the duration of the simulation. We then used various arrival processes to spawn new application client programs on random hosts. These client processes exchanged data with the designated server processes and terminated upon comple-

tion, with a single exception: Our HTTP model does not contain any notion of a process ending time; it does, however, model interarrival times between user requests for Web pages. We therefore used Poisson processes to place new HTTP clients on each of the client machines at random until every client machine contained a single Web browser, at which point the Poisson processes terminated.

In the case of our UDP-based multimedia applications, we created both the sender and the receiver of data at the same time. Both the sender and receiver processes terminated at the end of the conversation.

3.4 Evaluating IP over ATM Policies

We conducted our evaluation of different IP-over-ATM policies with respect to the application-layer performance of common Internet programs, examining metrics such as file transfer time and packet loss rate. Although this approach made consideration of lower-layer details (such as queue lengths and per-cell delays) difficult, there were two important advantages to this methodology. First, the results directly showed the effects that would be visible to applications and users. Second, addressing the requirements and behaviors of specific Internet applications allowed us to see effects peculiar to those applications, which a more abstract model might not permit.

As the applications we measured all have different requirements, we used different performance metrics for each of them. This section discusses the various measures we used, as summarized in Table 3-3.

For telnet, the main performance metric of interest to the user is the response time taken to receive a reply to a keystroke press. Delays of more than a few tenths of a second will be noticeably annoying. However, delays of less than about one tenth of a second are probably not perceptible. Another metric of interest to telnet users is the time required to establish the TCP connection used by a session. This time measures the delay between initiation of the telnet session to the time the user can actually begin typing keystrokes to the remote machine. Generally, longer delays are permissible for the connect time (compared to the round trip time), as connection setup is performed only once per session.

Application	Metric	Statistic
telnet	Connection time	Median
		90th percentile
	Response time	Median
		90th percentile
FTP	File transfer time	Median
		90th percentile
	Session transfer time	Median
		90th percentile
HTTP	File transfer time	Median
		90th percentile
	Page transfer time	Median
		90th percentile
audio	Loss rate	Overall average
	Overdue rate	Overall average
video	Loss rate	Overall average

Table 3-3. Application-Specific Performance Metrics.

In FTP file transfer sessions, the user is generally interested in retrieving a batch of files in succession from an FTP server. An appropriate performance metric is therefore the total amount of time taken to transfer all the files in an FTP session. To gain some further insights, we also examine the time taken to transfer individual files.

In the case of HTTP, the user is usually retrieving documents one by one, most likely reading each document before requesting the next. Thus, one suitable metric is the time taken to request and retrieve a Web page. Since Web documents can consist of multiple files, this response time includes the time needed to request and receive each of the component files. As with FTP, we also measured the time required to transfer each individual file.

We assume that Internet audio and video applications are flexible enough to adapt to changing end-to-end delays by buffering of data at the receiver. Given this premise, our primary evaluation criteria is the loss rate. The higher the loss rate, the less intelligible will be the audio and video stream at the receiver (we recall that these applications generally run over UDP, which does not provide reliable data transport).

For audio applications, we also measure and evaluate the end-to-end delays. In interactive voice communication, the round-trip delay between two communicating people can be a

determining factor on performance. Studies have shown that the round-trip time cannot exceed 250–300 ms, or the two parties will have difficulty interacting. This round-trip limit leads to a one-way deadline of 125–150 ms; we refer to these packets as *overdue*. We measured the fraction of audio packets meeting or missing a 150 ms deadline. Because Internet video data is typically transmitted at a low frame rate (5–15 frames per second), lip-synchronization with the audio data is fairly useless [Keshav94]. Because of this fact, end-to-end latencies are not as critical as for audio data, so we did not evaluate the delays suffered by the video application.

The sole purpose of including the NNTP and SMTP applications was to provide background network load. Thus, we did not measure the network performance received and experienced by these applications.

In addition to application-specific performance measurements, we occasionally examined other metrics, when appropriate. For example, in our evaluation of virtual circuit caching (Section 6.6), measuring the cache hit rate gave a measure of the effectiveness of reusing idle ATM connections.

We note that some of the quantities we measured are partially dependent on factors external to the network. For example, file transfer times were a function of both network performance and file size. While sizes of files requested in our experiments were generated randomly (and thus beyond the control of the network), they were drawn from identical distributions, across all experiments. Given this latter fact, we feel that comparisons based on these metrics were feasible.

3.5 Experimental Procedure

Our evaluation required two steps. First, we gathered a large amount of data by exploring the space of possible IP-over-ATM policies. We then analyzed our data by performing comparisons between results in such a way as to isolate the effects of the alternatives for each of the three design issues.

Section 3.5.1 provides a few details on the running of our experiments. In Section 3.5.2, we describe our analysis and discuss some of the statistics involved.

3.5.1 Gathering Data

Our experiments covered a wide range of IP-over-ATM policies, intended to explore the policy space discussed in Chapter 2. We used policies formed from the components listed in Figure 3-4, in all meaningful combinations (for example, QOS-aware policies could not be used with `router` multiplexing because that policy did not allow packet classification with a fine enough granularity). The useful combinations of quality-of-service, multiplexing, and virtual circuit management policies resulted in a total of 103 different network configurations.

Design Axis	**Policies**
Quality of Service	`noqos`, `sp`, `wc`, `nwc`
	`telnet`, `ftp`, `http`, `audio`, `video`, `isp`, `av`, `qos1`
Multiplexing	`conv`, `app`, `router`
Virtual Circuit Management	`pvc`, `svc`, `svccache`

Table 3-4. Components of IP-over-ATM Policies.

We ran each of the network configurations at least three times (in some cases more) with different initial random number generator seeds, in order to gain some statistical confidence in our results. There were a total of 336 simulation runs. Each ran for 4000 seconds of simulated time, to reduce the effects of startup transients. In all cases, we used the network configuration of Section 3.2 and the network workload described in Section 3.3.

3.5.2 Analysis of Data

In our analysis, we compared the performance of the Internet applications in our traffic mix when different IP-over-ATM policies were in use. In general, we did pairwise comparisons between policies as we varied single components (for example, comparing two setups with different multiplexing policies, but keeping the QOS and virtual circuit management policies fixed). In other words, our comparisons took place only *along* the design axes of Figure 2-6. This approach permitted us to isolate the effects of different types of policies, while (through complete coverage of the design space) we were also able to see various interactions.

The comparisons we made between quality-of-service policies form the basis for Chapter 4. Those involving multiplexing policies are presented in Chapter 5. Chapter 6 discusses results based on comparisons of virtual circuit management policies.

We note that many of our measurements resulted in sample distributions. For example, we obtained the time taken to transfer each Web page requested during the course of an experiment. To avoid the problem of comparing empirical probability distributions based on thousands of sample points, we chose to compare the median and 90th percentiles of the metrics in question. The performance metrics for which this procedure was applicable are indicated by the rightmost column of Table 3-3.

As described earlier, our evaluation involved a number of comparisons between configurations. For example, we compared the median time required to retrieve a Web page with two different QOS policies, on the basis of three repetitions of each configuration. In such comparisons, it is important to have some idea of the degree of confidence in a comparison, in addition to the result itself. This is expressed in the concept of *statistical significance*, which measures the probability that an apparent difference between systems is actually meaningful and not due to random sampling errors.

Many researchers use the popular *t Test* for comparing unpaired samples for repeated experiments. This statistical test is so named because it uses Student's *t* distribution to compute a confidence interval for the difference between the means of two sets of measurements, which is then used to compare the two corresponding systems. Thus, one can say that with 90% confidence (for example), one system is "better" than another, as measured by some metric. The procedure for computing a *t* Test is straightforward and computationally inexpensive. However, the *t* Test is only applicable in cases where the quantities being measured follow a normal probability distribution.[3]

We had no reason to believe the assumption of normality for any of the performance metrics of interest to us, and in any case we were not able to make enough measurements to

3. [Jain91], a popular performance analysis textbook, discusses the *t* Test, but unfortunately makes no mention of the requirement that data samples follow a normal distribution.

determine the exact distributions (recall that many measurements were run only three times). It is important to note that we do know a great deal about the distribution of the samples themselves (for example, the distribution of Web page retrieval times within a single simulation run). However, we know little about the distributions of various statistics (for instance the median or 90th percentiles) of these performance metrics across repeated simulations.

In our evaluation, we drew on methods from *nonparametric statistics*, which make few assumptions about the distributions of the sample data. In particular, we relied heavily on a statistical test known as the *Mann-Whitney U Test*, as explained, for example, in [Gibbons85]. This test uses the sorted ranks of the data values from both set of measurements to determine the significance of differences between two alternatives. Intuitively, if all the samples from one set of measurements fall below all those from the other set, we can express a fairly high degree of confidence that this is due to actual differences between the two respective systems. Conversely, if the sorted samples are interleaved, it is likely that the two systems do not differ significantly.

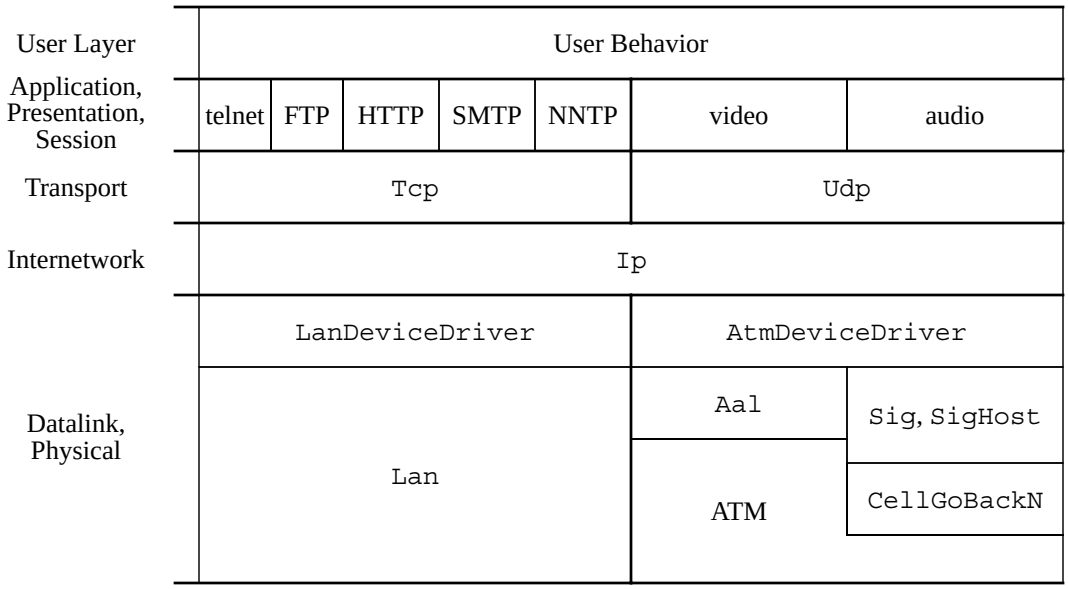
We used the Mann-Whitney *U* Test as a basis for computing a confidence interval for the mean difference between two datasets. If the confidence interval encloses zero, then the two datasets are not statistically different at the specified confidence level. Otherwise, the arithmetic sign of the endpoints of the confidence interval give the relation between the two datasets. In most cases, we performed our comparisons using 90% confidence intervals; this should be assumed in our presentation unless we state otherwise.

3.6 An Internet Simulated ATM Networking Environment

The vehicle for our experiments was a network simulator we have constructed, called the *Internet Simulated ATM Networking Environment* (INSANE). INSANE simulates the operation of a heterogeneous IP internetwork, which can include one or more ATM subnets. Simulated application processes (representing common types of Internet applications) interact with each other over the network and log the performance they receive from the network for off-line analysis. We model a number of different protocol layers, in both the IP and ATM protocol stacks. The bulk of INSANE is implemented in C++

[Stroustrup91], with object classes representing various components of hosts, routers, and switches.

The atomic and composite objects supplied by INSANE provide a simulated networking environment that allows us to measure the network performance seen by various applications such as the World Wide Web or digital video transmission. The complete network protocol stack supported by INSANE is shown in Figure 3-4.



igure 3-4. Protocol Stack of INSANE. Correspondence between the simulator's network layers and the OSI reference model are shown. Where applicable, the simulation object class implementing a particular protocol entity is shown in `fixed-point` type.

In this section, we describe the different protocol layers implemented in INSANE, starting from the datalink and physical layers (Section 3.6.1) and working up towards the application layer (Section 3.6.4). In Section 3.6.5, we discuss some of the more interesting aspects of INSANE's implementation.

3.6.1 Datalink and Physical Layers

Two different subnet technologies are available. The `Lan` type simulates a simple, idealized, shared-media local area network, whose bandwidth and latency can be configured on a per-subnet basis. A `LanDeviceDriver` class handles the software interface between the `Lan` subnet and higher-layer protocols (in this case, IP).

The second datalink layer is an ATM stack, which itself consists of several different components. The ATM layer provides connection-oriented cell delivery service (unreliable, but in-order). To provide for the transport of data units larger than single cells across the ATM network, our ATM stack provides a simple AAL protocol, which performs fragmentation and reassembly of packets in a manner similar to AAL5.

Signalling entities are necessary to establish virtual circuits through the ATM network. INSANE's ATM stack uses a protocol very similar to the Real-Time Channel Administration Protocol (RCAP) [Mah93], which provides signalling functions in the Tenet Real-Time Protocol Suite [Banerjea96]. RCAP uses a single end-to-end round trip of signalling messages to perform admission control and resource allocation for new channels. RCAP relies on a hop-by-hop reliable message delivery service (the prototype RCAP implementation uses TCP for reliable message delivery). In INSANE, this functionality is provided by a module that does a simple positive-acknowledgment retransmission protocol.

Finally, an `AtmDeviceDriver` class provides an interface between IP and the services of the ATM network. In some sense this class is the most important of the entire simulation. It implements all of the various IP-over-ATM policies for quality of service, multiplexing, and virtual circuit management. The specific policies used can be selected at simulation startup time.

The various components of the ATM stack are realized in switches and host adapters implemented as composite objects in INSANE. The structure of the ATM switch is shown in Figure 3-5. It is an output-queued switch, whose architecture is based loosely on the XUNET II switch [Fraser92]. Cells enter the switch via one of the `CellInputPort` objects, each of which represents an input port on the switch. The `SwitchModule` object translates the VCIDs of cells passing through the switch and routes each cell to the appropriate output port (one of the `CellQueueRcsp` objects).

In INSANE's switches (as with XUNET II), virtual circuit setup and teardown functions are performed by a signalling process running on an outboard switch control computer; these processes are simulated by `SigRcsp` objects. Signalling cells are sent using dedicated connections established between adjacent switch controllers at network startup time;

the SwitchModule on each switch routes cells on these connections to the SigRcsp module. A CellGoBackN object placed “in front” of each signalling process provides reliable delivery. We note that in contrast to ATM Forum standard virtual circuits, INSANE’s ATM connections are simplex. Thus, bidirectional communication requires a pair of virtual circuits.

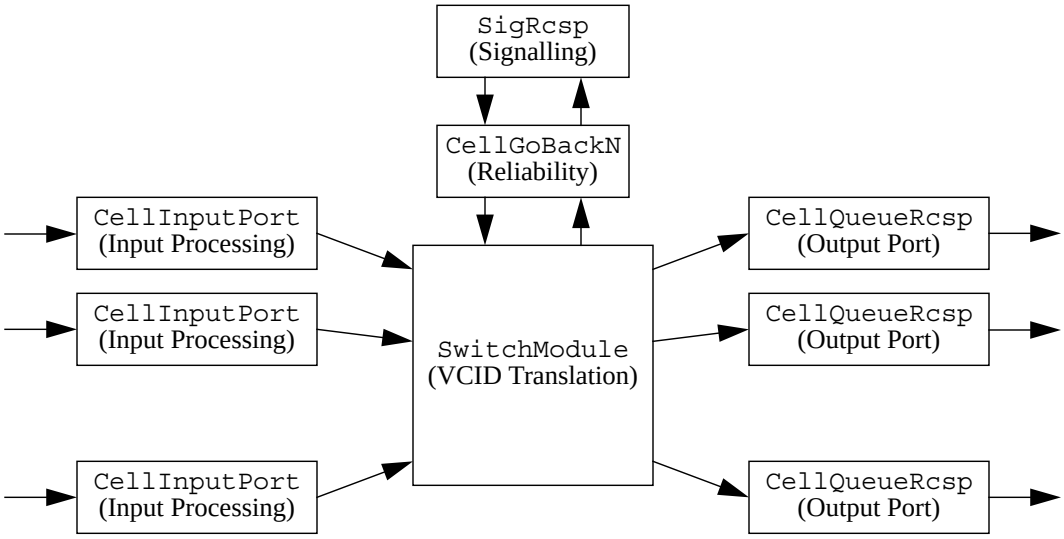


Figure 3-5. ATM Switch Composite Object. Primitive objects are labeled with using fixed-point type. The flow of cells is indicated by arrows. The scheduling discipline used in this output-queued switch (RCSP, as shown) can be changed by replacing the output ports and signalling module.

The structure of the ATM host adapter is very similar to that of the switch. The main difference is that one of the “switch ports” really consists of an Aal object and AtmDevice-Driver object leading to the host’s IP stack. The signalling module is slightly different, in order to account for the interactions between the ATM stack and the host.

INSANE’s ATM switches and host adapters implement a strategy known as *Early Packet Discard* (EPD). This technique attempts to improve the performance of TCP connections traversing ATM networks, in which the unit of congestion loss (a single ATM cell) is smaller than the unit of retransmission (a TCP segment, which may span many ATM cells). Simulation experiments have demonstrated that TCP throughput can be severely degraded due to link bandwidth being wasted carrying useless data for packets that have already lost cells due to congestion. EPD addresses this problem by dropping cells for packets known

to be incomplete due to cell losses [Romanow94]. As some ATM switches currently in production offer this feature, we implemented EPD in the cell queues for all ATM switches and host adapters.

3.6.2 Internetwork Layer

The Internet Protocol is simulated by the `Ip` module (one per host). The primary function of the `Ip` modules is the routing of IP datagrams between hosts. Each has a static routing table that can be loaded at configuration time. Although we have not yet implemented any facilities for supporting dynamic routing, such a feature would not be difficult to add.

3.6.3 Transport Layers

We have implemented two transport-layer protocols in INSANE. The more interesting is the *Transmission Control Protocol* (TCP), implemented by the `Tcp` object class. We based this protocol implementation very heavily on the TCP Reno implementation in BSD 4.4Lite UNIX (a detailed description of the BSD 4.4Lite kernel can be found in [McKusick96], with a walkthrough of its networking implementation in [Wright95]). For simplicity, we omitted several features such as urgent data, although we implemented TCP's connection management, slow start, congestion avoidance, and retransmission algorithms.

We also implemented the BSD-style fast and slow timers, to simulate the operation of BSD-like operating systems and to account for the effect of timer granularity on TCP performance. These timers fire every 200ms and 500 ms respectively, with each `Tcp` object's timers having a different random phase offset, in order to avoid synchronization effects.

The other transport protocol is the *User Datagram Protocol* (UDP), implemented by the `Udp` object class. UDP provides a lossy datagram service used by INSANE's multimedia applications. Although it is frequently used for certain local-area services such as Sun's Network Filesystem (NFS) [Sandberg85], these services and their accompanying protocols are not part of our study.

3.6.4 Application Layers

We have implemented various simulated applications to use the TCP and UDP services of INSANE. The applications, already described in Section 3.3, cover a range from remote logins to the World Wide Web to digital multimedia.

3.6.5 Simulator Implementation

INSANE is an object-oriented, discrete-event simulator. In this type of simulation, various objects, such as queues or network protocol modules, communicate by posting events to each other. Events signify occurrences such as a packet arrival or a timeout. Each event is a message consisting of four components: A time that the event should be delivered, the intended recipient of the event, the type of the event, and a data field whose interpretation is dependent on the event type. Each object in the simulation has an event handler that does all the required processing for each different type of event. A event scheduler is responsible for delivering events to the various objects in the correct chronological order.

This type of simulator organization lends itself to a model of objects as finite state machines, where they react to posted events by updating their internal state and potentially causing actions to happen (such posting events to other objects or writing information to the simulator's log files). In fact, all of INSANE's protocol objects are implemented in this fashion. While this model appears to be a good match for certain abstract objects such as queues, and even some protocol processing entities, it is not a very natural way of expressing the behavior of application programs. In particular, we found that expressing procedure calls or blocking system calls was difficult in this model.

INSANE's discrete-event scheduler, simulation infrastructure, and a large number of built-in objects (also referred to as *atomic objects* or *primitive objects*) are all implemented in C++ [Stroustrup91]. These built-in objects include various types of queues, network protocol modules, and user applications.

Every atomic object exports a set of commands for the Tcl scripting language [Ousterhout94], which allows objects to be easily created and manipulated via Tcl programs. We found the use of Tcl as a configuration language has proven quite useful; network configurations are simply Tcl scripts, which can utilize all of Tcl's control constructs

to build complex objects (also referred to as *composite objects*). Although Tcl is an interpreted language, we see no performance degradation from using it because Tcl scripts are primarily used only to configure a simulation scenario. The actual computations are performed almost entirely by the compiled C++ code. This approach has been taken by several other simulation packages, two fairly recent examples are Ptolemy [Ptolemy96] and `ns` [McCanne96a].

We feel that our simulator is fairly efficient, and is capable of simulating large networks within a reasonable amount of time. When simulating the network in Section 3.2 (approximately 1200 hosts) with the workload described in Section 3.3, INSANE running on an otherwise-idle Sun Ultra 1 (one 167 MHz UltraSPARC processor, 64 MB RAM, Solaris 2.5) could complete a 4000-second simulation run in approximately four hours of wall-clock time. This results in a slowdown of approximately 3.6:1 (the ratio of real time elapsed to simulation time elapsed). During our development, we noted slowdowns of 11:1 on Sparcstation 10 workstations (one 40 MHz SuperSPARC processor, 64 MB RAM, Solaris 2.4) and 8:1 on a Pentium PC (one 100 MHz P5 processor, 48 MB RAM, FreeBSD 2.1.0-RELEASE).

3.7 Experience with INSANE

The long running time of our simulations imposed some heavy demands on our computing environment. We used a distributed computing cluster belonging to the Network of Workstations research group at the University of California at Berkeley. At the time, the NOW cluster consisted of roughly one hundred Sun Ultra 1 workstations and forty Sparcstation 10s and 20s. By using idle workstations, we were able to run a large number of simulations in parallel. To avoid interference with other users of the cluster, we typically ran batches of ten to fifteen runs overnight or on weekends. Each simulation run required about four to five hours of wall-clock time on the Ultra 1s, or about twelve hours of wall-clock time on the Sparcstation 10s. Simulations such as INSANE can be run in parallel reasonably well, since each run was a single, sequential process which required no communication with processes running on any other machines.

The only I/O performed by each job consisted of writing to an output file. We compressed the output files before saving to disk in order to reduce the disk bandwidth requirements. Each run produced approximately ten megabytes of compressed output. We then ran a series of postprocessing scripts over the output files to summarize and analyze the network performance.

4 ATM Quality of Service and IP Conversations

In this chapter, we examine the effects of different ATM quality of service policies on the end-to-end performance of Internet applications. Although the Internet currently has no means for explicitly exploiting ATM QOS features, we found that these features can be used to affect end-to-end application performance. We investigated the effects of using four different scheduling disciplines across a simulated ATM backbone, along with various policies for assigning service parameters to Internet applications. We show that static priority scheduling can be used to indicate preference for certain applications, although low-priority traffic can suffer from starvation caused by high-priority bulk transfers. Continuous media applications can also benefit from the use of guaranteed-performance ATM connections.

4.1 Introduction

One of the features of Asynchronous Transfer Mode technology is its ability to provide performance guarantees. By contrast, the Internet Protocol, as presently deployed, has no support for end-to-end quality of service. We believe, however, that Internet applications can still derive some benefits from using ATM quality of service features to get preferential treatment across an ATM subnet.

The correct quality of service to be used for a conversation will depend on the application. For example, interactive applications such as telnet and rlogin require low delays to be useful. Bulk transfers (such as those performed by FTP) work best over high throughput connections. In many cases the QOS will be implied, based on pre-existing knowledge about applications. Currently, Internet applications are not required to specify their QOS require-

ments (indeed, there is no currently widely-accepted standard for doing so, though RSVP [Zhang93b] is a popular contender). There are, however, several methods for a network to infer the QOS requirements of a stream of IP datagrams:

- By examining TCP or UDP port numbers (or other higher-layer information), a host or router at the edge of the ATM network may be able to determine the application type of an IP conversation, and hence the appropriate quality of service. This approach relies on many applications in the Internet using well-known ports and the network usage of common applications being well-known.

- By monitoring the throughput of a given conversation over time, the network may be able to compute an appropriate set of requirements for an ATM virtual circuit (for example peak and average throughput requirements). Such an adaptive scheme is, of course, only useful when an IP conversation lasts long enough to permit reliable measurements.

- The application may be able to send some sort of QOS request indicating its requirements. Such a message could, for example, be contained within an IP option of a data packet or be sent using a signalling protocol such as RSVP [Zhang93b] or the Real-Time Channel Administration Protocol (RCAP) [Mah93].

- A default set of parameters (perhaps “best effort with no resource reservation”) is necessary for the case in which no QOS can be determined for a given conversation.

We recognize that these approaches to QOS, if implemented only the basis of single subnets, will not provide end-to-end performance guarantees to IP conversations (at least not in the general case of a heterogeneous internetwork). However, it can potentially improve network performance for applications traversing an ATM backbone network, where resources are presumably more scarce than in a local area environment.

In Section 4.2, we discuss some prior and related work in the area of providing quality-of-service guarantees. We discuss some of the mechanisms used in this work in Section 4.3. Three sections of this chapter describe our simulation results with various scheduling disciplines and policies for employing them; a summary of these results can be found in

Table 4-1. Section 4.4 shows the results of using static priority scheduling to give preference to selected Internet applications. In Section 4.5, we show similar results for the work-conserving variant of the Rate Controlled Static Priority (RCSP) scheduler. Section 4.6 discusses results with a rate-jitter-controlled variant of RCSP. Finally, we present our conclusions in Section 4.7.

Static priority scheduling can be used to improve the performance of interactive and continuous-media applications. However, starvation of low-priority traffic is a danger when attempting to use this mechanism to help FTP bulk transfers.
Attempting to use rate control to police bulk transfers yields inconclusive results. In some cases, resource allocation close to the ATM network's capacity can cause admission control tests to fail, causing bulk transfers to be routed over unpoliced, best-effort connections.
Audio and video applications can benefit from the use of guaranteed-performance connections using rate-controlled static priority queueing.
The smoothing provided by rate jitter control reduces the occurrence of buffer overflows and TCP retransmissions, improving long bulk transfer performance.

Table 4-1. Summary of Quality of Service Results.

4.2 Prior Work

Although no prior work has been published on this use of QOS in ATM subnets supporting IP, the problem of providing QOS in an internetworking environment such as the Internet has received considerable attention. For the most part, existing IP networks do not provide any quality of service support. All packets and conversations are treated identically. However, some work has been done with the Type of Service (TOS) bits in the IP header [Almquist92] or IP's precedence field [Bohn94] to express the priority assigned to a datagram. IPv6 contains support for a *Flow ID*, which can be used to identify datagrams as belonging to a particular flow and thus eligible to receive a particular treatment by routers [Deering96].

Various solutions exist to address quality of service considerations in non-IP internetworks. For example, networks based on algorithms and protocols such as the *Tenet Real-Time Protocol Suite* offer mathematically provable end-to-end real-time performance guarantees [Banerjea96]. They require the applications to specify their requirements to the network in advance. Admission control tests are used to limit the number and type of real-time connections allowed into the network, in order to provide deterministic or statistical performance guarantees. These guarantees hold even under "worst-case" conditions.

The Integrated Services model being designed for the Internet includes several service classes to support differential treatment of IP packets [Clark92, Braden94]. One is a *guaranteed service*, providing mathematically provable bounds on delay and bandwidth [Shenker96]. Another service being actively considered is a *controlled-load service*, which attempts to provide requesting applications the loss rates and delays they would receive from an "unloaded" (uncongested) network [Wroclawski96]. The resource reservation protocol RSVP is designed to provide signalling functions for these services in the global Internet [Zhang93b].

4.3 Quality of Service Mechanisms

Several mechanisms are required to support giving different qualities of service to packets in an IP-over-ATM setting. At the lowest level, the ATM network needs to support different qualities of service for cells. We describe the schemes we used in this work in Section 4.3.1.

At a higher layer, IP routers with interfaces onto the ATM subnet can then use these ATM mechanisms according to some QOS policy. To do this, the routers must contain mechanisms for classifying IP packets and forwarding them onto different virtual circuits. Section 4.3.2 describes these methods.

4.3.1 ATM Network Support for Quality of Service

We configured the RCSP schedulers in our simulated switches to support a range of delay bounds. We recall that each of the schedulers controlled access onto a T-1 speed link, (1,536,000 bits per second). At this bit rate, a 53-byte ATM cell (including payload and header, but no other overheads) has a transmission time of 276 microseconds. We ignored delays caused by T-1 framing.

For our simulations, we defined delay bounds supported by each scheduler between 16 and 128 cell transmission times. Due to scheduling granularity, the regulators in the RCSP scheduler imposed an additional worst-case delay of 8 cell times. These settings yielded the local delay bounds for each priority level, as shown in Table 4-2. Best-effort traffic was given the lowest scheduling priority, and was unregulated. Signalling messages were sent

via PVCs whose cells were prioritized higher than best-effort traffic, but lower than all of the guaranteed priority classes.

Level	Cell Times	RCSP Delay Bound (ms)	Switch Delay Bound (ms)
0	16	4.4	6.6
1	32	8.8	11.0
2	64	17.6	19.9
3	128	35.3	37.5
Signalling			
Best-Effort			

Table 4-2. Scheduling Priority Levels and Local Delay Bounds.

The ATM-attached routers requested virtual circuits by specifying the endpoint of the connection and the various QOS parameters, as shown in Table 4-3. We note that these parameters were a subset of those used in [Banerjea96]; the remaining parameters are set by implicitly assuming the use of deterministic delay bounds ($Z_{\min} = 0$), fixed-size ATM cells ($S_{\max} = 48$ bytes), and no packet drops due to buffer overflows ($W_{\min} = 0$).

We note that the RCSP scheduler supports a discrete number of local delay bounds. If a router requested a per-switch delay bound falling between the delay bound values supported by a queue, our signalling software treated the establishment as if it had requested the next lower delay bound supported by the queue.

Note that the delay bounds specified were local delay bounds (in other words, per switch), not end-to-end delay bounds as would normally be expected [Ferrari90]. We designed the network in this way because the decomposition of end-to-end delay bounds into a set of local delay bounds is a problem outside the scope of this work. In particular, some policy must generate, from an end-to-end delay bound, a feasible set of local delay bounds to be requested at each queue. While this computation is simple when given the state of resource utilization at every switch in the network, such information (in perfect form) is unlikely to be available in real networks. Thus, we deferred consideration of this particular issue.

In our different experiments, we actually used several different variants of the RCSP queue, each with different “strengths” of QOS support, as summarized in Table 4-4. The

Parameter	Metric
destination	Destination endpoint of the virtual circuit. Each virtual circuit is simplex (unidirectional) and unicast (one receiver). The ATM protocol stack in INSANE uses small integers as network addresses.
X_{min}	Minimum inter-cell spacing. Determines the peak data rate along a virtual circuit.
X_{ave}	Average inter-cell spacing. Determines the average data rate along a virtual circuit.
I	Averaging interval, over which the average rate specified by X_{ave} must hold.
d	Local delay bound at each switch.

Table 4-3. Parameters Given to Signalling System to Establish a Virtual Circuit.

first, giving the weakest QOS assurances, was a pure static priority scheduler. It uses only the priority queueing mechanisms, without rate control or admission control. Although it provides no performance guarantees, it is a simple approach to providing different network services to various conversations.

Type	Symbol	Priority Queueing	Rate Control	Admission Control	Jitter Control
Best-effort	`noqos`	No	No	No	No
Static Priority	`sp`	Yes	No	No	No
Work-Conserving RCSP	`wc`	Yes	Yes	Yes	No
Non-Work-Conserving RCSP (Rate Jitter Control)	`nwc`	Yes	Yes	Yes	Yes

Table 4-4. Rate-Controlled Static Priority Variants. These different schedulers can all be expressed as similar “versions” of the original RCSP scheduler.

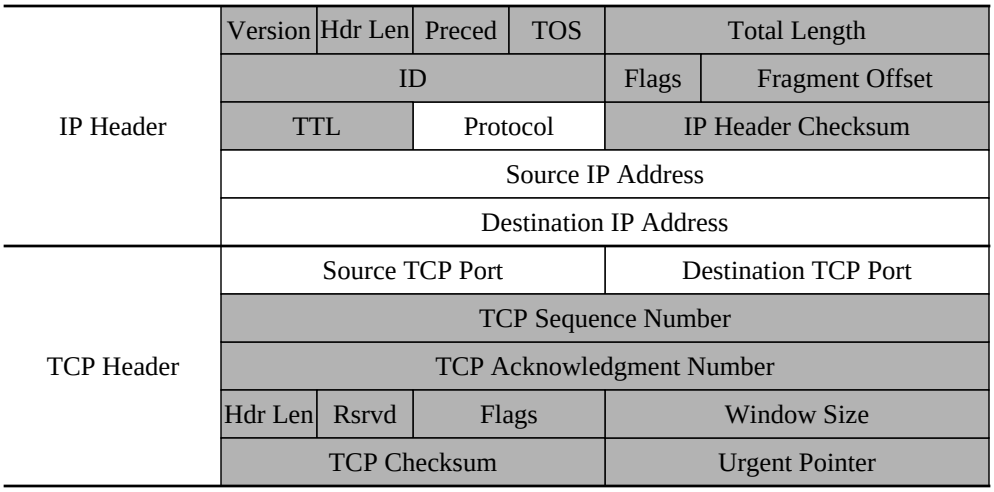
The second variant was a work-conserving RCSP queue. This type of scheduler implements priority queueing and rate control. In order to provide performance guarantees, it relies on admission control at channel setup time.

The last variant was non-work-conserving RCSP. In addition to the features of work-conserving RCSP, it performs a form of distributed *rate jitter control*, in which cells are deliberately delayed in the network in such a way as to partially reconstruct the originally arrival pattern of cells into the network. Although this tactic reduces the variation in end-to-end delays along guaranteed connections, it has the disadvantage of increasing the average end-to-end delay.[1]

4.3.2 Packet Classification

In order to provide differing qualities of service, ATM-attached routers need to be able to classify incoming packets into different IP conversations. This classification must be done on a packet-by-packet basis, since IP is a connectionless network layer protocol. Given the current architecture of IP, it makes the most sense to do this in a thin layer beneath IP and above the ATM adaptation layer; in a BSD-based UNIX implementation, the code would reside as a part of the device driver for the ATM network.

We identify different IP conversations by a *conversation key*, which consists of the source and destination IP address, IP type-of-service field, the transport layer protocol, and (where applicable) the source and destination port numbers at the transport layer. The applicable header fields of a TCP/IP packet are shown in Figure 4-1. Analogous fields are used for UDP or other protocols layered on top of IP. We note that this approach violates the traditional layering paradigm common in networks; it is necessary because IP does not support any notion of connections. Our concept of a conversation is similar to that of an IPv6 flow. In fact, if we were to extend this work to IPv6-over-ATM, we would be able to use its Flow ID field to perform some of the packet classification.



igure 4-1. TCP/IP Header Fields Used for Conversation Keys. White (non-shaded) fields are those used to identify individual IP conversations.

1. The RCSP algorithms support another form of jitter control known as *delay jitter control*. It perfectly reconstructs the original arrival pattern of cells into the ATM network, at every switch. However, it requires timestamping of every ATM cell. We deemed this functionality infeasible to implement in high-speed ATM networks, and so did not investigate this alternative.

We distinguish different applications in one of two ways. All of our TCP applications have the property that they are assigned "well-known ports" (well-known transport-layer addresses) [Reynolds94]. Assigning fixed port numbers to certain applications enables client processes to easily locate server processes (for example, a telnet client application knows that it can locate telnet servers on remote hosts on TCP port 23). An ATM-attached router can check the source and destination port numbers of a TCP packet; if it sees a well-known port number in the TCP source port field, the packet is likely transmitted by a server process to a client process. Conversely, it a well-known port number appears in the TCP destination port field, the packet is likely transmitted by a client process to a server process.[2]

For our UDP applications, determining the application is slightly more problematic. For the audio and video applications we simulated, there are no well-defined port numbers; although we can use the UDP port numbers for the purpose of determining a conversation, we cannot, in general, use these fields to determine the application.[3] We instead simulated the use of higher-layer protocol fields which specify the media type of each packet (e.g. audio or video). An example of such a protocol is the Real-Time Transport Protocol (RTP) [Schulzrinne96].

In Table 4-5, we list the different conversation types and corresponding QOS parameters supported by our simulated routers. In some cases (specifically the telnet, video, and audio applications) the parameters can be derived based on known traffic patterns. Bulk transfer applications such as FTP and HTTP are more difficult to characterize, since these transfers can "expand" to consume all the resources on a link. For these cases, we settled for a set of "reasonable" values.

---

2. Among the applications we studied, there exists one notable exception to this heuristic. World Wide Web URLs can specify a port number for HTTP requests; thus it is possible for Web servers to listen to port numbers other than the default, which is 80. However, in a recent survey of HTML documents collected by the Inktomi "Web crawler", approximately 94% of the documents surveyed were accessed by the standard HTTP port [Woodruff96].

3. Beginning with version 3.5, `mrouted`, the MBONE routing daemon, now assumes a mapping of priorities (and suggested applications) onto UDP port numbers. For example, "highest priority, i.e. audio" data is assumed to map to UDP ports 16384–32767. The Internet session directory tool `sdr` conforms to this mapping, as of version 2.1a1 [Handley96].

Conversation Type	**Direction**	d	X_{min} **(Peak Rate)**	X_{ave} **(Average Rate)**	I
telnet	up	10 ms	20 ms 19.2 Kbps	50 ms 7.68 Kbps	500 ms
	down	10 ms	10 ms 38.4 Kbps	20 ms 19.2 Kbps	1000 ms
FTP (control)	up	20 ms	100 ms 3.84 Kbps	500 ms 0.77 Kbps	5000 ms
	down	20 ms	100 ms 3.84Kbps	500 ms 0.77 Kbps	5000 ms
FTP (data)	up	80 ms	100 ms 3.84 Kbps	100 ms 3.84 Kbps	2000 ms
	down	80 ms	20 ms 19.2 Kbps	20 ms 19.2 Kbps	2000 ms
HTTP	up	40 ms	50 ms 7.68 Kbps	100 ms 3.84 Kbps	10,000 ms
	down	40 ms	4 ms 96.0 Kbps	8 ms 48.0 Kbps	1000 ms
audio	any	20 ms	4 ms 96.0 Kbps	5 ms 76.8 Kbps	100 ms
video	any	50 ms	2.6 ms 148 Kbps	3.5 ms 110 Kbps	2000 ms

Table 4-5. QOS Parameters By Conversation Type. The background applications SMTP and NNTP are not listed because they were always sent best-effort.

Some traffic types have associated with them two different sets of QOS parameters, corresponding to the two directions of a duplex connection. These directions are labelled “up” and “down”. “Up” refers to data sent from a client to a server (for example, packets from a Web browser to a Web server). We apply the label “down” to traffic from a server to a client (such as the packets containing telnet keystroke echoes). We note that even a unidirectional data transfer (such as an FTP download) requires bidirectional connectivity, due to the need to send TCP acknowledgments. Because our simulated ATM connections were simplex only, TCP traffic required two virtual circuits, one each for the “up” and “down” traffic.

We also recall that FTP uses two types of TCP connections, one for control messages and another for actual data transfer. As they use different TCP ports, it is easy to distinguish the two and assign them different QOS parameters, if necessary.

4.4 Static Priority Schemes

Our experiments with static priority schedulers within the ATM network showed that, although prioritizing the traffic produced by individual applications enhances their performance, this effect can come at the cost of the performance of other applications. Some low-volume applications, such as telnet, had no noticeable effect on other traffic. However, assigning higher priorities to bulk transfer applications such as FTP has a severe, detrimental impact on the performance of interactive traffic.

To investigate the effects of a simple static priority scheduler, we experimented with several QOS policies, whose parameters are shown in Table 4-6. Some gave a higher priority to traffic generated by *individual applications*. These policies allowed us to see the "best-case" improvement which individual applications could see, as well as the consequences for other applications. We designated these QOS policies `sp-telnet`, `sp-ftp`, `sp-http`, `sp-audio`, and `sp-video`, so named after the application selected to receive preferential treatment.

Other policies assigned increased priorities to *combinations of applications*, in order to explore interactions between applications. The first, named `sp-isp`, forwards both telnet and HTTP traffic at higher priority. (It received this appellation because it gives preferential treatment to interactive applications of interest to present-day Internet Service Providers.) A second policy, designated `sp-av`, sent all audio and video data at high priority. Finally, the `sp-qos1` policy transmitted data for all supported applications (telnet, FTP, HTTP, audio, and video) at their assigned higher priority levels, with only the background traffic (SMTP and NNTP) sent best-effort.

Within the experiments using each QOS policy, we also varied the multiplexing and virtual circuit policies (two alternatives each, for a total of four different setups, designated `app-svc`, `app-svccache`, `conv-svc`, and `conv-svccache`). We also performed several repetitions (usually three, but sometimes more) of each experiment, with varying random number seeds.

Our results show comparisons of the different QOS policies against results obtained with the `noqos` policy, which sent all traffic best-effort, at the default priority level.

QOS Policy	Conversation Types									
	telnet		FTP (control)		FTP (data)		HTTP		Audio	Video
	up	down	up	down	up	down	up	down		
sp-noqos										
sp-telnet	1	1								
sp-ftp			2	2	3	3				
sp-http							3	3		
sp-audio									2	
sp-video										3
sp-isp	1	1					3	3		
sp-av									2	3
sp-qos1	1	1	2	2	3	3	3	3	2	3

Table 4-6. Scheduling Priority Levels for Static Priority Policies. Each row represents a single QOS policy for the static priority scheduler, while each column stands for the data generated by a given application. Numbers signify the priority level given to an application's data by a specific QOS policy (lower numbers indicate priorities). Blank entries signify that an application receives best-effort, lowest-priority service. SMTP and NNTP conversations, not shown here, are always sent best-effort.

4.4.1 Single-Application Static Priority Policies

Our results with single-application static priority policies show that they can yield significant performance improvements for bulk transfers (such as FTP and HTTP) and continuous media applications. The interactive remote login application (telnet) saw smaller gains. However, the lack of admission control or policing caused problems for telnet, audio, and video applications when FTP bulk transfers were given priority.

The `sp-telnet` QOS policy caused only small decreases in both the telnet connection setup time and keystroke response time. We saw improvements in the median connect time in the two `svccache` configurations (20 ms faster connections, for a 10% speedup). The 90th percentile only showed statistically significant improvements in the `app-svc` configuration (the average speedup was 100 ms, or 29%). Figure 4-2 illustrates these effects.

We also measured the median and 90th percentile of telnet response times; that is, the time needed to get a response to user keystroke packets. We saw statistically significant improvements in the response time, averaging about 20% in the median and almost 40% at the 90th percentile. The absolute magnitude of these differences, however, was quite

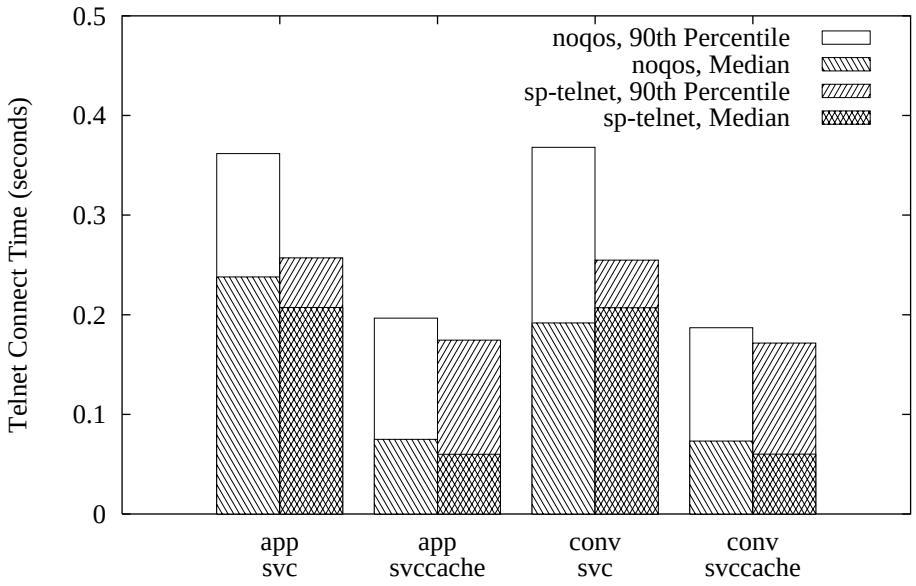


Figure 4-2. Effects of `sp-telnet` Policy on Telnet Connect Times. Stacked bars denote the 90th percentile and median connect times. The effect of the new QOS policy can be seen by comparing adjacent bars within each pair. Pairs of bars correspond to different multiplexing and virtual circuit management policies, as discussed in later chapters.

small (about 16 ms and 70 ms respectively), probably not perceptible to humans. These effects are shown in Figure 4-3.

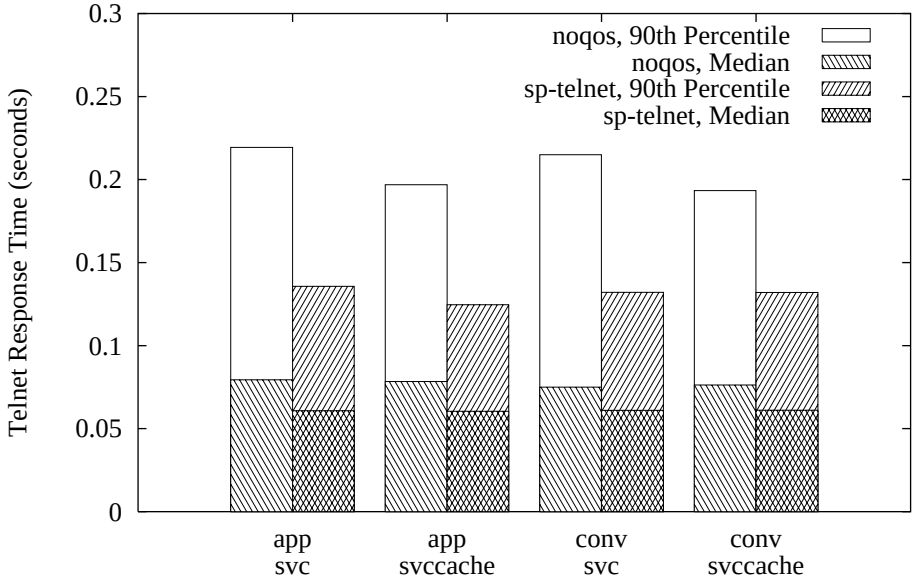


Figure 4-3. Effects of `sp-telnet` Policy on Telnet Round-Trip Times.

When we turned our attention to the `sp-ftp` policy, we saw that it significantly reduced the amount of time required to complete single FTP file transfers, as shown in Figure 4-4.

The use of `sp-ftp` had only a minuscule effect on the median file transfer time. However, it succeeded in decreasing the 90th percentile of file transfer times by about half. We attribute the difference to fixed overheads (such as propagation delays) playing a more significant role in small file transfers than in large ones.

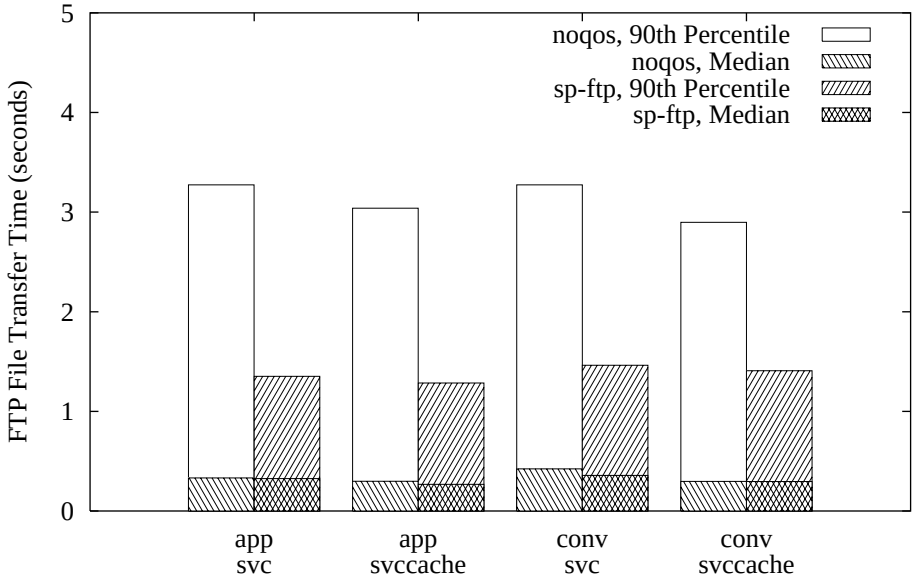


Figure 4-4. Effects of `sp-ftp` Policy on FTP File Transfer Time.

The effects on the completion time of an entire FTP session were more apparent. This metric measures the time to establish a control connection from client to server, transfer some number of files, and close the control connection. We saw improvements of about 30% in the median session time and about 50% in the 90th percentile of session times, as shown in Figure 4-5. The absolute values of these improvements, tens of seconds, should be easily perceived by users.

The drawbacks of a static priority scheme were borne out, however, in the fact that the `sp-ftp` policy delivered significantly worse performance to other applications. For example, telnet delays were much increased; the 90th percentile of telnet round-trip times increased by 32–78%. The median Web page retrieval time increased by 1–29%, as shown in Figure 4-6. This difference was only statistically significant for the `conv-svc` configuration at 80% confidence, but the `app-svc`, `conv-svc`, and `conv-svccache` setups showed longer retrieval times with 60% confidence. Perhaps more importantly, the audio

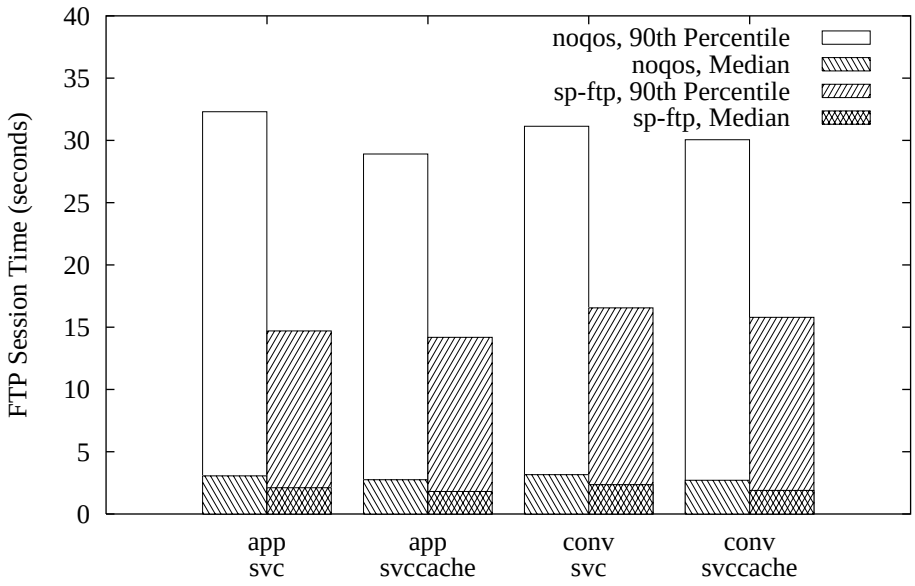


Figure 4-5. Effects of `sp-ftp` policy on FTP Session Times.

loss rate increased to 2.0–2.8% (an almost tenfold increase). We observed the video loss rate to be in the range of 4–8%, almost three times what it was with the `noqos` policy.

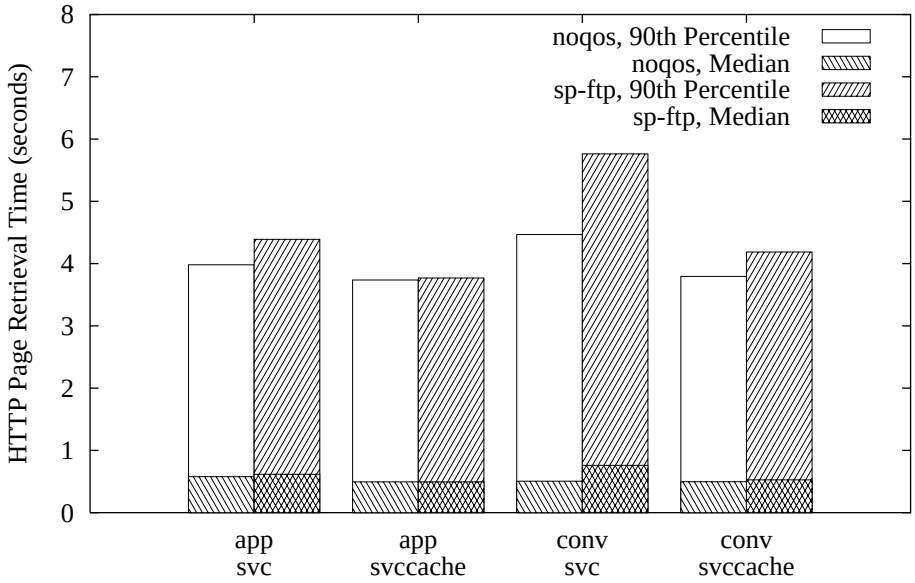


Figure 4-6. Effects of `sp-ftp` Policy on HTTP Performance.

The effect of the `sp-http` policy on Web traffic was similar to that of `sp-ftp` on FTP. It had the general effect of reducing the time to transfer individual files by 12–20% at the 90th percentile and by 30–35% at the median. For the most part, however, the per-file gains were reflected in improvements in the transfer time of complete Web pages. The median

page transfer times were shortened by about 20%. The 90th percentile of transfer times reflected 37–45% improvements. Interestingly, we could find no statistically significant effects on other applications, even at the 80% confidence level, and examining the mean differences provided no obvious trends. We believe that this result was a consequence of the generally shorter length of HTTP files, compared to FTP files.

When we ran the `sp-audio` policy, it had the expected effect of reducing the loss rate of audio data, at least in scenarios using per-application (`app`) multiplexing. The packet loss rate, in the range of several packets per thousand in the case of best-effort data, was reduced, in most cases, to slightly less than one packet per thousand. This was an improvement of 44–82%, the effects of which are shown in Figure 4-7. The effects with per-conversation (`conv`) multiplexing were not statistically significant. However, we saw that the fraction of overdue audio packets (taking longer than 150 ms to reach their destinations) dropped from 5–7% to 2–3%, a reduction of 43–75%.

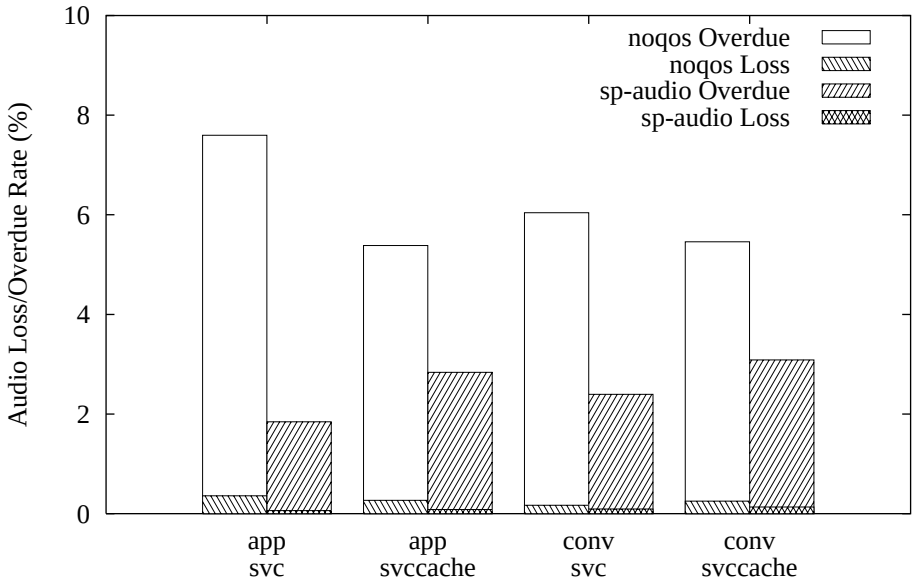


Figure 4-7. Effects of `sp-audio` Policy on Audio Loss Rate.

We expected, and saw, large reductions in the packet loss rate when sending video data at a higher priority (the `sp-video` policy). The loss rate decreased from 1–2% to about 0.1% or less, a reduction in the loss rate of about 90–95%. This improvement was statistically significant for `app-svc`, `app-svccache`, and `conv-svccache` setups. In a

qualitative sense, the results were similar to those of the `sp-audio` policy on audio data, but more pronounced.

4.4.2 Combination Static Priority Policies

To a certain extent, running combinations of applications at higher priority had effects similar to those of individual applications. In some cases, however, an excess of high-priority traffic caused a number of expected performance improvements to disappear. One likely explanation is the starvation of the ATM signalling system, which we observed in the case of the `sp-qos1` policy.

As described earlier, the `sp-isp` policy assigns a higher-than-default priority to both telnet and HTTP traffic. We noted a general improvement in the performance of both applications, only slightly less than those seen with the single-application policies `sp-telnet` and `sp-http`, individually. For comparison, the 90th percentile of telnet response times improved by 29–39% (with three of four configurations showing statistically significant improvement), as compared to 36–46% with `sp-telnet`. The 90th percentile of Web page transfer times improved by 35–44%, whereas with the `sp-http` policy it improved by 37–46%. In general, sending these two applications' data via high-priority connections had no significant impact on the other measured applications.

In a similar fashion, audio and video applications saw improvements with the `sp-av` policy, similar to the individual `sp-audio` and `sp-video` policies. Both applications saw sizeable reductions in their loss rates (and, in the case of the audio application, the overdue rate). Most setups exhibited improvement with 90% confidence, and a few only with 80% confidence. As with the `sp-isp` policy, applications using the default priority were only slightly affected (or not at all) by the `sp-av` policy.

The `sp-qos1` scheme exhibited an interesting combination of effects. Telnet connect times rose significantly for `svc` setups, an average of two to four times longer at the 90th percentile. To explain this, we note that telnet connect times consist of two components. First is the time to establish an ATM virtual circuit across the backbone. Second is the time taken to do a TCP connection establishment using that ATM connection. We infer that the observed increase was due to drastically longer virtual circuit setup times, since the actual

TCP packets for telnet had the highest priority of any in the network. Our evidence indicates that signalling traffic (given its own priority level just above the default, best-effort priority) was partially starved by the large volume of higher-priority data traffic. This situation could be avoided by sending all signalling traffic at the highest priority level, or by using guaranteed-performance virtual circuits for signalling traffic. We note that this effect was somewhat mitigated by the use of virtual circuit caching, because telnet performance became less dependent on the ability of the ATM signalling system to establish virtual circuits quickly.

Some other metrics also showed changes with the `sp-qos1` scheme. The 90th percentile of telnet round-trip times was reduced by an average of 50–90 ms. FTP and HTTP transfer times showed little significant changes, except for 20–25% speedups in the `conv-svc-cache` setup. The `conv-svccache` configuration also produced large, significant reductions in the audio and video loss rates, as well as the audio overdue rate. At lower confidence levels (80%) these improvements became significant for other configurations as well.

4.5 Work-Conserving RCSP

In general, our continuous media (audio and video) applications benefited from the use of work-conserving RCSP, in particular with respect to audio delays and video losses. Applications that performed bulk transfers (FTP and HTTP) experienced large degradations of performance because the policing in the ATM network prevented them from opportunistically using all of the link bandwidth. These effects, observed when applications were selected individually for guaranteed-performance connections, generally extended to multiple-application policies as well.

There are two primary differences between the work-conserving RCSP scheduler and the static priority scheduler used in Section 4.4. First is the addition of rate control. The work-conserving RCSP scheduler maintains the notion of an eligibility time for each cell. Roughly speaking, cells that arrive too early, according to their connection’s traffic parameters, are marked “ineligible”. Ineligible cells receive the lowest-possible scheduling pri-

ority (even worse service than for best-effort traffic). The net effect is that any traffic sent in excess of a connection's traffic specification will receive much-degraded service.

The second difference is the addition of admission control tests, performed at connection setup time. These admission tests allow the ATM network to support performance guarantees; a connection request is denied if its performance guarantees cannot be met or if its acceptance could permit another connection's guarantees to be broken. It is therefore possible for a virtual circuit request to fail. We instructed our IP-over-ATM implementation to react to the failure of a guaranteed-performance request by opening a best-effort connection instead.

As with the static priority scheduler, we performed tests both with individual application policies (designated `wc-telnet`, `wc-ftp`, *et al.*) and with multiple application policies (`wc-isp`, `wc-av`, and `wc-qos1`). The applications selected for special treatment by the scheduler were the same as those for the static priority scheduler, but we set the per-application traffic parameters using the values in Table 4-5. We present comparisons of these policies with the default `noqos` policy.

4.5.1 Single-Application Work-Conserving RCSP Policies

We saw that interactive logins were almost unaffected by the use of work-conserving RCSP scheduling. Applications that performed file transfer (FTP and HTTP) saw significant increases in the time taken to complete their operations. Audio and video seemed to benefit from the use of guaranteed connections.

When we sent telnet traffic using work-conserving RCSP (`wc-telnet`), we observed no statistically significant effects on either the connect times or round-trip response times. We recall that the `sp-telnet` policy caused slightly shorter times for both metrics. The difference between the `wc-telnet` and `sp-telnet` cases can most likely be explained by the effects of policing in the work-conserving RCSP scheduler. We recall that it degrades the priority of ineligible cells, which would have retained their higher priority with a static priority scheduler.

When we used the `wc-ftp` policy, FTP file transfers suffered from significantly increased completion times. At the 90th percentile, files took as much as thirteen times longer to transfer than with the `noqos` policy. The effect at the median was somewhat less (a 15–25% increase in file transfer times). We observed a similar effect on FTP session completion times. The median sessions took 5–7 times longer to complete, while at the 90th percentile, sessions took 11–14 times longer to transmit all their data. These results, shown in Figure 4-8, were a consequence of the policing and rate limiting of the RCSP scheduler. With no QOS controls in place, each FTP file transfer could potentially use all of the available bandwidth along its path (subject to TCP's congestion and flow control algorithms). However, the work-conserving RCSP scheduler only guaranteed the performance of data sent within the confines of its traffic specification. Data sent in excess of the traffic specification could have been delayed considerably.

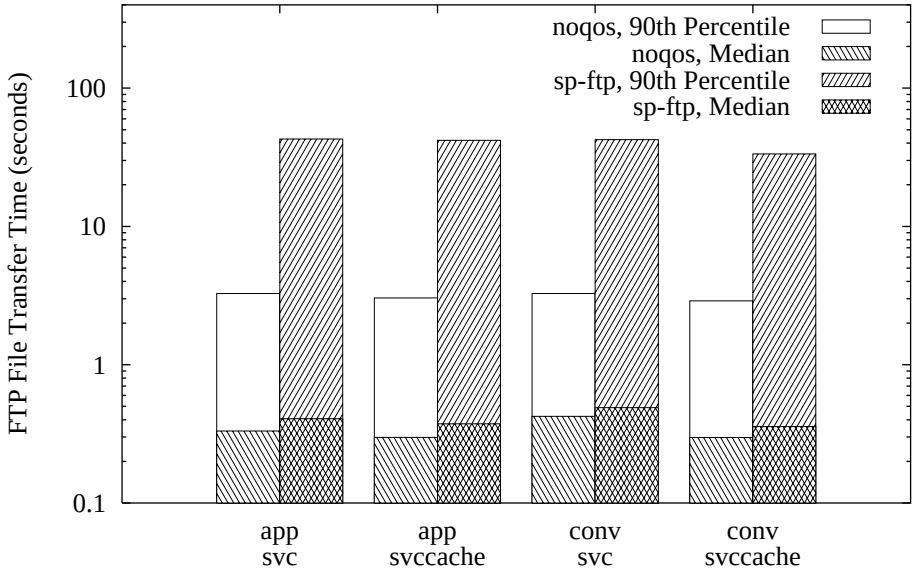


Figure 4-8. Effects of `wc-ftp` Policy on FTP File Transfer Times. Note that completion times are shown on a logarithmic scale.

While detrimental to the performance of FTP, we expected that the policing provided by the `wc-ftp` policy and the RCSP scheduler would be useful to other applications which could benefit from the network bandwidth that would otherwise be used by FTP. This effect would manifest itself, for example, in lower telnet response times and Web page

transfer times. We were, however, unable to find any statistically significant effects on other applications.

In the same manner that the `wc-ftp` policy limited the performance of FTP, the `wc-http` policy throttled the performance of HTTP. Compared to the `noqos` policy, the median file transfer time was increased by 12%–55%, while the 90th-percentile time was increased by 36%–434%. We observed similar effects in the HTTP page transfer times, which increased by 27–212% in the median and 68%–1494% in the 90th percentile.

Somewhat surprisingly, the `wc-audio` policy had no statistically significant effects on the loss rates. For packets actually delivered, however, the `wc-audio` policy decreased the rate of overdue audio data by 30–64%, though these improvements were only statistically significant for the `app-svc` configuration. These effects are illustrated in Figure 4-9.

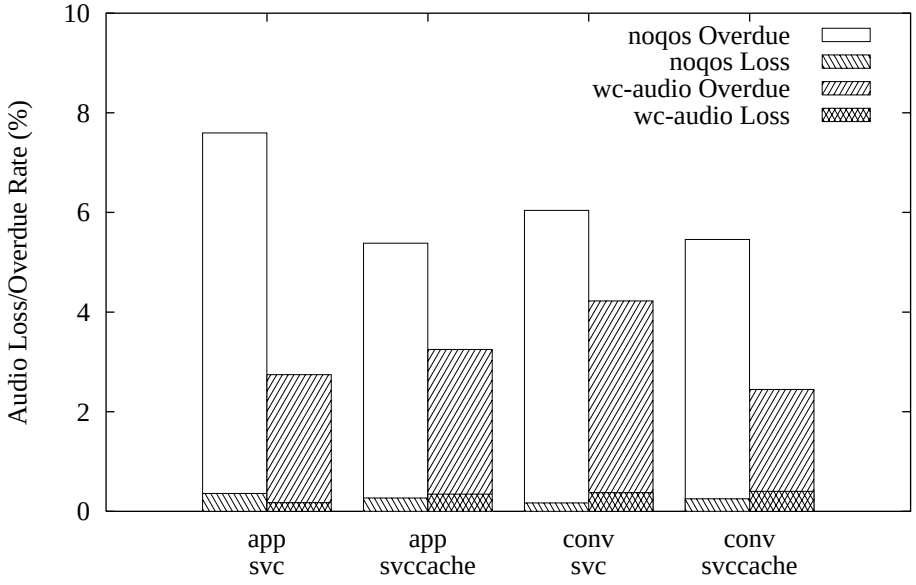


Figure 4-9. Effects of `wc-audio` Policy on Audio Overdue Rate and Loss Rate.

When we used the `wc-video` QOS policy, we saw a reduction in the video loss rate, from an original loss rate of 1–2% to well under 1%. The improvement in loss rate was 54–87%. However, the variances we observed were so large that even at 60% confidence, only the `svc` policies showed statistically significant improvements.

4.5.2 Combination Work-Conserving RCSP Policies

As we expected, the effects of the multiple-application QOS policies were very similar to the superposition of the single-application `wc` policies.

When we ran our workload using the `wc-isp` policy, Web browsing (HTTP) was generally slower (the 90th percentile of Web page retrievals increased by 44–147%, a difference of several seconds per page). Telnet performance improved slightly. With 90% confidence, the 90th percent of round-trip times showed improvements for the `app-svc` case (only), whereas all but the `conv-svccache` showed improvements with 60% confidence. The delays were reduced by a few hundredths of a second (an 8–20% improvement).

Interestingly, the `wc-av` policy failed to produce any statistically significant effects on audio or video loss or overdue rates. Given the magnitude of the effects of the `wc-audio` and `wc-video` policies on the same applications, this is perhaps not surprising; however, our initial intuition led us to expect some significant performance improvements.

The effects of the `wc-qos1` policy were a combination of the individual application policies. Telnet traffic experienced some minor improvements in both connection setup time and round-trip response time. At the 90th percentile, these speedups were on the order of 10–60 ms (5–15%) and 40–60 ms (13–25%), respectively. However, we only observed statistically significant differences for the `app-svc` configuration.

Bulk transfers under `wc-qos1`, throttled down by policing in the ATM network, saw increased delays. The 90th percentile of FTP sessions took 21–53 seconds (71–171%) longer. Similarly, the 90th percentile of Web page transfers increased by 0.7–3.4 seconds (16–90%), although this degradation was only significant for scenarios with per-application multiplexing.

4.6 Non-Work-Conserving RCSP

Applications using non-work-conserving RCSP exhibited effects similar to those of work-conserving RCSP. Bulk transfer performance was improved over work-conserving RCSP because the smoothing effects of the rate jitter control caused fewer cells to be dropped in the ATM network. We found that the policing effect of RCSP virtual circuits could be ren-

dered less effective when there were enough guaranteed virtual circuits to cause admission control tests to fail; this resulted in a number of bulk transfers being sent over unpoliced, best-effort connections.

There is only one difference between the work-conserving and non-work-conserving variants of the RCSP scheduler. We recall that the work-conserving variant can forward cells that have arrived too early or too quickly, albeit at a much lower scheduling priority. In the non-work-conserving variant, these cells are buffered until they are once again compliant with the traffic specification for their connection. This difference translates into increased delays for sources that send data faster than their traffic specification.

4.6.1 Single-Application, Non-Work-Conserving RCSP Policies

For the most part, the use of non-work-conserving, jitter-controlled RCSP (`nwc`) had effects similar to those with the use of work-conserving RCSP (`wc`). Although the per-application policies succeeded in restricting the performance of bulk transfer applications, we could find no statistically significant evidence that other applications were able to take advantage of the resulting conservation of network resources. Interestingly, the performance degradation with the `nwc` schedulers was not as severe as with the `wc` schedulers, due to the former's traffic shaping helping to control buffer overflows. The continuous media applications were, in general, helped by having their data sent over guaranteed connections, suffering from less dropped or overdue data.

When telnet traffic alone was sent using guaranteed connections (the `nwc-telnet` policy), connect times were, for the most part, only slightly changed.

The `nwc-ftp` policy had a somewhat counterintuitive effect on FTP traffic. As expected, FTP file and session completion times were increased (the 90th percentile of file transfer times increased by 680–742%, with FTP sessions taking 645–814% longer). However, these effects were somewhat less than for the `wc-ftp` policy, as shown in Figure 4-10.

This result ran contrary to our expectations. We expected FTP performance with `nwc-ftp` to be worse than with the work-conserving `wc-ftp`, due to the extra delays introduced by jitter control. We investigated further and found that bulk transfers carried by

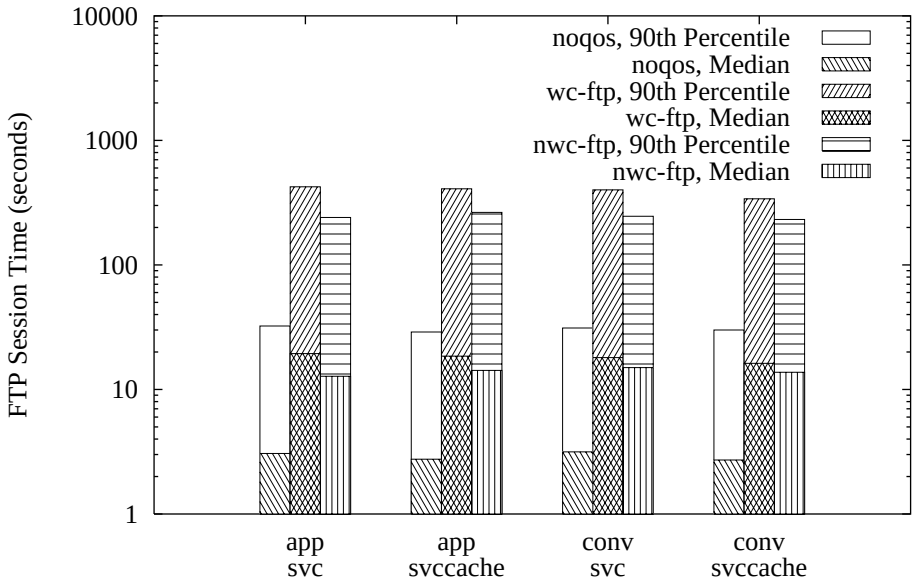


Figure 4-10. Effects of `nwc-ftp` Policy on FTP Session Times. Note that the use of a on-work-conserving RCSP variant (rightmost bars in each set of three) actually results in less performance degradation than work-conserving RCSP (center bars).

work-conserving connections suffered from more dropped packets, due to buffer overflows. These overflows appear to have been caused by bursts of cells. The resulting drops caused TCP timeouts and retransmissions, and reduced the throughput sufficiently to offset the effects of the lower delays. We did not observe these packet drops with the `nwc` scheduler; the jitter-controlled queues tended to smooth out large bursts at the entrance to the ATM network, as well as in intermediate switches.

In the same way that the `nwc-ftp` policy slowed down FTP transfers, the `nwc-http` policy caused Web browsing to take more time. The 90th percentile of Web page transfers increased from 3.8–4.0 seconds to 5.8–11.4 seconds, an increase of 53.6%–185%. As with the case of FTP described earlier in this section, the degradation of the `nwc-http` policy on HTTP was not as great as that for `wc-http`. The latter, we recall, caused the 90th percentile of page transfer times to increase as much as fifteen times in one configuration.

The `nwc-audio` QOS policy only had minor effects on audio traffic. We observed average loss rates of 0.12%–0.22% (a reduction in the loss rate of 12%–52%). Only 1.7%–4.0% of packets were overdue (a reduction in the overdue rate of 33%–53%). However, the only

statistically significant difference in the overdue rate was in the `app-svc` setups. We saw no significant changes in the loss rate. These effects are illustrated in Figure 4-11.

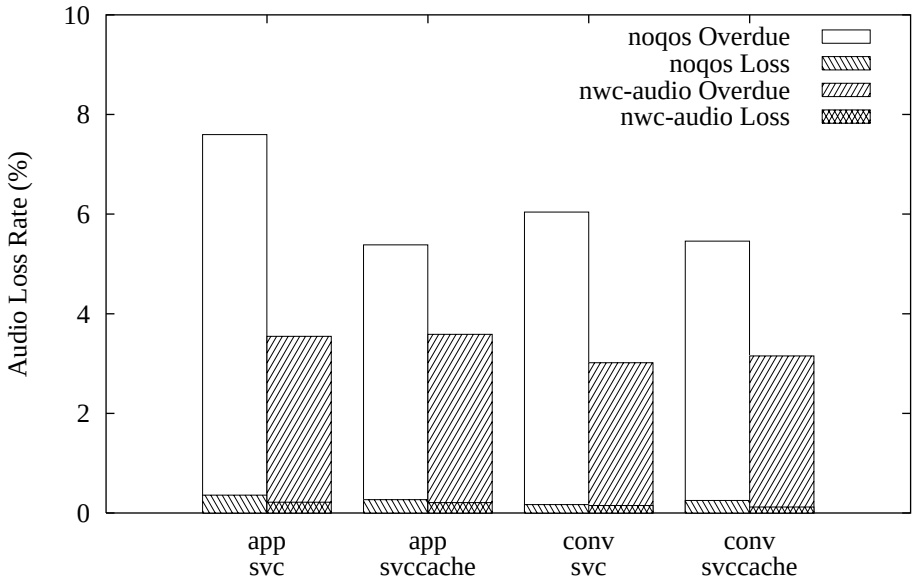


Figure 4-11. Effects of `nwc-audio` Policy on Audio Loss Rate and Overdue Rate.

As we expected, the `nwc-video` policy lowered the loss rate of video data by providing it with guaranteed performance. The loss rates dropped from 1.1%–2.6% (with the original `noqos` policy) down to 0.39%–0.74%. This reduction corresponded to an improvement of 31–77% in the loss rate. As with our audio application, however, these effects were generally not statistically significant.

4.6.2 Combined, Non-Work-Conserving RCSP Policies

In general, we found that the multiple-application `nwc` policies had effects very much like the combination of the individual application policies. An interesting side-effect we observed was that, in some cases, large amounts of reserved resources caused a high failure rate of guaranteed virtual circuit requests. These failures caused a significant number of bulk transfers to be carried by best-effort virtual circuits, with no policing or jitter control, which actually caused them to have lower transfer times.

The effects of the `nwc-isp` policy were similar to those of `nwc-telnet` and `nwc-isp` combined. There were no significant effects on telnet latencies. HTTP applications saw lower transfer times with the `nwc-isp` policy than with the `nwc-http` policy, with the

90th percentile of Web page transfers increasing to 5.2–7.4 seconds, or an increase of 37–97%. At first this result seems somewhat counter-intuitive. However, we note that the `nwc-isp` setups suffered from higher connection failure rates than the `nwc-http` scenarios (25–37% for `nwc-isp` to 7%-32% for `nwc-http`, depending on the multiplexing and virtual circuit usage policies). These failures occurred because the network could not provide the performance guarantees requested by new channels. The IP conversations corresponding to the failed VCs were then sent using best-effort connections. Since these connections were unpoliced, the conversations using them actually tended to receive better performance (leading to lower transfer times) than they would have if the original guaranteed virtual circuit setups were successful.

The `nwc-av` policy produced few statistically significant effects on the performance of the audio or video sessions (the overdue rate for `app-svc` setups was reduced, with 90% confidence). However, we observed that the audio loss rates for application multiplexing dropped by 32–47%, and the overdue rate (for all configurations) was reduced by 18–55%. This result was consistent with those produced by the `nwc-audio`, `nwc-video`, and `wc-av` QOS policies, examined earlier.

Given the results of the use of single-application non-work-conserving QOS policies, the effects of the `nwc-qos1` policy were fairly predictable. Telnet experienced small, but not significant, reductions in both connect time and round-trip time (3–14% at the 90th percentile). FTP suffered from increased file and session times (an increase, for example, of 52–109% for the 90th percentile of session completions). We note that this performance degradation is minor compared to that produced by the `nwc-ftp` policy. We believe that, as seen with HTTP in the `wc-isp` policy, increased amounts of guaranteed traffic caused some FTP conversations to be carried by best-effort VCs due to admission control failures. This in turn actually allowed FTP conversations to experience better performance than they would have otherwise. Understandably the situation for HTTP traffic under the `nwc-qos1` policy was similar. We saw no significant effects on either the audio or video traffic.

4.7 Conclusions

In this chapter, we examined the use of several different scheduling disciplines to be used in the queues of an ATM network. We looked at a best-effort policy (`noqos`), a static priority scheduler (`sp`), and two variants of rate-controlled static priority scheduling (`wc` and `nwc`). We explored the use of these schedulers to express preference for different network applications, and measured their effect on application performance.

We saw in these experiments that static priority queueing in the ATM backbone is a useful mechanism for giving preferential treatment to selected applications. However, allowing bulk transfer applications (such as FTP or HTTP) to use high-priority virtual circuits can have an adverse affect on interactive and continuous media performance. We have also seen situations in which a large amount of high-priority traffic can cause partial starvation of service to the ATM network signalling system (especially without virtual circuit caching). As mentioned earlier, giving higher priority to signalling messages would alleviate this particular problem.

We noted that policing of guaranteed ATM virtual circuits can be an effective mechanism in controlling the performance of bulk transfer applications. However, our QOS policies generated some counter-intuitive effects when the amount of guaranteed connections in the ATM network was high. To wit, admission control failures resulted in IP conversations being carried by (unpoliced) best effort connections, thus receiving better performance than they would have if they had been carried by guaranteed ATM connections. We believe that a partial solution to this problem can be implemented by using (if available) a priority of service even lower than “best effort”, for conversations that fail admission control tests in the ATM network.

We saw that bulk transfers can also benefit from the effects of jitter control, as this mechanism is an effective way of smoothing traffic, and helping to prevent buffer overflows in ATM queues. These buffer overflows in turn lead to packet losses, which in turn cause TCP timeouts and retransmissions. In isolated tests we performed, we saw that the packet loss rate was so high that the TCP fast retransmission and recovery mechanisms were rendered useless.

Based on our simulation results to date, we advocate using static priority schemes to improve the performance of interactive (e.g. telnet) and multimedia (e.g. audio, video) applications. Another additional, useful option is to use guaranteed-performance virtual circuits for multimedia traffic. Both of these classes of QOS policies improve performance for some selected applications without significantly impacting the behavior of others. The actual choice of a QOS policy will likely depend on administrative concerns (for example, determining which applications are “important” and should receive better service).

We close with some potential directions for future work. Some of the expected performance effects failed to materialize, or did so without any meaningful level of statistical confidence. Thus, we must make several observations about the effectiveness of our experiments and methodology. Statistical significance was, in many cases, difficult to see, even with confidence intervals as wide as 80%. The most effective remedy to this problem would likely be to perform a larger number of experiments (difficult to do with our experimental setup, because of the computational cost). We further believe that this would allow some of the secondary effects to become more visible.

We also feel that some of the results were potentially colored by a light workload in the network; the intuitive idea being that some of these QOS policies could not make performance “better” because it was already “good”. A larger amount of background traffic, as well as a higher arrival rate of supported user applications, would likely make the effects of different QOS policies more apparent.

The classification method we used to determine QOS parameters is somewhat rigid, in that it depends on using fixed quantities (port numbers) in transport-layer headers to select one of a small, fixed set of QOS parameters. This scheme can encounter difficulties in the event of misclassification (for example, if a video conversation were mistakenly classified as telnet, rate control in the ATM network would render the resulting stream unusable). It also has no capability to handle new or unknown traffic types. These limitations could potentially be solved by more flexible mechanisms, perhaps adapting to the traffic load of a conversation or using additional information provided by applications.

Finally, there is a large space to be explored in setting and tuning the QOS parameters to be used for different Internet applications. In particular, it would be interesting to compare parameter settings for bulk transfers, since they can, in an uncontrolled network, opportunistically use as much network bandwidth as there is available.

5 IP over ATM Multiplexing Policies

This chapter examines different IP-over-ATM multiplexing policies and their effects on the performance of common Internet applications. Because IP has no network-layer connections, there is no single, obvious mapping from IP conversations to ATM virtual circuits. An ATM-attached router can use ATM virtual circuits to carry individual conversations, or it can choose to aggregate the traffic from several IP conversations onto one connection. Our simulation results show that multiplexing traffic in this way generally improves the performance of file transfers (such as those required for small FTP or Web files). In some cases, however, contention for buffer space or interactions with traffic policing inside the ATM network can lead to performance degradations for longer file transfers and continuous media applications.

5.1 Introduction

One of the most noticeable differences between IP and ATM is found in their respective connection models. Because IP is connectionless, there is no one, unique mapping from IP conversations onto ATM virtual circuits. One natural policy is for ATM-attached routers to give each IP conversation its own connection across the ATM subnet. However, other approaches might multiplex the data from several conversations (for example, several TCP conversations) together onto the same ATM connection.

Since many conversations are short (such as those for a majority of Web pages), we expect that aggregating them together will result in improved performance, due to the elimination of virtual circuit setup overheads. The time to establish a connection may well be a large fraction of the time needed to perform a short file transfer. Multiplexing a number of con-

versations together means that only a single virtual circuit needs to be established for the group, thus amortizing (or masking) the delays caused by virtual circuit setup.

The use of guarantees has some additional implications. Intuitively, we expect that multiplexing a virtual circuit among many IP conversations should yield better utilization of that virtual circuit's allocated network resources due to statistical multiplexing "within" the connection. However, this greater utilization comes at the expense of a decreased level of protection between conversations sharing the same ATM virtual circuit.

To see how this aggregation of traffic might be useful, we can examine the characteristics of audio-video conferences on the Internet, as described in [Mah94b] and [Keshav94], and consider their transmission across an ATM subnet[1]. In these conferences, only one user speaks at a time (except for occasional transients). A single ATM virtual circuit could be multiplexed among all of the involved UDP/IP conversations and used to carry the audio for the entire conference. That virtual circuit would only require enough resources to support one sender at a time. By contrast, most users send video data continuously throughout a video conference; in order to protect the performance of each of the video streams, each of the associated UDP/IP conversations would need to be assigned its own ATM virtual circuit. The issue of resource sharing among related conversations is investigated in more detail in [Gupta95a] and [Gupta95b].

In Section 5.2, we briefly discuss some prior evaluations of IP-over-ATM multiplexing policies. Section 5.3 presents three multiplexing policies that we examined in this study. We show our simulation results in two sections (highlights are listed in Table 5-1); Section 5.4 addresses the merits of per-conversation and per-application multiplexing, while Section 5.5 deals with per-router multiplexing as a separate case. Our conclusions are presented in Section 5.6.

1. The conferences studied in [Mah94b] and [Keshav94] were sent using IP multicast. The support of IP multicast over an ATM network introduces additional issues and is beyond the scope of this study. The discussion in this example merely addresses the question of how to forward the data from an audio/video conference across an ATM subnet.

Multiplexing several IP conversations over a single ATM virtual circuit generally shortens file transfer times (especially short FTP or HTTP transfers).
When the ATM network does rate policing, unrelated IP conversations sharing an ATM virtual circuit can interfere with each other, causing each other's packets to be delayed. This effect can result in somewhat longer file transfer times, especially for long file transfers.
Large amounts of multiplexing can cause large packet losses through buffer contention. This effect manifests itself in higher loss rates for multimedia applications.

Table 5-1. Summary of Multiplexing Results.

5.2 Prior Work

Several studies have already examined the issue of multiplexing datagrams for different IP conversations over the same ATM virtual circuit. This research, however, was performed in the context of best-effort virtual circuits. [Cáceres92] explored several different policies for multiplexing TCP conversations in a wide-area ATM network carrying TCP/IP data traffic. The study, as updated by [Cáceres93], suggested that the best multiplexing policy is to establish a virtual circuit per conversation (combined with a round-robin service discipline in ATM switches); however, this policy was only considered for best-effort virtual circuits.

Another study, based on Internet traffic measurements, found that many wide-area conversations are short [Claffy94]. It recommended that such conversations be routed through a mesh of Permanent Virtual Circuits (PVCs) in order to avoid the latency incurred by ATM virtual circuit establishment. The study also claimed that on-demand Switched Virtual Circuits (SVCs) are only necessary for conversations with different priorities or QOS requirements, or for conversations whose high resource utilization would adversely impact the performance of other traffic over the PVC mesh (such as high bitrate video).

Commercially available ATM LANs, such as the FORE Systems ATM LAN described in [Biagioni93], typically multiplex all communication between a given pair of hosts on a single virtual circuit (either a PVC or an SVC). This approach has the advantage of implementation simplicity, but has the disadvantage of giving identical treatment to all packets between a given host pair.

5.3 Multiplexing Policies Examined

We examined three different IP-over-ATM multiplexing policies, which we describe here in order of increasing levels of traffic aggregation. The first policy was per-conversation multiplexing, abbreviated as `conv`. It assigns each IP conversation its own virtual circuit, where a conversation is a TCP connection or a flow of UDP datagrams. This policy is illustrated in Figure 5-1.

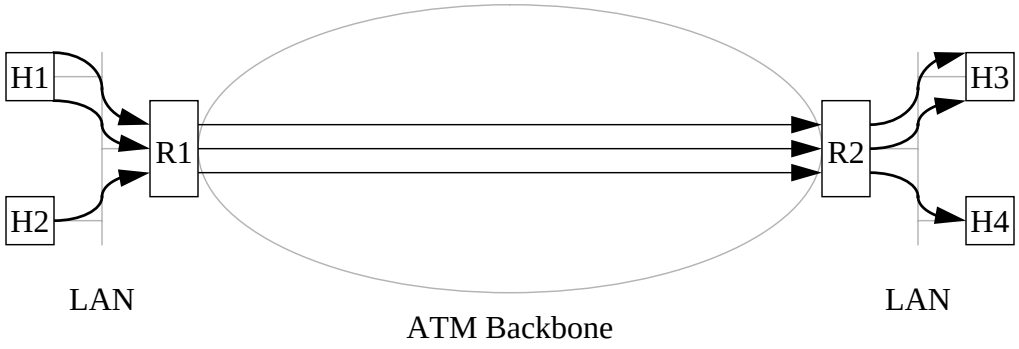
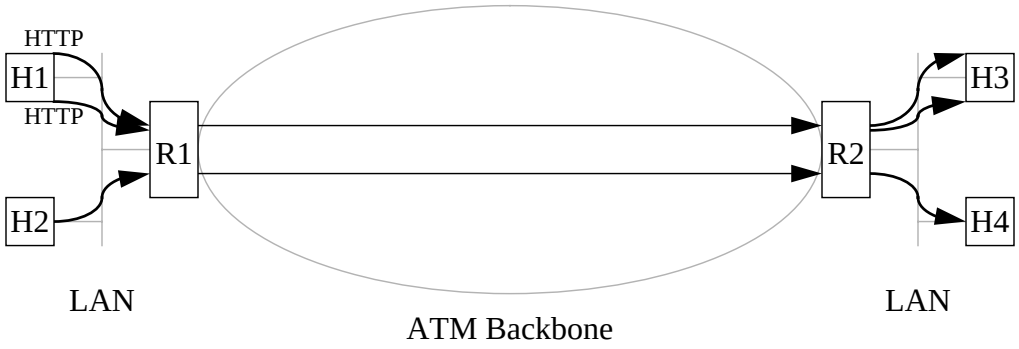


Figure 5-1. Per-Conversation Multiplexing. Each of the three IP conversations has its packets transported over the ATM backbone by a separate ATM connection between the routers R1 and R2.

Aggregating all of the traffic from a single application between a pair of end hosts yields the per-application multiplexing policy, known as `app`. This policy, whose operation is shown in Figure 5-2, was evaluated in [Cáceres92]. It has the effect of reducing the number of virtual circuit setups required, in the case of repeated TCP connections or UDP flows from one host to another. For example, this policy would forward all of the HTTP connections for a Web page onto a single virtual circuit, because they all share the same IP source and destination, and carry traffic for the same application.[2]

With the `app` multiplexing policy, a router cannot know the exact number of IP conversations that will be sharing a virtual circuit (although we can describe them, as in the example above). When the QOS policy uses guaranteed-performance virtual circuits, this introduces the problem of not knowing, at connection setup time, the appropriate QOS to carry this set of conversations. In our experiments, where it was applicable, we set the QOS for

2. An ATM-attached router can generally determine the application for which a packet carries data by examining packet headers and looking for well-known TCP or UDP port numbers, as discussed in Section 4.3.2.



igure 5-2. Per-Application Multiplexing. The top two conversations, both carrying HTTP data from host H1 to host H3, share the same virtual circuit over the ATM subnet.

a multiplexed virtual circuit to be the same as that for a non-multiplexed (per-conversation) virtual circuit. This choice seemed appropriate as, due to the traffic patterns of the applications we studied, there was usually only one active conversation per virtual circuit. However, in Section 5.4.2, we note some interesting transient effects which arose from the sharing of resources between IP conversations.

Finally, we consider a policy that aggregates all of the traffic between a given pair of routers onto a single virtual circuit. We refer to this policy as per-router-pair or per-router multiplexing, abbreviated as `router`. We shown an example in Figure 5-3. Per-router-pair multiplexing is frequently used by commercial ATM LANs such as the FORE Systems network described in [Biagioni93]. Several ATM network testbeds have used PVCs with this policy, such as XUNET II [Fraser92] and BAGNET [Johnston95].

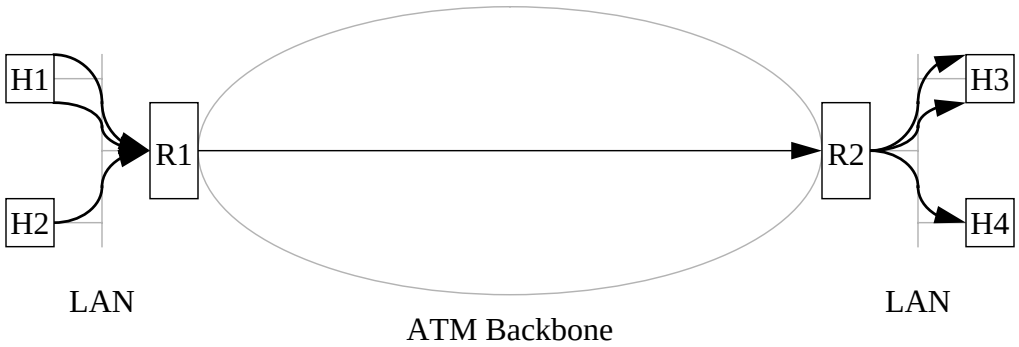


Figure 5-3. Per-Router Pair Multiplexing. All of the data passing through router R1 and router R2 (regardless of end hosts or traffic types) is forwarded over a single connection.

Similar to the app multiplexing policy, router lacks knowledge about the conversations sharing each virtual circuit. Unlike the app policy, however, ATM connections supporting the router policy are likely to carry a wide variety of different conversations, with many active at any given time. Because our QOS parameters are designed to support specific applications, we cannot assign any single QOS to this aggregation of traffic. Therefore, all of our experiments with the router policy involved best-effort service only.

5.4 Per-Application and Per-Conversation Multiplexing

In this section we compare the application performance under two different multiplexing policies: per-application multiplexing (app) and per-conversation multiplexing (conv). We performed these comparisons using a variety of QOS policies and both the svc and svccache virtual circuit policies. We found that app multiplexing was helpful for operations that depended heavily on the speed of connection setup, such as telnet connection setups and short bulk transfers. However, interactions between different IP conversations sharing an ATM connection, even though not active simultaneously, caused long file transfers to take place more slowly in cases where the ATM network did traffic policing (the nwc-* and wc-* QOS policies).

5.4.1 Telnet

In general, the performance of the telnet application was marginally improved by the use of per-application multiplexing, over a per-conversation policy. The per-conversation policy yielded longer connect times and round-trip times. For example, when we ran the sp-ftp policy, the 90th percentiles of connect times were longer by 40–80 ms (14–18%) and the 90th percentile of round-trip times was lengthened by 30–90 ms (12-30%). This effect is shown in Figure 5-4. The average effect of the multiplexing policy was greater with svc (as might be expected), but we only saw statistical significance in the svc-cache case.

5.4.2 FTP

An interesting effect on FTP performance was that the performance of short files and long files tended to favor different multiplexing policies. Short files were generally transferred

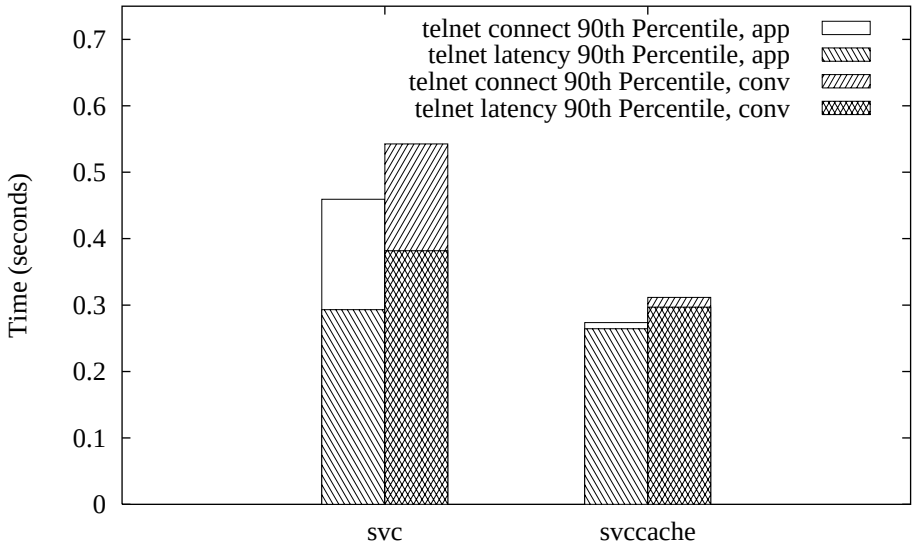


Figure 5-4. Performance Effects of `app` and `conv` Multiplexing on Telnet Performance, `sp-ftp` QOS Policy.

in less time with `app` multiplexing but, in rate-controlled scenarios, long files took less time to send using `conv` multiplexing.

Short FTP file transfers seemed to complete in less time when the IP-over-ATM service used `app` multiplexing. This effect was most pronounced in the `sp-ftp` setups. In this scenario, we saw that switching from `app` to `conv` multiplexing resulted in an increase in file transfer time of 60–90 ms (27–33%) at the median and 110–120 ms (8–10%) in the 90th percentile. Comparisons of the performance of both FTP file transfers and session transfers are shown in Figure 5-5 and Figure 5-6, respectively. We note that this effect is proportionately more pronounced at the median of file transfer times, probably because the connection setup time accounts for a larger fraction of the total transfer time for smaller files.

In setups where FTP data conversations were policed by the ATM schedulers, however, long files and sessions tended to benefit more from the use of per-conversation multiplexing (or showed little change between the two multiplexing policies). Both the `wc-ftp` and `nwc-ftp` setups showed this effect. We illustrate the `nwc-ftp` scenarios in Figure 5-7 and Figure 5-8; in both plots, the difference illustrated by the `svccache` bars is statistically significant.

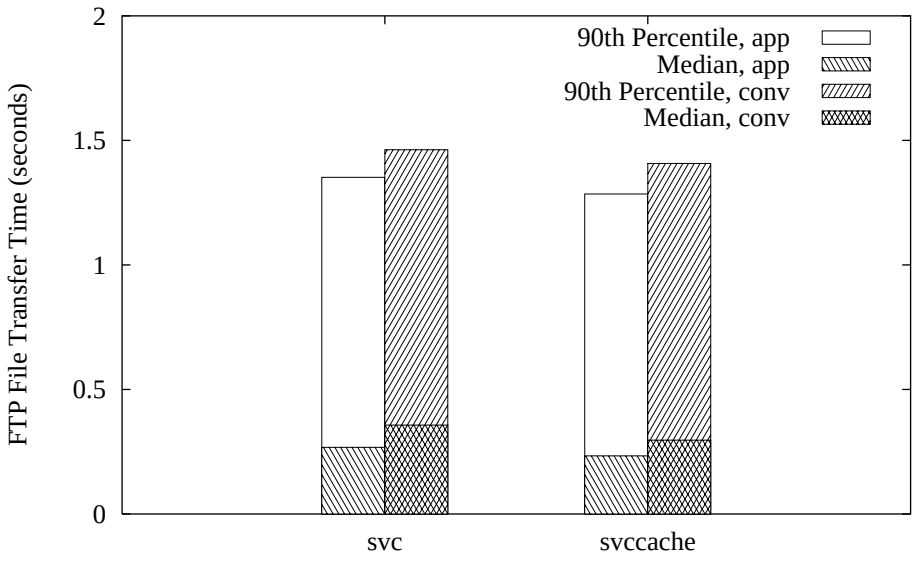


Figure 5-5. Performance Effects of `app` and `conv` Multiplexing on FTP File Transfer Time, `sp-ftp` Policy.

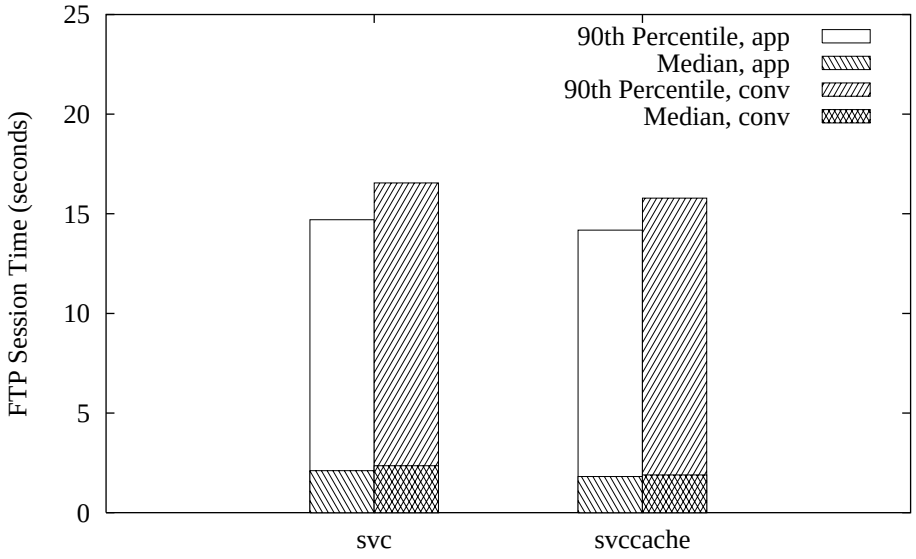


Figure 5-6. Performance Effects of `app` and `conv` Multiplexing on FTP Session Time, `sp-ftp` Policy.

Intuitively, longer files (and sessions) do not benefit as much from not having to wait for connection setup, because this one-time cost becomes small compared to the time needed to transfer a large file. However, the bursts generated by large files can interfere with other files using the same virtual circuit. This phenomenon takes place in the queues inside the ATM network. A burst of cells generated by a packet from one conversation can have the

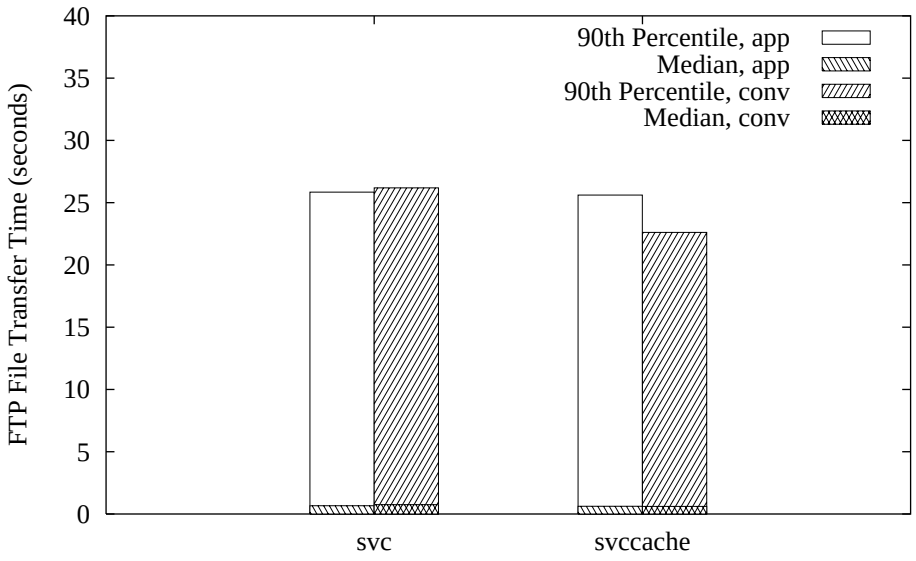


Figure 5-7. Performance Effects of `app` and `conv` Multiplexing on FTP File Transfer Times, `nwc-ftp` QOS Policy.

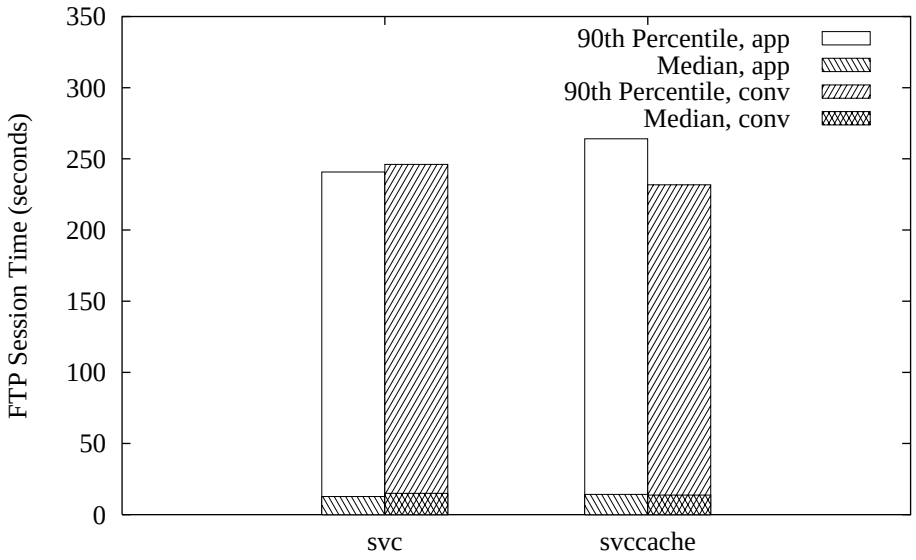


Figure 5-8. Performance Effects of `app` and `conv` Multiplexing on FTP Session Times, `nwc-ftp` QOS Policy.

effect of delaying the cells for a following conversation on the same virtual circuit. This effect occurs because the combined traffic load momentarily exceeds the traffic specification for the ATM connection. The policing mechanisms in the ATM queues reshape the traffic to correspond to the traffic specification, thus artificially delaying following cells and increasing their end-to-end delay. In Figure 5-9, we illustrate this effect. We note that

this occurrence does not require that the two (or more) conversations be active simultaneously. It can easily involve the last packet of one conversation and the first packet of the following, non-overlapping conversation.

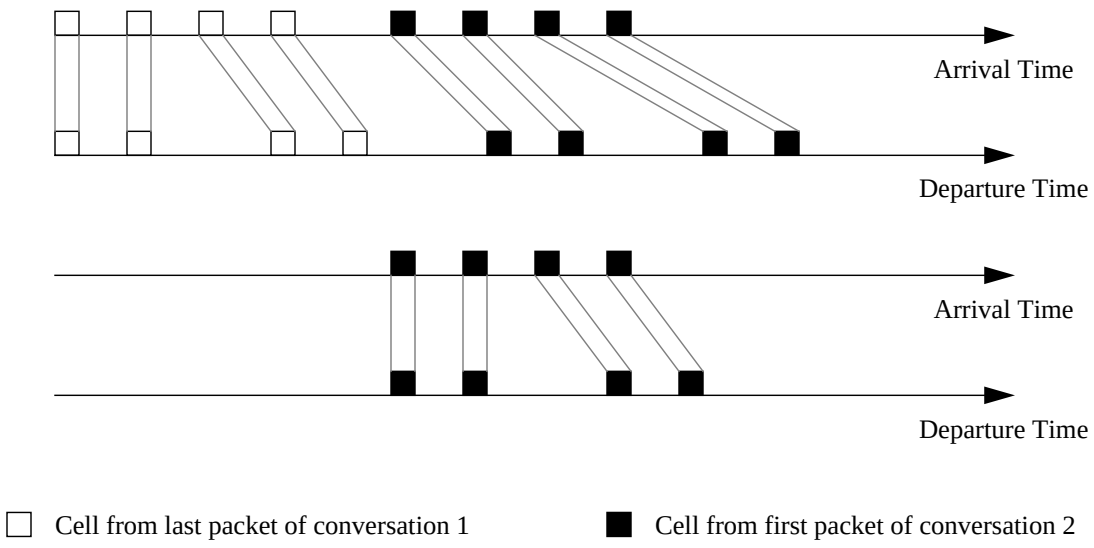


Figure 5-9. Illustration of Interference Between Conversations Sharing an ATM Virtual Circuit. This graph shows arrival and departure times from a rate-controlled ATM queue. In the top picture, two conversations share a virtual circuit. A burst of cells causes the eligibility time of subsequent cells, from an unrelated conversation, to be pushed into the future; the cells will leave later than in the bottom picture, in which conversation 2 is assigned its own virtual circuit, without interference from conversation 1.

5.4.3 HTTP

Similar to the case with FTP, we observed two different effects with Web traffic. For the QOS policies without rate control (i.e. all of the `sp-*` policies and the `noqos` policy), the use of per-application multiplexing produced shorter item and page retrieval times than did the use of per-conversation multiplexing. In Figure 5-10 and Figure 5-11, we compare the transfer times with the two multiplexing policies; we found statistically significant differences for all metrics in the `svc` case and for the 90th percentile of item retrievals in the `svccache` setup. As was the case with FTP, this effect is mostly likely due to the elimination of virtual circuit setups for repeated TCP connections from HTTP clients to their servers.

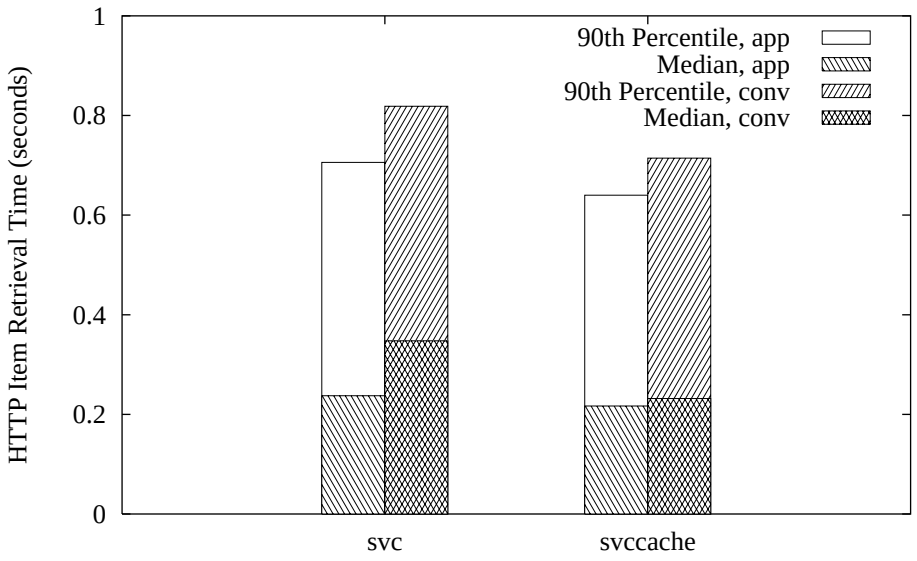
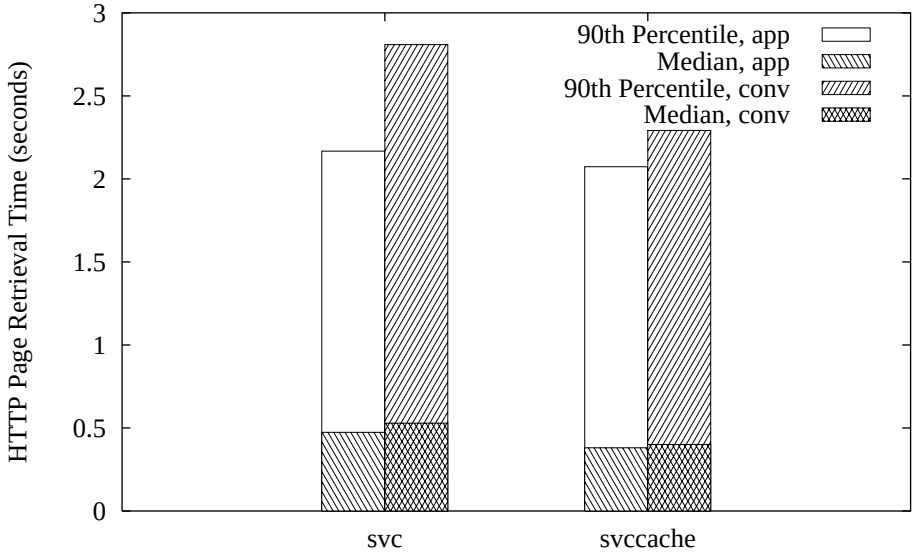


Figure 5-10. Performance Effects of `app` and `conv` Multiplexing on HTTP Item Retrieval Time, `sp-http` Policy.



igure 5-11. Performance Effects of `app` and `conv` Multiplexing on HTTP Item Retrieval Time, `sp-http` Policy.

Conversely, in the case of QOS policies that implemented rate control (those using the `wc` and `nwc` schedulers), per-conversation multiplexing tended to reduce item and page transfer times compared to per-application multiplexing. With the `nwc-http` QOS policy, using per-conversation multiplexing resulted in significantly lower median and 90th per-

centile item and page transfer times. Figure 5-12 and Figure 5-13 show the comparison between transfer times with the two multiplexing policies.

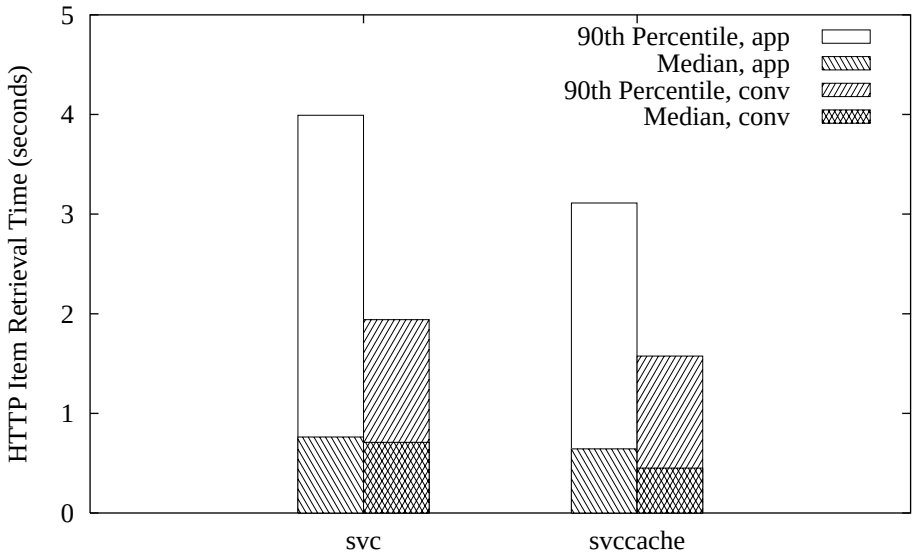


Figure 5-12. Performance Effects of `app` and `conv` Multiplexing on HTTP Item Retrieval Time, `nwc-http` Policy.

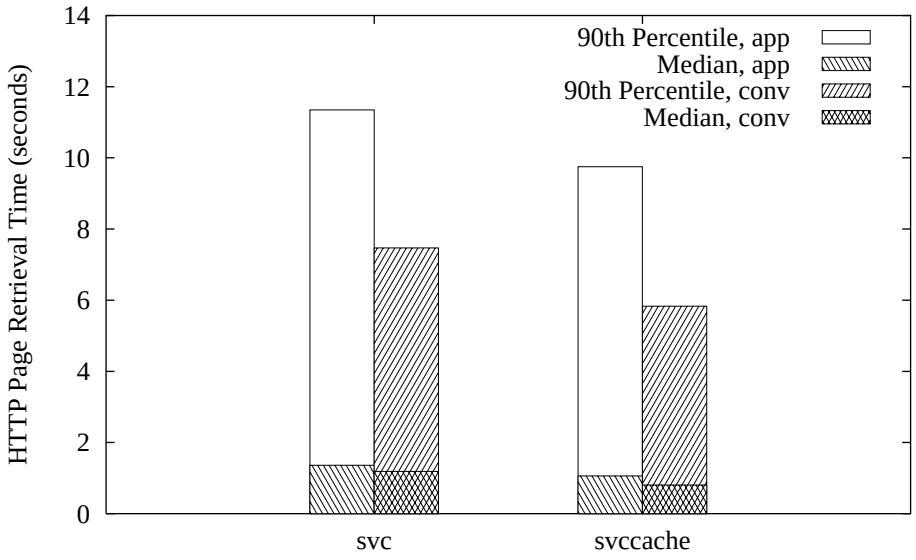


Figure 5-13. Performance Effects of `app` and `conv` Multiplexing on HTTP Page Retrieval Time, `nwc-http` Policy.

A likely explanation for these performance effects is that, as we discuss in Appendix A, much Web activity involves repeated, short TCP connections. As we saw for FTP in Section 5.4.2, repeated TCP connections sharing a virtual circuit interfere with each other in ATM rate controllers. Bursts of cells sent from earlier item retrievals will, through the

ATM policing, cause the data for subsequent retrievals to be delayed (either through degrading the priority or explicit queueing of ineligible cells). This effect would likely be exacerbated if the Web browsers modeled had used multiple, parallel TCP connections (currently done in contemporary Web clients to improve interactive response time).

5.4.4 Audio

We saw few significant differences in either the audio loss rate or the audio overdue rate. This result is likely due to the fact that, as with telnet, there is not enough locality of audio conversations for the multiplexing policy to directly affect audio performance. In addition, our workload has a fairly small number of audio conversations active at any one time.

5.4.5 Video

The performance experienced by our video application was similarly unchanged with respect to the multiplexing policy we used. We feel that the reasons are the same as for the audio application (no locality between endpoints, with few active conversations).

5.5 Per-Router Multiplexing

This section deals with per-router multiplexing, and compares it to setups using per-conversation multiplexing. The results we saw were similar to those in Section 5.4. Operations whose completion time was strongly influenced by TCP connection setup were generally sped up by the use of `router` multiplexing. However, long file transfers and sessions were completed faster with `conv` multiplexing, most likely due to competition for buffers along the path of shared virtual circuits. This assertion was borne out by the performance of audio traffic, which also suffered from increased losses under the `router` policy.

There is one other notable difference between our tests in this section and those in Section 5.4. Per-router multiplexing implies the use of best-effort virtual circuits only, since all of the various QOS policies we examined are based on application-specific parameters. Therefore this evaluation, which compares per-conversation to per-router multiplexing, was considerably simpler because it only needs to consider the `sp-noqos` QOS policy.

The use of per-router multiplexing made it extremely unlikely that a new conversation would require the establishment of an ATM connection. Due to the fact that each virtual circuit carried an aggregate of many sources, it would almost never be idle long enough to be timed out and torn down. Across all simulation runs with `router` multiplexing, there were between 30 and 48 virtual circuit requests per run; we note that 30 virtual circuits is the minimum necessary to support a fully-connected mesh between all routers.[3] By comparison, the `sp-noqos-conv-svc` setups required between 162,000 and 174,000 virtual circuit establishments over the course of a run.

5.5.1 Telnet

Intuitively, we would expect that the telnet clients would take less time to connect to their servers. We did observe this effect, for both `svc` and `svccache` virtual circuit policies; however we only saw a statistically significant difference for the `svc` case, which saw the median and 90th percentile reduced by 62% and 57%, respectively. This effect is illustrated in Figure 5-14. The minor effects for the `svccache` setups can likely be attributed to the effectiveness of the virtual circuit caches (we explore this subject in Section 6.6).

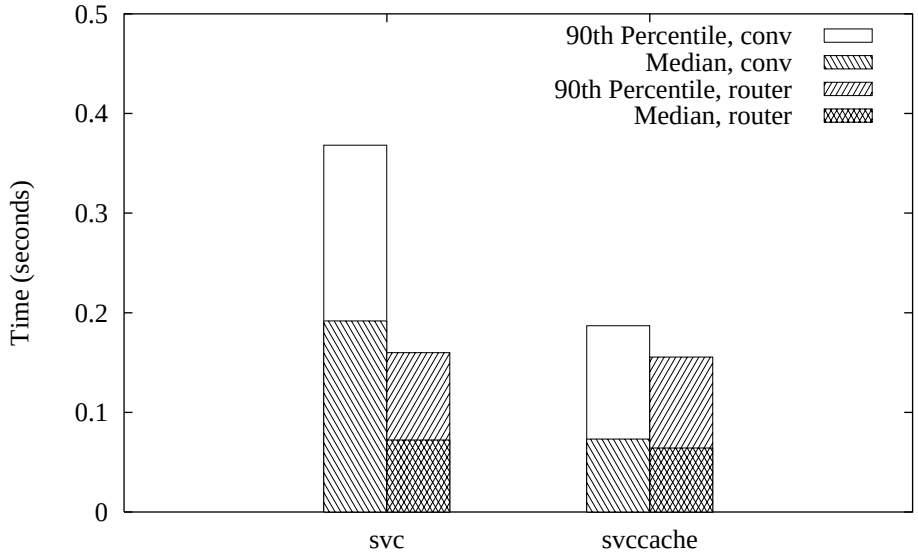


Figure 5-14. Effect of `router` Multiplexing on Telnet Connect Times.

3. Our simulations had six sites, each with a single router; for each router to set up a (unidirectional) virtual circuit to the other five required $n(n-1) = 6(5) = 30$ virtual circuits.

Telnet round-trip times exhibited small, but not statistically significant, reductions of 0.5–12% in the median and 14–15% at the 90th percentile.

5.5.2 FTP

We saw slightly smaller median file transfer times with `router` multiplexing (statistically significant only for the `svc` setup), most likely due to the elimination of most virtual circuit establishments. We present this result in Figure 5-15.

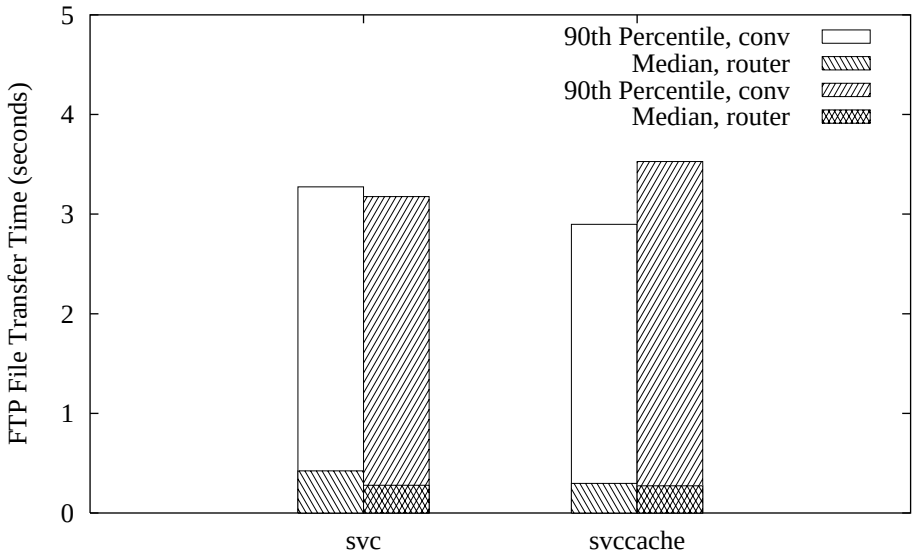


Figure 5-15. Performance Effects of `router` Multiplexing on FTP File Transfer Times.

By contrast, when we enabled virtual circuit caching, we noted longer transfer times for the 90th percentile of individual files, as well as the median and 90th percentile of sessions, when we used per-router multiplexing. Although not constituting a statistically significant difference, this performance degradation was 10–22%. We believe that because so many virtual circuit setups were absorbed by the virtual circuit cache (97% of potential virtual circuit requests were cache hits, as we see in Section 6.6), the use of `router` multiplexing gained little over `conv` multiplexing. However, long transfers felt the effects of competition for buffers along virtual circuits, which, with `router` multiplexing, are shared by many different IP conversations.

We were able to partially validate the latter assertion by examining the drop rates of cells in the ATM switch queues. When using `conv` multiplexing, we saw a cell drop rate (due

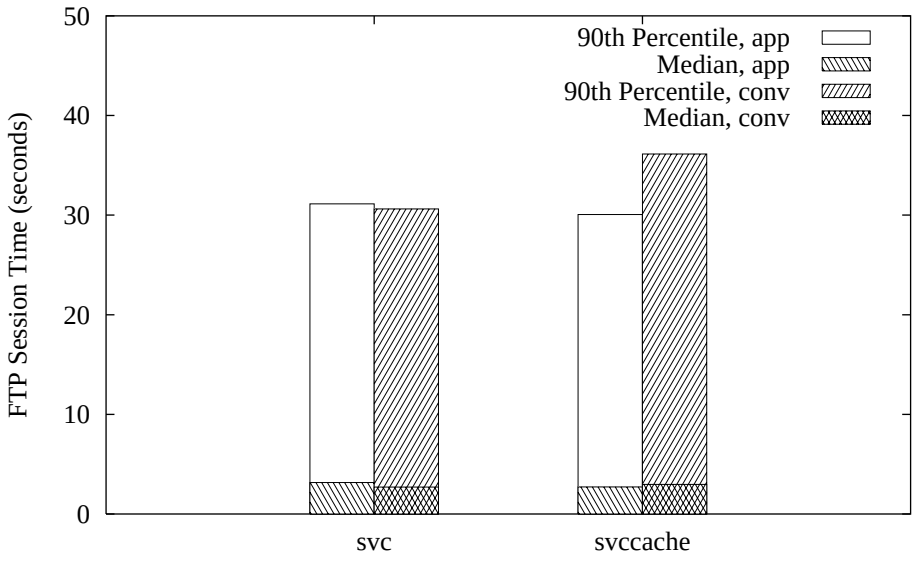


Figure 5-16. Performance Effects of `router` Multiplexing on FTP Session Times.

to queue overflow) of 0.051%, averaged across all switch queues and multiple repetitions. However, when we used `router` multiplexing, the drop rate rose to 0.064%. Although this difference was not statistically significant, removing a single outlying run (with `conv` multiplexing) caused our tests to show that the drop rates were higher with per-router multiplexing, with 90% confidence.

5.5.3 HTTP

The performance of Web clients was affected by router multiplexing in much the same way as that of FTP clients. We saw that short sessions tended to complete faster with `router` multiplexing due to the delay taken for `conv` multiplexing to set up virtual circuits. For the `svc` case, the difference was approximately 140 ms at the median of both file and page retrieval times.

These effects were not present when we enabled virtual circuit caching. As with FTP, long Web files and pages took longer to transfer with router multiplexing, although these differences were not statistically significant. The 90th percentile of HTTP files took 300 ms longer to transfer, a difference of 29%. The 90th percentile of Web pages took 500 ms longer with router multiplexing, a degradation in performance of 13%.

5.5.4 Audio

The audio loss rate was dramatically higher with `router` multiplexing, in all cases. The loss rate, which was less than 0.25% for `conv` multiplexing, jumped to over 1% with `router` multiplexing, an increase of 310–880%. We attribute this effect to the loss of protection between different IP conversations sharing the same virtual circuit (they compete for buffer space in the ATM switches). Figure 5-17 illustrates the differences in loss rates (as well as packet overdue rates).

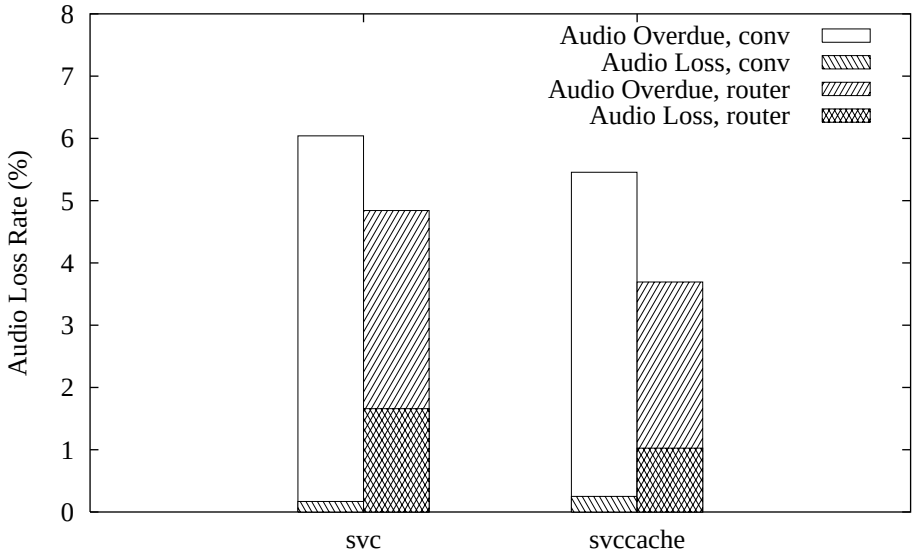


Figure 5-17. Effect of `router` Multiplexing on Audio Loss and Overdue Rates.

The overdue rate (for packets that actually made it all the way to their destinations) was reduced somewhat, but not enough to make a statistically significant difference. We saw reductions of about 20–32%.

5.5.5 Video

Video traffic was not significantly affected by the use of per-router multiplexing. We saw a drop of 19% in the loss rate when `svc` virtual circuits were used. However, when we used `svccache` virtual circuits, switching from `conv` to `router` multiplexing doubled the loss rate (from 1.1% to 2.2%). The exact reasons for these results are not immediately apparent.

5.6 Conclusions

In these experiments, we compared the performance of three different multiplexing policies. The first (`conv`) assigns an ATM virtual circuit to every IP conversation. Another uses an ATM virtual circuit for all traffic of a given application type between a pair of hosts (`app`). The last uses virtual circuits like trunks, carrying all data between a pair of routers (`router`).

We saw that increased levels of multiplexing (aggregating traffic on a per-application or per-router basis) were beneficial to short TCP conversations. For small file transfers, the initial setup (virtual circuit establishment and TCP connection setup) can account for a significant amount of the time to complete the transfer. The elimination of some virtual circuit setups was, therefore, helpful to the performance of these short operations.

However, the performance of long transfers under increased multiplexing suffered in two circumstances. First, `app` multiplexing was detrimental to long transfers, when the ATM schedulers performed rate control. The policing performed by rate controllers allowed bursts of cells for one conversation to delay the eligibility of cells for following, unrelated conversations sharing the same virtual circuit. The result was an erosion of most of the gains made by elimination connection setups; in some cases, `app` multiplexing actually resulted in longer transfer times.

Per-router multiplexing was similarly detrimental to long file transfers, as the high degree of aggregation allowed conversations to compete with each other for buffer space in switches. A particularly hard-hit application was the audio application, which saw a significantly increased loss rate.

These results suggest that a router should attempt to place long conversations on their own virtual circuits, while allowing some degree of aggregation for shorter conversations. For some traffic types, such as audio and video, it is generally safe to make the assumption that the conversation will be long-lived, and thus place them on their own, dedicated virtual circuits.

In some other cases (in particular Web and FTP), it is not immediately clear how to accomplish this classification, given that current IP routers have no idea how long a given IP conversation will last. We feel, however, that it is desirable to perform per-application multiplexing for these applications to get the performance gains for small files, on the assumption that users are more likely to notice performance differences for small transfers.

There are some natural areas for future investigations. Two types of policies, not investigated in this study, would afford some additional flexibility. We have seen that only some traffic types would benefit from multiplexing; it would therefore be useful to investigate some hybrid policies that, for example, perform `conv` multiplexing for audio and video data, while doing `app` multiplexing for Web traffic.

Another type of policy worth examining would be one of a class of dynamic policies that could move conversations between different virtual circuits. An instance of this policy would, for example, be able to move an FTP file transfer from a shared to a dedicated virtual connection after a certain number of bytes had been transferred.

6 Management of ATM Virtual Circuits Used for IP

In this chapter, we investigate policies for the management of ATM virtual circuits used for carrying IP datagrams. This issue arises because of a fundamental difference between the data forwarding models of IP and ATM. Because IP is connectionless, end hosts have no way to indicate the start and end of IP conversations and thus cause ATM connection setups and teardowns. The policies described and evaluated here provide the means for ATM-attached routers to infer an appropriate course of action. We show simulation results demonstrating that, under many circumstances, caching of idle ATM connections can significantly improve the performance of Internet applications as well as reduce the signaling load in the ATM network.

6.1 Introduction

ATM-attached routers need to implement policies for the setup and teardown of virtual circuits because IP hosts have no means of performing (or requesting) these actions themselves. Such policies may be trivially simple, involving a mesh of ATM Permanent Virtual Circuits (PVCs), or more complex, requiring Switched Virtual Circuits (SVCs) to be created and destroyed according to the needs of IP conversations.

The use of performance-guaranteed virtual circuits raises additional problems. For instance, the approach of creating a PVC mesh becomes less attractive in an ATM subnet attempting to provide QOS support for IP traffic. A fixed set of PVCs cannot truly be expected to provide the quality of service suitable for a possibly unknown traffic load. Moreover, a fixed mesh of QOS-guaranteed virtual circuits ties up resources unnecessarily, as the connections are not *a priori* known to be necessary.

An important tradeoff, made more so by the implications of resource reservations, concerns the lifetime of ATM virtual circuits. To reduce the effects of virtual circuit setup time on latency, it may be desirable to cache unused SVCs in the hope that they will be needed again (possibly for a different IP conversation). However, to keep the real-time utilization of the network high, it is important to free up the resources allocated to SVCs as soon as they are no longer needed (releasing resources implies closing a connection, as, in our model, a virtual circuit's resources are associated with it throughout its lifetime). A virtual circuit management policy must attempt to balance these two goals, possibly incorporating the characterizations of individual types of IP conversations as well.

Multiplexing a single virtual circuit among multiple IP conversations introduces additional virtual circuit management issues, with new implications for the fixed timeout strategies suggested in [Cáceres92] and [Claffy94] or the adaptive strategies proposed by [Lund95]. For example, a connection's lifetime may depend on the arrival patterns of traffic for multiple IP conversations, or on the relationships between different IP conversations multiplexed over that virtual circuit.

In Section 6.2, we present some prior work with virtual circuit management policies, both in the form of research studies and actual implementations. Section 6.3 presents the three different virtual circuit management schemes we evaluated in this study. The three following sections describe simulation results (key results are summarized in Table 6-1). In Section 6.4, we present the effects of caching idle SVCs used for IP traffic. We present a similar comparison for the special case of per-router multiplexing in Section 6.5. Section 6.6 examines the behavior of the virtual circuit cache, and its implications for the signaling load on the ATM network. Finally, we present our conclusions in Section 6.7.

For IP-over-ATM policies using per-application and per-connection multiplexing, caching idle ATM connections provides significant improvements in application performance.
For IP-over-ATM policies using per-router-pair multiplexing, there were no significant differences between any of the virtual circuit management policies.
For IP-over-ATM policies using per-router-pair multiplexing, the performance of switched virtual circuits with caching enabled was identical to that with permanent virtual circuits.
Connection caching dramatically reduces the number and frequency of ATM connection establishments, at a nominal overhead.

Table 6-1. Summary of Virtual Circuit Management Results.

6.2 Prior Work

The time to establish a virtual circuit may be relatively long, especially in wide-area backbones with long propagation times. Therefore it may be useful to keep virtual circuits in existence even when they are not actively being used to transmit data. A simple solution that eliminates virtual circuit setup time altogether is to create a mesh of PVCs between all pairs of endpoints. This approach is recommended by both [Cáceres92] and [Claffy94], but has inherent scaling problems in the case of large ATM subnets.

When SVCs are being established and torn down dynamically, it may be possible to amortize the connection setup time by caching virtual circuits in the hope they can be reused for other IP conversations. The utility of such caching is dependent on the arrival and duration of IP packets and conversations, as well as the characteristics of those conversations. [Lund95] describes an adaptive strategy for computing virtual circuit holding times, which involves gathering an empirical distribution of packet interarrivals. The approach is fairly simple to implement and has shown promising results in trace-driven simulations.

The IP-over-ATM implementations in current production ATM LANs typically use either SVCs with static timeouts or PVCs. For example, the FORE System ATM LAN ties the lifetime of virtual circuits to a host's ARP cache, which results in a timeout of fifteen minutes. Thus, any virtual circuit which is idle for longer than fifteen minutes is torn down [Biagioni93].

[Maher95] documents the ATM signaling necessary to support the IP-over-ATM service of [Laubach94]. It recommends that when switched virtual circuits are used, they should employ an idle timeout of at least twenty minutes. However, no justification for this value is given; it seems too long for setups using per-conversation multiplexing, given the short duration of many IP conversations.

6.3 Virtual Circuit Management Schemes

In this work, we examined three different virtual circuit management policies. The first was a simple approach based on PVCs, which we refer to simply as `pvc`. Networks using PVCs to carry IP traffic (such as the XUNET II or BAGNET testbeds) typically have hard-

wired connections which are pre-configured into the network. While this approach does not offer much flexibility, it is simple and avoids any dependence on having a functioning signaling protocol. In our experiments, we simulated PVCs by using SVCs that, once established, were never torn down. We feel that this approximation is reasonable, as these connection setups all took place early in the simulations and their effects were likely masked by other startup transients.

Another policy examined was a simple SVC scheme (`svc`), in which virtual circuits are established on demand and torn down after some period of inactivity. In all these experiments, we set this timeout to ten seconds for all connections. We note that the timeout values we used, for this policy and the following one, were set somewhat arbitrarily to reflect durations we felt to be reasonable. We based the timeouts on our belief that IP conversations (and gaps during them) tend to be short. However, we made no attempt, in this work, to evaluate the benefits of longer or shorter timeouts.)

The third scheme we considered was a variant of the `svc` policy that allows caching of idle connections (`svccache`). SVCs are established on demand as before. After a certain idle time (ten seconds in our experiments), a virtual circuit becomes “unbound” from its IP conversation. For a somewhat longer period of time (300 seconds in our simulations, as configured), the virtual circuit is available to any IP conversation needing the connection (including the one that used the connection originally). If no conversation was able to make use of the unbound SVC by the end of the 300 seconds, it is torn down.

For a new IP conversation to be able to make use of a cached ATM virtual circuit, the virtual circuit needs to have a source and destination appropriate to the conversation. In the case of QOS-aware policies, the cached connection also needs to have an appropriate set of QOS parameters. In INSANE, each virtual circuit is tagged with the type of conversation that originally created it (e.g. telnet up, FTP down, audio). Currently, we require that a virtual circuit have a type identical to that of a new request in order for it to be reused.

6.4 Effects of Caching on Switched Virtual Circuit Performance

In this section, we examine the effects of virtual circuit caching in IP-over-ATM setups using switched virtual circuits. We compared the performance of different Internet applications using two virtual circuit management policies (`svc` and `svccache`) across two different, applicable multiplexing policies (per-application and per-conversation), as well as a variety of QOS policies.

We found that the FTP and HTTP applications showed significant improvements in file transfer time when virtual circuit caching was enabled. Because both applications transfer bursts of (typically) small files, eliminating the overhead to establish an ATM connection was demonstrably beneficial to application performance. The multimedia applications (audio and video) showed performance improvements in only a few cases, probably because their conversations were relatively long-lived.

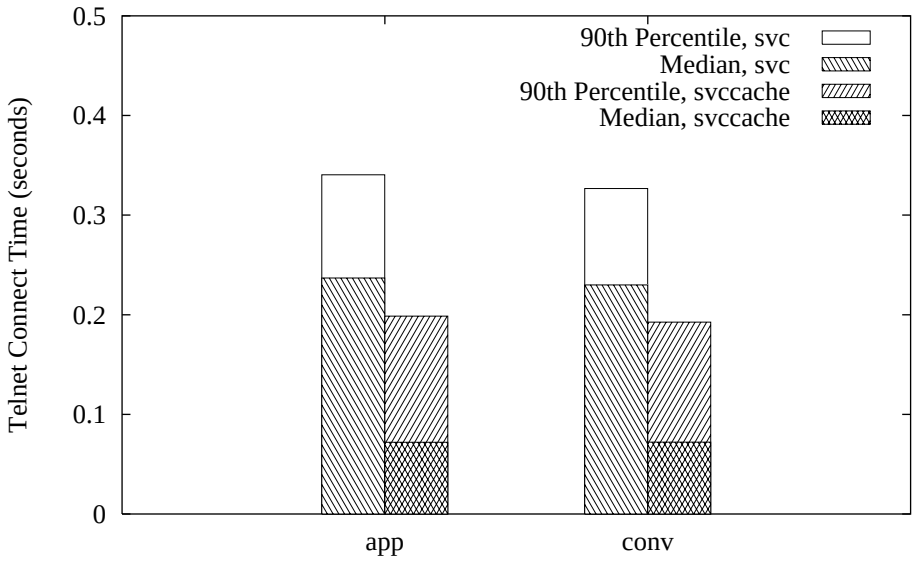
6.4.1 Telnet

We found that, in almost all cases, the use of a cache of idle virtual circuits significantly reduced the connect time for new telnet conversations. This fact is unsurprising, since the telnet connect time was entirely dependent on the overhead of setting up a TCP connection. The `wc-telnet` scenarios were typical, in which we saw the setup times decrease by 158–165 ms (69–70%) at the median and 134–142 ms (41–42%) at the 90th percentile. This effect is shown in Figure 6-1.

In comparison, the effects on telnet round-trip times were less visible. Only a few setups exhibited significant changes to the distributions of response times; all those that did, however, demonstrated that the use of `svccache` virtual circuits was successful in improving telnet performance. Figure 6-2 shows the effects in the `nwc-telnet` case, which showed a 90 ms improvement (3–8%) at the median. This represented a significant difference in the case of per-conversation multiplexing, but not `app` multiplexing.

6.4.2 FTP

In a variety of circumstances, FTP performance was significantly improved when we enabled ATM connection caching. A simple case was the `sp-ftp` setup, whose results are



igure 6-1. Performance Effects of Virtual Circuit Caching on Telnet Connect Times, `wc-telnet` QOS Policy.

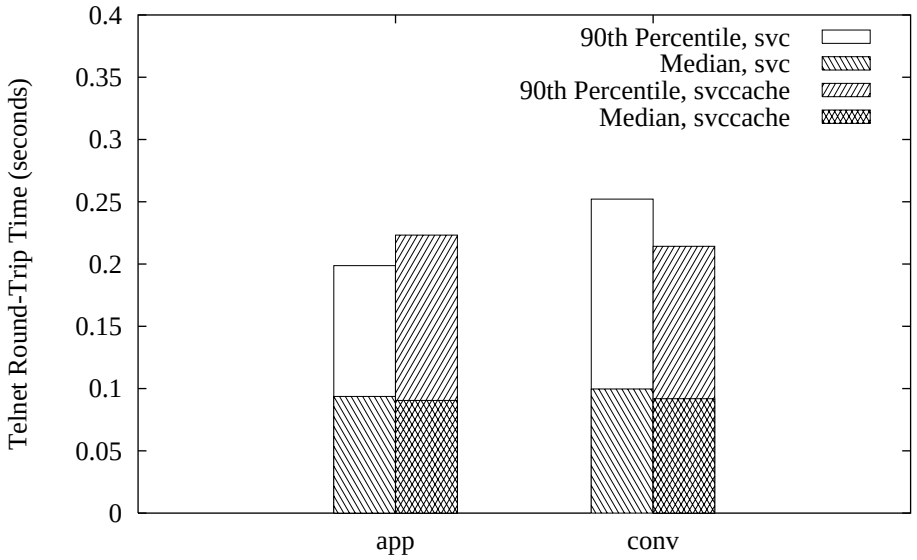


Figure 6-2. Performance Effects of Virtual Circuit Caching on Telnet Round-Trip Times, `nwc-telnet` QOS Policy.

pictured in Figure 6-3 and Figure 6-4. In these scenarios, the median file transfer time and 90th percentile of session times showed statistically significant improvements of 13–17% (34–61 ms) and 14–20% (298–466 ms), respectively.

Significant differences surfaced primarily when we used `conv` multiplexing. In the `wc-ftp` setup, for example, file transfer times decreased by 27% at the median (0.13 seconds.

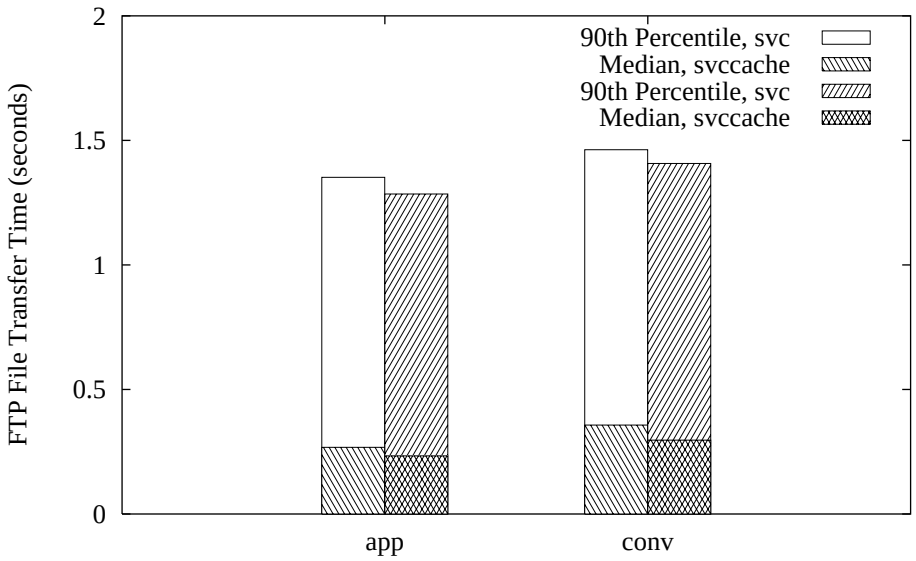


Figure 6-3. Performance Effects of Virtual Circuit Caching on FTP File Retrieval Times, `sp-ftp` QOS Policy.

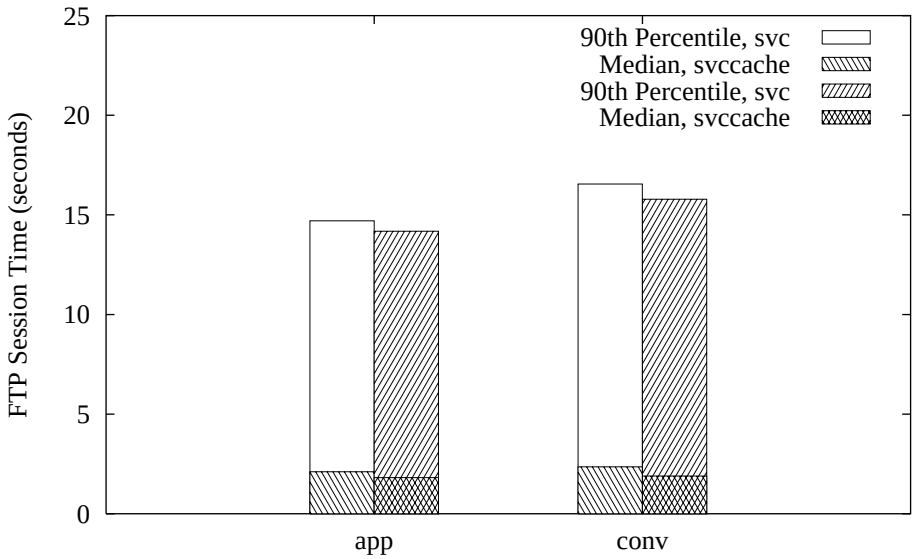


Figure 6-4. Performance Effects of Virtual Circuit Caching on FTP Session Times, `sp-ftp` QOS Policy.

The median session time was shortened by 1.8 seconds (a 10% improvement). All of these gains represented statistically significant differences. The differences with `app` multiplexing were not statistically significant, and had much smaller values (the median file and session times improved by only 0.03 and 0.87 seconds, respectively). We illustrate the contrast in session completion times in Figure 6-5. These results seem reasonable, as we

would expect that the connection caches would be more effective (and more necessary) without aggregation of multiple IP conversations into the same virtual circuit.

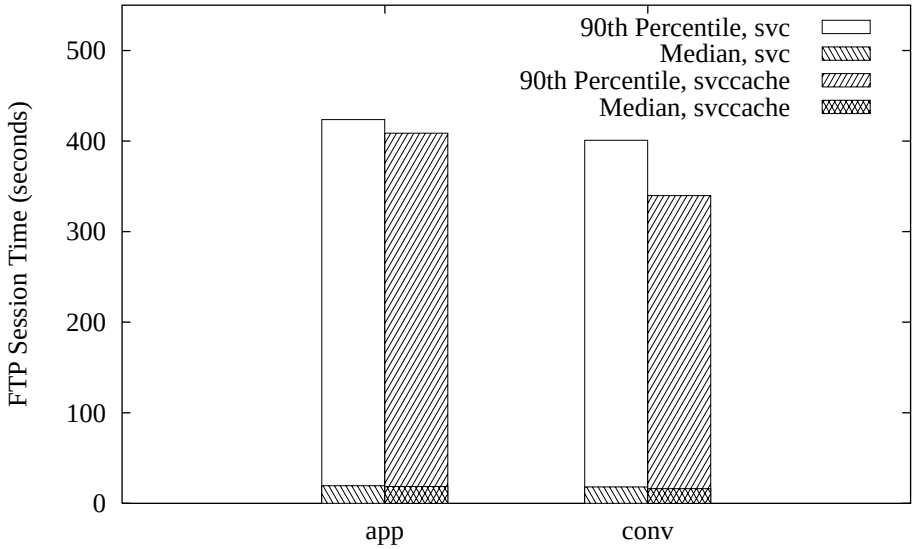


Figure 6-5. Performance Effects of Virtual Circuit Caching on FTP Session Times, `wc-ftp` QOS Policy.

6.4.3 HTTP

As with FTP, application-level Web performance was generally improved by the use of virtual circuit caching. One of the more apparent improvements was in the `nwc-http` setup, which saw statistically significant differences in item and page transfer times for both `app` and `conv` multiplexing. Median item transfer times improved by 120–259 ms (16–36%) and the median page times improved by 299–383 ms (22–32%). We show these gains in Figure 6-6 and Figure 6-7.

We note that, in the cases of both HTTP and FTP, we observed no interference between different conversations using a cached virtual circuit, as we did with `app` multiplexing in Chapter 5. We recall that, with the rate-controlled service disciplines, bursts from long bulk transfers could cause subsequent transfers to be delayed, even if they were not simultaneous. There were no analogous effects with `svccache`, because the ten-second idle timeout (before a virtual circuit could be reassigned to a new conversation) was enough for the rate controllers to "forget" about the effects of prior traffic.[1]

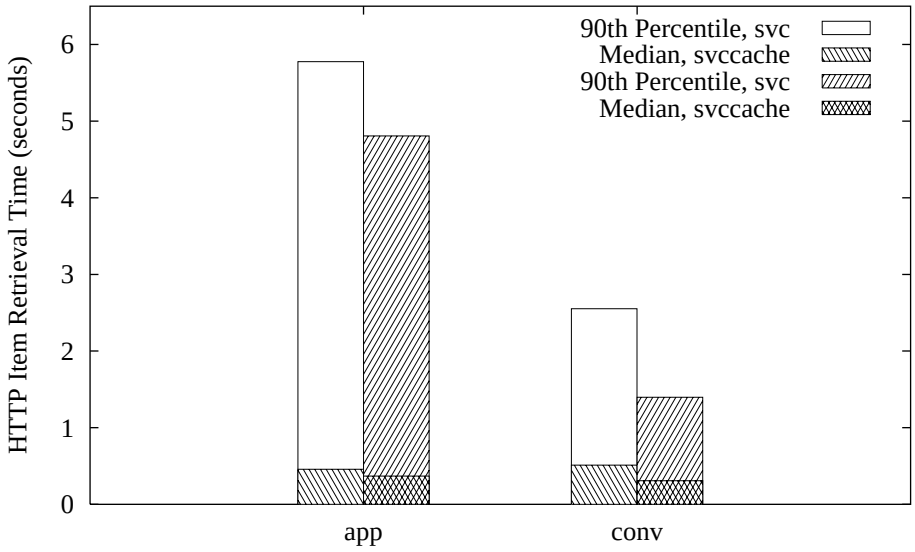


Figure 6-6. Performance Effects of Virtual Circuit Caching on HTTP Item Retrieval Times, `nwc-http` QOS Policy.

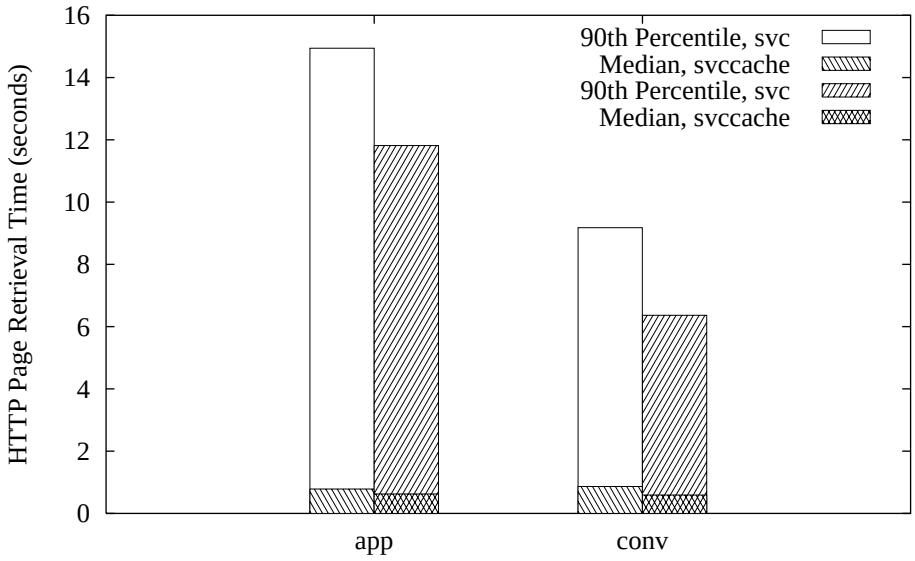


Figure 6-7. Performance Effects of Virtual Circuit Caching on HTTP Page Retrieval Times, `nwc-http` QOS Policy.

We note that Persistent Connection HTTP (P-HTTP) [Padmanabhan94] reduces the demand for TCP connections, because it can transfer multiple Web files over a single TCP connection. This protocol would probably diminish the benefits of connection caching,

1. The implementation of INSANE's RCSP rate controllers allows for each queue to maintain up to two seconds' worth of buffering and rate control information. Ten seconds of idle time is more than enough to flush all the state from the rate controllers.

because each Web page would require fewer TCP connections, and hence present fewer opportunities for caching to improve performance. We did not, in this study, experiment with the effects of a P-HTTP application. Our intuition, however, says that the combined deployment of P-HTTP and connection caching would reduce file and page times at least as much as either of them used separately.

6.4.4 Audio

For the most part, we observed no significant effects in audio performance arising from the use of virtual circuit caching.

In some cases, however, we did observe large, statistically significant reductions in the audio loss and overdue rates when we used the `svccache` policy. One such class of setups was formed by those using the `sp-qos1` QOS policy, in which we saw a loss rate of 1.11–1.47% decrease to 0.76–0.78% with the introduction of caching. The overdue rates similarly decreased from 2.71–3.38% to 1.42–2.08%. We show this improvement in Figure 6-8.

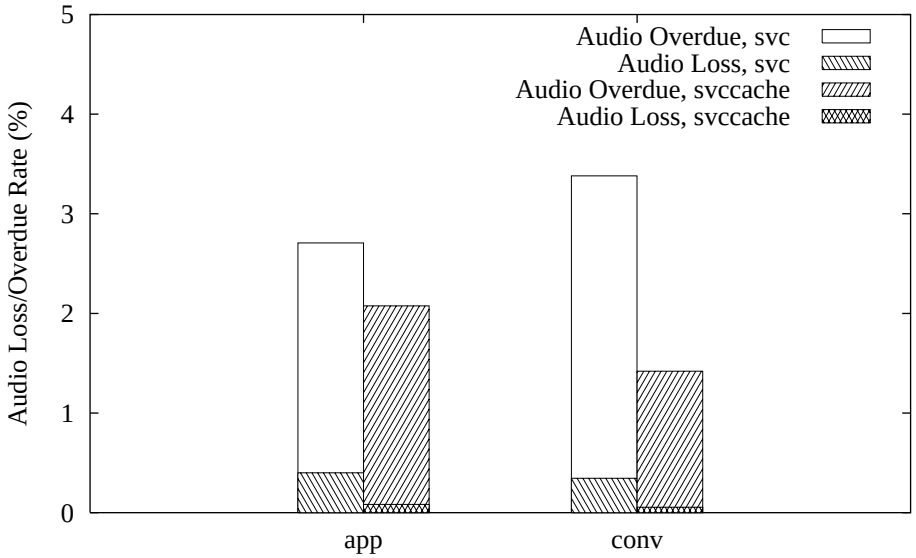


Figure 6-8. Performance Effects of Virtual Circuit Caching on Audio Loss and Overdue Rates, `sp-qos1` Policy.

The setups using the `sp-av` and `wc-http` QOS policies showed similar improvements with caching. The reasons for these performance gains are not immediately apparent. Intuitively, we would not expect audio performance to be significantly affected by the use of

the virtual circuit caching policy because its effects at connection setup time should influence only a small part of the lifetime of any one audio conversation.

6.4.5 Video

The enabling of virtual circuit caching had a similar effect on the performance of the video applications. In most cases, the `svccache` policy offered no statistically significant improvements over the `svc` policy. However, in a handful of cases, we observed lower loss rates when connection caching was used, together with the `conv` multiplexing policy. For example, in the `sp-ftp` scenarios, we saw the video loss rate decrease from 4.64–8.11% to 4.09–4.47%, as shown in Figure 6-9.

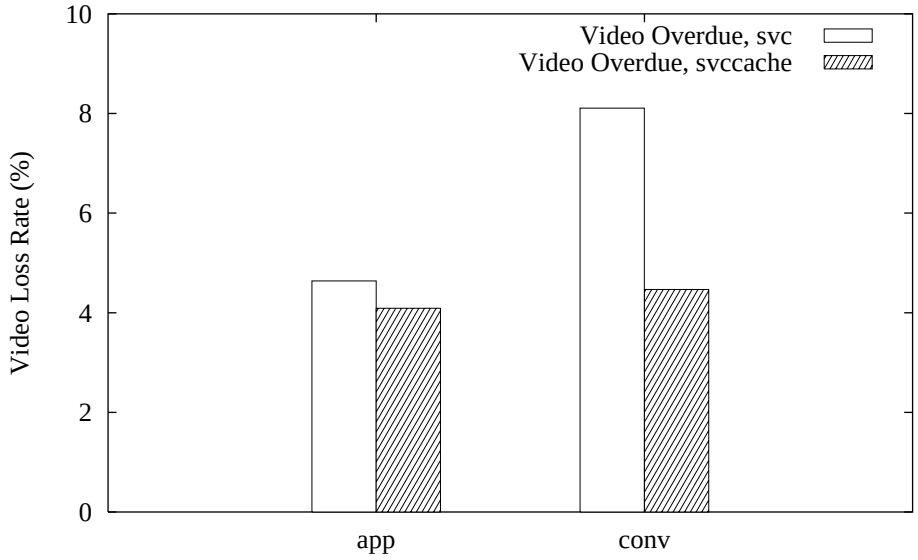


Figure 6-9. Performance Effects of Virtual Circuit Caching on Video Loss Rate, `sp-ftp` QOS Policy.

As with improvements to audio performance, the reasons for these differences are not immediately clear. One possible explanation may lie with the lower amount of signaling traffic (sent at a higher priority than the default, best-effort data). Few signaling messages can cause data sent with the default QOS to receive better treatment, because there is less competition from higher-priority signaling traffic.

6.5 The Special Case of Per-Router Multiplexing

This section examines the effects of connection caching in scenarios that used per-router multiplexing. The experimental results were not terribly interesting, in that connection

caching provided few benefits. We somewhat expected this outcome. Given the fact that connection establishments under per-router multiplexing are rare events, it is difficult for different virtual circuit management policies to have much impact on application performance.

We note that with `router` multiplexing, the total stream of aggregated traffic sent over any given virtual circuit contained few idle periods longer than the ten-second idle timeout for teardown (`svc`) or caching (`svccache`). From Section 5.5, we recall that, for any scenarios using `router` multiplexing and `svc` VCs, we saw at most 18 more connections than were needed for full connectivity, over the course of an entire simulation. Thus, there were at most 18 idle virtual circuit timeouts over the 4000-second run, very few opportunities for any differences in virtual circuit management policies to have an influence.

This fact can account for two results. First, the scenarios using `pvc` and `svccache` policies yielded exactly identical performance for all metrics. As explanation, we note that virtual circuits were never idle long enough in the caching scenarios to be torn down. In all cases, another packet would arrive to reactivate a cached connection before it was torn down (i.e. within five minutes). Thus, except for a small amount of bookkeeping in routers (whose effects on network performance were not modeled), the entire network behaved identically to the case of permanent virtual circuits.

The second result was that there were no significant performance differences among any of the virtual circuit management policies when we used per-router multiplexing. Given the results from Section 5.5, this is hardly surprising. There were simply not enough periods of idle time (and subsequent virtual circuit teardowns) for caching (or the use of PVCs) to effect much improvement over the case of uncached SVCs.

6.6 Cache Effectiveness

The virtual circuit cache returned surprisingly high hit rates. It decreased both the number and rate of virtual circuit setups required by more than an order of magnitude. Although the cache imposed a certain overhead in terms of additional virtual circuits required, we judged this cost to be small.

For the noqos setups (where all virtual circuits were treated identically for the purposes of being able to carry traffic), we observed cache hit rates of 97–98%. For the static priority schemes and most of the QOS policies using any form of RCSP, the virtual circuit caches in the routers were able to absorb 95% of the virtual circuit setups with app multiplexing and approximately 92% with conv multiplexing.

Some of the QOS policies (wc-isp, wc-qos1, nwc-isp, and nwc-qos1) approached or surpassed the capacity of the ATM network to support guaranteed connections, which resulted in a large number of virtual circuit establishment failures.[2] Because there was not a large pool of idle virtual circuits to support new real-time connections, the virtual circuit cache was significantly less effective. The hit rates in these scenarios were 65–69% for app multiplexing and 56–59% for conv multiplexing.

An important implication of the high cache hit rates is that it drastically reduces the *rate* of signaling requests made (by ATM-connected routers) by one or two orders of magnitude. The ability to reduce the number of virtual circuit setups can be crucial; [Schmidt93] and others have cited high signaling rates as potential bottlenecks for high-speed ATM networks.[3] To investigate this effect, we collected the number of virtual circuit establishments performed each second at two of the six routers[4], as well as totals across all routers in the network.

In our simulation runs, noqos-conv-svc scenarios required an average of 168,936 virtual circuits. Over a 4000-second simulation run, the six routers in our network therefore performed an average of 7.0 establishments per second. The two busiest routers recorded

---

2. We recall that a virtual circuit has resources allocated to it throughout its lifetime, even when idle. These resources, such as scheduling priority and bandwidth, are unavailable to other guaranteed-performance connections, although they may be temporarily exploited by best-effort traffic if unused by their "owning" connection.

3. Although these figures are not necessarily indicative of current ATM WAN virtual circuit setup performance, we note that XUNET II needed 300 ms (!) per hop to establish a virtual circuit. An ATM LAN based on an early version of the Synoptics (now Bay Networks) LattisCell 10114 switch took 70 ms to establish a switched virtual circuit over a one- or two-hop path.

4. Due to the Zipf's Law distribution for selecting FTP and Web servers in the user workload generator, the distribution of traffic is not symmetric. We elected to measure the call setup rates at the sites containing the two most common servers, on the rationale that this choice would give us the "worst case" (i.e. highest) call setup rates.

a peak call setup rate of 37 calls per second. By contrast, the `noqos-conv-svccache` setups required only 4337 virtual circuit setups, on average. This translates to an average setup rate of 0.18 establishments per second per router, with an observed peak of 25 calls per second.

The effect with per-application multiplexing was similar. The `noqos-app-svc` scenarios required, on average, 71,136 connection setups (3.0 establishments per second per router). When we enabled virtual circuit caching, we observed an average of 1424 establishments per run (each router performed an average of 0.06 setups per second). However, the peak rate of connection establishments was the same regardless of whether virtual circuit caching was in use: 15 connections per second.

Another quantity of interest when evaluating the usefulness of virtual circuit caching is the maximum number of virtual circuits in use at any one time. A large cache may be impractical if it requires a large number of idle circuits to be useful. Thus, we measured the maximum number of virtual circuits used by each of the routers, sampled at one-second intervals.

In the `noqos` scenarios using per-application multiplexing, we observed a maximum of 162 connections in use at any time with no caching and at most 200 connections used with caching enabled, an increase of 24%. To run the per-conversation multiplexing scenarios, we needed a maximum of 341 virtual circuits from a router with no virtual circuit caching and 472 virtual circuits with caching enabled, a “cost” of 38%.

Although we did not keep statistics on virtual circuit usage in the ATM switches, we note that we configured our switches to support a maximum of 8192 virtual circuits per port, and that we saw no evidence of any call setup failures due to a lack of virtual circuit identifiers.

6.7 Conclusions

In this chapter, we examined the effects of three ATM virtual circuit management policies on end-to-end Internet application performance. We looked at a policy using permanent

virtual circuits (`pvc`), a simple switched virtual circuit policy (`svc`), and a variant which caches idle switched virtual circuits for later reuse (`svccache`).

We found that caching idle ATM connections provides significant improvements in application performance, in scenarios using per-application and per-conversation multiplexing. We observed no significant effects in setups using per-router multiplexing; however, we found that, with the static timeout values we chose, the operation of the network with cached SVCs was identical to that with PVCs. Finally, we examined the signaling load at some of the ATM-attached routers and saw that connection caching dramatically reduced the number of call setups required, in exchange for a small number of additional virtual circuits.

Based on these results, we believe that circuit caching should be implemented in IP-over-ATM networks using switched virtual circuits. Given the short duration of many IP conversations, the elimination of virtual circuit setup overheads can significantly improve end-to-end application performance and reduce the signaling load on the ATM network.

We note several possible areas of further study. A natural issue to raise is that of the timeout values used to teardown virtual circuits (or to cache them, when appropriate). While we only used a fixed (“reasonable”) set of values, it would be useful to investigate a range of different timeout values. We feel, in particular, that the current timeout values are longer than necessary for many bulk transfer applications such as FTP or HTTP. Telnet, however, may benefit from longer timeouts, so that virtual circuits are not torn down during periods of “user think time”. This observation raises the possibility of setting timeouts on a per-traffic-type basis, or a dynamic scheme such as that proposed in [Lund95].

Finally, keeping a cache of idle virtual circuits raises some pricing issues, which we did not address in this study. In particular, it is not clear who should “pay” for resources, such as network bandwidth and scheduling priority, which are allocated to idle, cached connections, and thus unavailable for other guaranteed connections. Modifying or releasing the resources allocated to an idle connection might reduce the penalties associated with caching. However, this would require a somewhat richer interaction with the ATM network than our models currently support.

7 Conclusions

In this dissertation, we have investigated three issues in the design of IP-over-ATM systems. We have examined various alternatives for policies addressing these issues, and evaluated their impacts on end-to-end Internet application performance, via a large-scale network simulation.

In Section 7.1, we summarize the findings and contributions of this research. We present some possible areas for future work in Section 7.2. Lastly, we present some final remarks in Section 7.3.

7.1 Summary of Contributions

In Chapter 1, we motivated our research by noting the growing popularity of Asynchronous Transfer Mode (ATM) networks, and the desire to use them as an effective part of the global Internet, running the Internet Protocol (IP).

We provided some background on both IP and ATM in Chapter 2. We outlined some of the contrasts between the two types of networks, in particular the connection models, the differing support for quality of service and performance guarantees, and the differences in types of packets. From the resulting research issues, we sketched out a space of possible IP-over-ATM policies to be implemented by ATM-attached hosts and routers. This space consists of policies for using ATM quality of service, multiplexing, and connection management to support Internet applications.

Chapter 3 described our methodology. We outlined the set of IP-over-ATM policies under investigation. Next we described a set of simulation experiments, which measured the performance of common Internet applications (expressed using metrics such as file transfer

times or packet loss rates). These measurements were performed on applications as they ran in a large, simulated IP internetwork with an ATM backbone. We showed how we varied the IP-over-ATM policies to investigate their effects on application performance. Finally, that chapter described the design and implementation of a new network simulation tool, the Internet Simulated ATM Networking Environment (INSANE).

In Chapter 4, we examined a variety of ATM scheduling disciplines, including First-Come-First-Served, Static Priority, and two variants of Rate-Controlled Static Priority. We constructed a number of policies for using these scheduling disciplines to give preferential treatment to different Internet applications, and measured the end-to-end performance effects. We found that a Static Priority scheduler can be effective at giving preference to any application, but at the risk of starving out low priority traffic. We looked at the effects of RCSP's traffic policing, and found that, although it could prevent applications from monopolizing network resources, the benefits derived by other applications were uncertain. Finally, we saw that distributed jitter control (such as that provided by RCSP scheduling) was useful in controlling losses in long TCP bulk transfers. From these results, *we conclude that a Static Priority scheme (if used carefully) may be the most effective in offering differential treatment of various traffic types.*

We examined three different IP-over-ATM multiplexing policies in Chapter 5. Each aggregated increasing amounts of traffic onto individual ATM virtual circuits. We saw the performance of small file transfers improve with aggregation, due to the elimination of virtual circuit setup overheads. Long files, however, fared better on their own individual connections in networks that performed traffic policing, due to an undesirable interaction between different transfers sharing the same ATM connection. Finally, we saw that, at very high levels of aggregation, contention for buffers increased packet drops for loss-sensitive data such as audio. *At least for short transfers, IP conversations should be aggregated onto common virtual circuits, using a policy such as per-application multiplexing. The optimum strategy for longer transfers depends somewhat on the scheduling policy in use in the ATM network, but in general placing them on their own connections is preferable.*

Finally, in Chapter 6, we looked at three policies for managing ATM virtual circuits being used to carry Internet traffic. The first two policies used ATM SVCs to carry IP traffic. In the first, virtual circuits were created on demand and torn down when idle. In the second, ATM-attached hosts and routers kept a cache of idle connections, which could be reused for other, potentially unrelated IP conversations. We found that in networks doing either per-application or per-conversation multiplexing, applications benefited from the use of virtual circuit caching. Moreover, connection caching was beneficial to the network as well, as it dramatically lowered the volume and frequency of signalling traffic. From these results, *we recommend ATM networks using switched virtual circuits implement connection caching*.

The last virtual circuit management policy we examined used Permanent Virtual Circuits, which are useful in networks using per-router-pair multiplexing. We found that in such networks, there were no significant differences in application performance between any of the virtual circuit management policies we studied.

7.2 Future Work

There are, of course, many areas for future work in the area of IP-over-ATM systems. We touch on some of them briefly in this section.

As the Internet continues to evolve and grow, so will its workload. New applications continue to be deployed, each with their own traffic patterns and characteristics. Simulations or analysis of future networks will need to take these developments into account when constructing a workload to be used for evaluation purposes.

In addition, new network protocols will have some implications on our work. For example, IPv6 includes support for “flows”, which can be used to identify a stream of related packets at the network layer. Conceivably, this information could be used in our scheme to identify particular IP conversations with less overhead and more reliably than our current scheme, which is based on using port numbers and other higher-layer identifiers.

Real-world ATM networks will likely have their own idiosyncracies and bottlenecks. Studies targeted towards the characteristics of a particular network may yield slightly dif-

ferent (but hopefully not too different) results from those obtained in our idealized ATM environment.

Each of the three design issues we investigated has possibilities for future investigation, as well. Our examination of different QOS policies could be extended to include a study of different sets of traffic parameters for each type of application. Some other methods of specifying performance guarantees could be attempted, including measurement-based schemes in routers or the use of an explicit signalling protocol such as RSVP.

There are other multiplexing policies that could be investigated, beyond those we studied here. It would also be useful to investigate the implications of having some hybrid policies in a network (for example, aggregating many short conversations together, but sending the data for long conversations on their own connections).

Finally, a study of different timing constants for our virtual circuit management policies would be useful—either sets of static timeouts (perhaps set on a per-application basis) or a dynamic timeout scheme. As with multiplexing, it could be interesting to investigate the effects of hybrid schemes. One example would be to send short conversations on permanent virtual circuits, but to send long conversations on their own, dedicated, on-demand connections (assuming a router could determine the length of a conversation in advance).

7.3 Some Final Remarks

While the evaluations performed in this research yielded some initial results and guidelines for IP-over-ATM policies, the utility of these policies will ultimately depend on the traffic workload and administrative policies of each individual site. We believe that, in order to gain maximum benefit from the implementation of these policies, vendors implementing them should provide each site the ability to define and evaluate the policies to be used on that particular site’s networks.

We believe that IP and ATM networks can interoperate effectively; the issues we addressed in this research explored the space of possible policies governing their interactions. The policies we investigated were designed to try to gain the benefits of each type of network, while minimizing their respective weaknesses. We believe that similar opportu-

nities exist for other situations in which two dissimilar networks meet, and must be made to work together.

A An Empirical Model of HTTP Traffic

We have developed an empirically-derived model of HTTP network traffic, designed to provide a synthetic workload to a simulation of a wide-area IP internetwork. This model captures a variety of aspects of World Wide Web network activity. At the lowest level, it describes the sizes of individual Web files; these files combine to produce multi-file documents, separated by "user think time". At the highest level, our model describes the browsing behavior of users, both within a visit to a single Web server and between different Web servers. Our model is based on network packet traces, and uses analysis and heuristics to derive information about files and higher-layer units of information.

A.1 Background

The World Wide Web (frequently shortened to *WWW* or *Web*) is a collection of documents and services available to the global Internet. Servers furnish these documents on request to clients (also known as *browsers*). Each document (sometimes called a *page*) may consist of a number of files. For example, a multi-file document could consist of text represented using the Hypertext Markup Language (HTML) [Berners-Lee95], along with some number of images to be displayed "inline" with the text.

The Hypertext Transfer Protocol (HTTP) [Berners-Lee96] is a request-response protocol for transferring the files making up the parts of Web documents. Each transfer consists of the client requesting a file from the server, then the server replying with the requested file (or an error notification). Both the request and reply contain identification and control information in headers. HTTP uses the services of TCP [Postel81b] for reliable transport across the unreliable global Internet. In current versions of HTTP, each TCP connection can be used for at most one HTTP retrieval. Future versions of HTTP, as described in

[Fielding96], incorporate the work of [Padmanabhan94] and [Mogul95], which propose the reuse of TCP connections for multiple retrievals between the same client and server.

We will occasionally take several liberties with terminology. Strictly speaking, Web documents can be transferred by means other than HTTP. For example, the File Transfer Protocol FTP [Postel85] can be used to serve documents where HTTP cannot be deployed for administrative reasons and FTP servers exist already. Thus, the terms “Web server” and “HTTP server” are not strictly synonymous, though we will frequently use them interchangeably. Our usage of the terms “Web browser” and “HTTP client” is similar. Contexts in which differentiation is required should be easily apparent.

A.2 Prior Work

In this section, we summarize three approaches that have been taken in attempting to characterize Internet applications. Two methods, server logs and client logs, have been used in prior investigations of the World Wide Web. The last approach, traffic traces, has been used for past studies of a number of other Internet applications, such as file transfers and remote logins.

A.2.1 Server Logs

Most Web servers keep logs of the requests and files they have served, for reasons ranging from operational monitoring to collecting demographic information. A workload model can be created by processing the logs of a running Web server. In some sense this approach is the easiest to take, because the machinery for collecting data already exists and, in fact, the data is very likely being collected anyway. Indeed, for some studies, such as [Mogul95] and [Arlitt96], it is appropriate to model a stream of HTTP requests arriving at a Web server.

However, there are two principal drawbacks to this approach. One large disadvantage of using server logs is that they cannot easily capture user access patterns across multiple Web servers. In particular, it may be difficult to make any determination about the locality of references during any given user session. Another shortcoming is that current server log formats do not capture any aspects of HTTP overheads, such as protocol headers.

A.2.2 Client Logs

[Crovella96], [Cunha95], and [Catledge95] relied on data gathered by instrumenting the NCSA Mosaic browser [Mosaic95] to log all retrievals made during Web user sessions. The instrumented systems were in public computing laboratories in academic environments. These studies were primarily concerned with investigating various characteristics of Web accesses. However, based on these types of measurements, it would be possible to construct a corresponding model, suitable for generating a synthetic workload.

Unlike server logs, this approach captures user accesses between multiple Web servers quite well. In addition, it allows the characterization of the effects of client-side caching of documents (or parts of documents). However, this technique requires that browsers be able to log their requests or, more likely, the availability of source code for the Web browser so that such logging can be added. Source code for newer Web browsers, including the popular Netscape Navigator [Netscape96], is generally not available. In addition, supporting a variety of browsers would be difficult if modifications for logging needed to be made to each one.

A.2.3 Packet Traces

Another method of gathering workload data consists of analyzing packet traces taken from a subnet carrying HTTP traffic, typically an Ethernet or other broadcast-style LAN.[1] From the packet traces and knowledge about the higher-layer protocols used, traffic analysis can yield a model of the behavior of the original application. This approach has been used in a number of other traffic studies, such as [Cáceres91] and [Paxson91], that predate the Web. [Stevens96] analyzes the packets arriving at an HTTP server and presents some interesting statistics and observations. [Danzig91] describes a library of traffic models for common (circa 1991) Internet applications, which, in fact, we used for several of INSANE's other simulated applications. [Paxson94a] additionally describes analytic models derived from traffic traces, which have a more compact representation than purely empirical models and can be parameterized to reflect specific networks more accurately.

1. It is possible to use this methodology on a point-to-point link acting as a transit network, but such opportunities are less common.

This approach eliminates the principal disadvantages of the two previous methods mentioned. However, it too introduces drawbacks. We recall that models based on application-level logs can easily record higher-level information such as specific files requested, HTTP message types, and document types. While such information could in principle be gleaned from a packet trace, it would involve considerable effort in reconstructing the contents of each TCP connection. In addition, the effects of client caching of documents are more difficult to ascertain, since only cache misses generate network traffic detectable by a packet trace.

A.3 Methodology

We chose to use a packet trace-based approach for our model, principally because it allowed us to capture the behavior of individual users and we would be able to use this methodology with any popular, currently-deployed HTTP clients. While this approach loses higher-level information such as the actual files accessed, we felt that such a characterization is not essential to a network workload model.

We used the freely-available `tcpdump` packet capture utility [Jacobson95] running on a DEC Alpha 3000/300 to record packet headers on a shared 10 Mbps Ethernet in the Computer Science Division at the University of California at Berkeley, during four periods in late 1995. This procedure saved the TCP and IP headers of each packet, as well as a small number of payload bytes. These data were saved to disk for off-line processing.

The subnet examined is a stub network (no transit traffic), one of a dozen or so in use in the Computer Science Division. There are approximately one hundred hosts on this subnet; the majority of them are desktop UNIX workstations, each principally used by a single user. The user community consists primarily of Computer Science graduate students. While no statistics are available on the relative popularity of different Web clients used in this environment, operational experience suggests that the prevalent one is Netscape Navigator [Netscape96]. There are also several Web servers on this subnet, associated with various research groups.

Most HTTP servers bind to a well-known TCP port (port 80).[2] By looking for all TCP packets to or from this well-known port, we captured what we believe is the vast majority of HTTP traffic. Table A-1 summarizes our traffic traces. The first three traces were collected as a part of an effort to examine various types of network traffic (not just HTTP traffic); the packet counts from these traces include only those packets attributable to HTTP. The last traffic trace collected HTTP packets only. From these streams of packets, we extracted those comprising HTTP connections originating from clients on the tracing subnet.

Start Time	**End Time**	**Number of HTTP Packets**
Tue Sep 19 16:12:33 1995	Thu Sep 21 07:53:22 1995	186068
Wed Oct 11 09:48:53 1995	Thu Oct 12 14:10:16 1995	458264
Wed Nov 1 11:22:47 1995	Thu Nov 2 10:53:12 1995	369671
Mon Nov 20 11:13:36 1995	Sun Nov 26 05:28:17 1995	676256

Table A-1. Summary of Traffic Traces.

Although we do not have complete packet loss figures for these traces, we did record the loss of approximately 6000 out of 44,000,000 packets during the 1 November 1995 trace (before filtering to isolate HTTP packets). These figures yield a packet loss rate of only 0.014%. Similar packet capture experiments using this hardware and network have produced figures consistent with this loss rate.

A.4 Model

Our model of HTTP traffic captures logically meaningful parameters of Web client behavior such as files sizes and user "think times". The traffic traces described in the preceding section provided us with empirical probability distributions describing various components of this behavior. We used these distributions to determine the characteristics of a synthetic workload. In this section, we present the various components of our model, which are summarized in Table A-2.

2. In a recent study of the characteristics of HTML documents indexed by the Inktomi "Web crawler", approximately 94% of the documents surveyed were accessed using the normal TCP port 80 [Woodruff96].

Quantity	Units	Description
request length	bytes	HTTP request length
reply length	bytes	HTTP reply length
document size	files	Number of files per document
think time	seconds	Interval between retrieval of two successive documents
consecutive document retrievals	pages	Number of consecutive documents retrieved from any given server
server selection	server	Relative popularity of any Web server, used to select each succeeding server accessed

Table A-2. Quantities Modeled.

At the lowest level, our model deals with individual HTTP transfers, each of which consists of a single request-reply pair of messages. In the most common case, the client application sends a request for some data; the server in turn replies by supplying that data. The first two quantities of our model are therefore the *request length* and *reply length* of HTTP transfers.[3] The request and reply are both transmitted over a single TCP connection [Berners-Lee96].

At first glance, it may seem more appropriate for a model of network traffic to concern itself instead with the number, size, and interarrival times of TCP segments. However, we note that, in particular, packet interarrival times are governed by the TCP flow control and congestion control algorithms. These algorithms depend in part on the latency and effective bandwidth on the path between the client and server. Since this information cannot be known *a priori*, we conclude that an accurate packet-level network simulation will depend on a simulation of the actual TCP algorithms. This is in fact the approach taken for other types of TCP bulk transfers in the traffic model described in [Danzig91]. In a similar fashion, our model generates transfers that need to be run through INSANE's TCP algorithms; the model does not generate packet sizes and arrival times by itself.

Web documents can consist of multiple files. Thus, a server and client may need to employ multiple HTTP transactions, each of which requires a distinct TCP connection, to transfer a single document. For example, a document could consist of HTML text [Berners-Lee95],

3. [Mah97] presents a slightly more sophisticated model, which describes the request and reply lengths of the first HTTP transfer on any Web page separately from that of any remaining retrievals for that page.

which in turn could specify three images to be displayed “inline” in the body of the document. Such a document would require four TCP connections, each serving one HTTP request and reply. The next higher level of behavior above individual files is naturally the Web document, characterized in terms of the *number of files* needed to represent a document.

Between Web page retrievals, the user is generally considering her next action. We admit the difficulty of characterizing user behavior, due to its dependency on various human factors beyond the scope of this study. However, we can construct a distribution of *user think time* based on empirical observations.

Assuming that users will tend to access strings of documents from the same server, it is useful to characterize the locality of reference between different Web pages. We therefore define the *consecutive document retrievals* distribution as the number of consecutive pages that a user will retrieve from a single Web server before moving to a new one (either as a result of following hyperlinks in an existing document, or by selecting a completely unrelated document).[4]

Finally, the *server selection distribution* defines the relative popularity of each Web server, in terms of how likely it is that a particular server will be accessed for a set of consecutive document retrievals.

A.5 Experimental Results

From our traffic traces and subsequent analysis, we derived the various probability distributions for the different components of our model. We found these distributions to be consistent with the results of existing Web measurement studies. We have summarized the more interesting facets of these measurements in Table A-3.

A.5.1 Anomalies

In some cases, we noticed odd trends in our data, which indicated a large number of nearly-identical Web documents transferred periodically. For example, the 11 October 1995 trace

4. Implicit in this component of the model is the additional assumption that all the components of a Web document tend to come from the same server.

HTTP request sizes show a bimodal distribution.
HTTP reply sizes have a heavy-tailed distribution, and tend to be larger than request sizes.
A simple heuristic based on timing can be used to group individual files into documents.
The number of files per document tends to be small; 80% of documents required less than four file transfers.
The number of consecutive documents retrieved from a given server tends to be small. 80% of visits to a server's document space resulted in fewer than six documents being retrieved.

Table A-3. Selected Measurement Results.

showed a number of Web page retrievals with interarrival times of about five minutes. There were 291 such transfers, accounting for approximately 20% of those transferred during the whole trace. Upon further investigation, we determined that the documents came from a Web server that displayed real-time still images of the San Francisco, CA skyline. A Web page used an extension to HTML which caused clients to automatically reload documents every five minutes, thus updating the picture. As these (and other) periodic HTTP retrievals were skewing our data, we removed them from our traces prior to further analysis.[5]

A.5.2 Request Length

HTTP requests are sent from a client to a server. They typically specify a file to retrieve, although they may also provide information to a computation to be performed on the server. Also contained in each request are some identifying fields about the user, the client software, and the request itself.

The only user bytes sent from client to server are those contained as a part of the HTTP request. Thus, we measured the request sizes by simply counting the number of bytes in the appropriate direction of each TCP connection, summed over all packets. The statistics summarizing the requests in our four traces are shown in Table A-4.

The cumulative distribution functions (CDFs) for the request size distributions are shown in Figure A-1. The reply sizes in our traces all exhibited a bimodal distribution, with one large peak occurring around 250 bytes and another, smaller one around 1 KB. We believe

5. While it may be argued that these retrievals should contribute to our traffic model since they actually occurred in real life, the nature of this model is such that it cannot accurately capture the correlations between successive document retrievals from such a Web client. A model attempting to characterize such periodic Web traffic should explicitly account for this behavior.

	19 Sep 1995	11 Oct 1995	1 Nov 1995	20 Nov 1995
Number	5030	5699	3659	18034
Minimum Size	10	40	40	8
Maximum Size	1825	1786	1333	2404
Mean Size	356	327	325	301
Median Size	231	244	235	244

Table A-4. Summary of HTTP Request Lengths (in Bytes).

that the former requests correspond to simple file retrievals, while the latter may contain more complex requests such as those generated by HTML forms. However, there is insufficient information in our existing traces to prove or disprove this hypothesis. (Investigating further would require packet traces containing all or most of the payload bytes from each packet.)

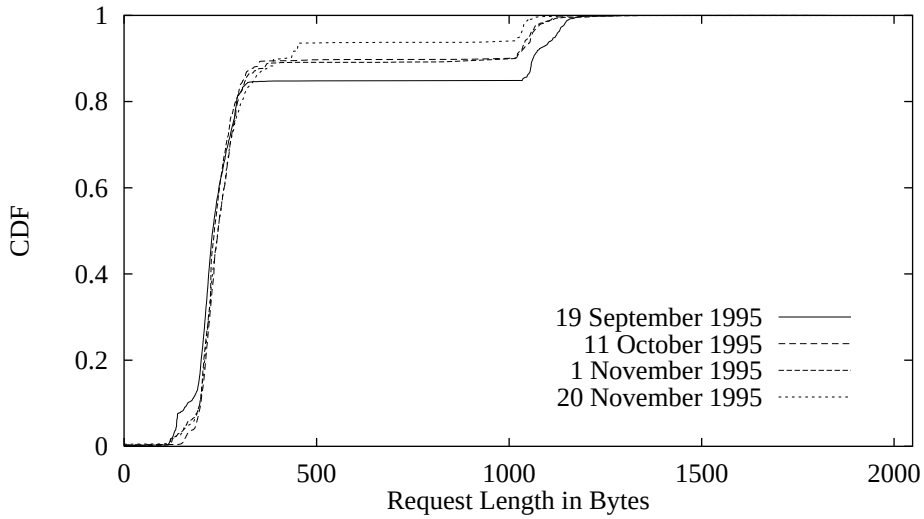


Figure A-1. Cumulative Distribution Functions of HTTP Request Lengths.

A.5.3 Reply Length

The HTTP reply consists of the bytes sent from the server to the client. Typically, the reply contains either HTML text or some multimedia data (e.g. an image or audio clip) to be displayed by the Web client. In the case of an error (e.g. a nonexistent file), the HTTP reply contains an error message. As with HTTP requests, some identifying information is also included.

In Table A-5, we present a summary of the HTTP replies recorded in our four traces. The CDFs for the reply size distributions are shown in Figure A-2. We note that two of the maximum file sizes are identical. Upon further investigation, we found that these replies were both generated by downloads of a single large data archive file from a Web server operated by a local research group.

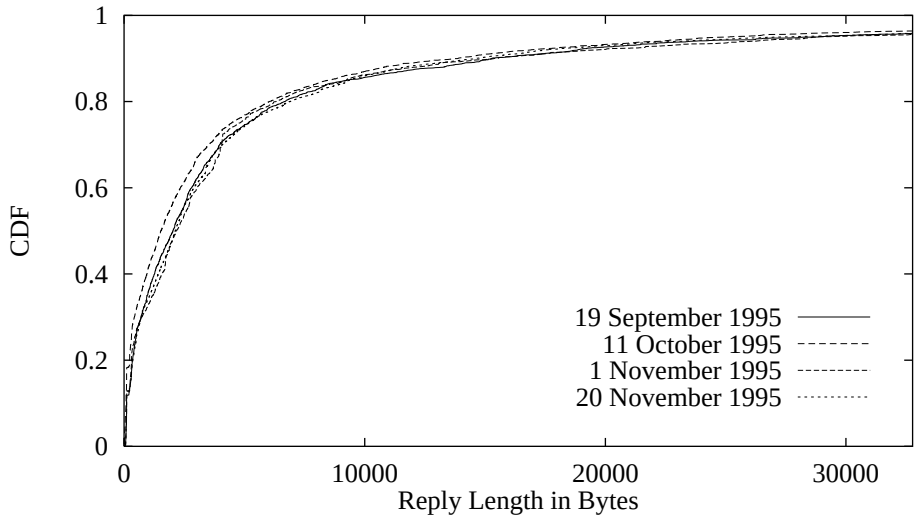


Figure A-2. Cumulative Distribution Functions of HTTP Reply Lengths.

	19 Sep 1995	**11 Oct 1995**	**1 Nov 1995**	**20 Nov 1995**
Number	5030	5699	3659	18,034
Minimum Size	62	81	57	30
Maximum Size	8,146,976	3,270,319	1,740,705	8,146,976
Mean Size	10664	8899	8319	8812
Median	2035	1532	2179	2127

Table A-5. Summary of HTTP Reply Lengths (in Bytes).

In each of the traces, the minimum reply length was very short (only tens of bytes). It is likely that these replies represent either errors or “not modified” responses to `If-Modified-Since` (conditional document retrieval) requests. While the actual length of some files may indeed be in the range of tens of bytes, the addition of HTTP headers makes the reply messages somewhat longer.

We note that the maximum reply sizes are rather large (over 1 MB in each of the traces). Furthermore, the means (8–10 KB) are much larger than the median reply sizes (about 2 KB). These characteristics are consistent with distributions that are "heavy-tailed" (with a large amount of the probability mass in the tail of the distribution). It has been in fact demonstrated that WWW file sizes are heavy-tailed [Crovella96].

On the assumption that HTTP retrievals generally result in the transfer of a WWW file (and in particular, the assumption that large HTTP replies contain WWW files), it seems natural to expect that HTTP replies would share this characteristic. We repeated the analysis of [Crovella96] on our data, and found that reply sizes above 1 KB are reasonably well-modeled by Pareto distributions with α estimates ranging from $\alpha = 1.04$ to $\alpha = 1.14$.[6] Further details are given in Table A-6. By comparison, [Crovella96] arrived at an estimate of $\alpha = 1.06$.

	19 Sep 1995	**11 Oct 1995**	**1 Nov 1995**	**20 Nov 1995**
α	1.05	1.04	1.09	1.14
R^2	0.98	0.99	0.97	0.98

Table A-6. Estimates of the α Parameter for the Tail of HTTP Reply Size Distributions. R^2 is the coefficient of determination, and takes values in the range $[0 \ldots 1]$. Values near 1 indicate a "good" fit of the regression, and that the simple linear regression used to estimate α can account for nearly all the variation.

A.5.4 Page Length

Determining the number of files per page is less straightforward, because we cannot determine exactly which TCP connections transferred parts of a single document. An HTTP client merely issues the requests for the files making up a given document, in succession.

We therefore used two simple heuristics to determine whether two HTTP connections belong to the same document. First, the two connections must originate from the same IP address, since retrievals from two different client machines cannot possibly belong to the same document. We note that it is possible for two connections from the same IP address

6. The Pareto distribution is a "heavy-tailed" probability distribution with a CDF given by $F(x) = P[X \leq x] = 1 - \left(\frac{k}{x}\right)^{\alpha}$, where k is the minimum value of X.

to be associated with two unrelated documents, which can happen in the case that two different users on the same host fetch a document at the same time. However, we evaluated this possibility as unlikely, because the end hosts were workstations each used almost exclusively by a single user.

Second, the two connections cannot be separated by "too much time", an interval determined by a parameter we call T_{thresh}. More formally, let c_1 and c_2 be two HTTP connections. Let $S(c)$ be the arrival time of the starting packet of connection c and let $E(c)$ be the arrival time of the ending packet of connection c. Assuming $S(c_1) < S(c_2)$, we judge c_1 and c_2 to belong to the same document only if $S(c_2) - E(c_1) \leq T_{\text{thresh}}$. If $S(c_1) < S(c_2) < E(c_1)$, the two connections overlap and we assume that their respective files belong to the same document. This latter condition can occur with browsers that use multiple, overlapping TCP connections to improve interactive performance, such as Netscape Navigator. Figure A-3 illustrates the role of T_{thresh} in determining the relation between two HTTP connections.

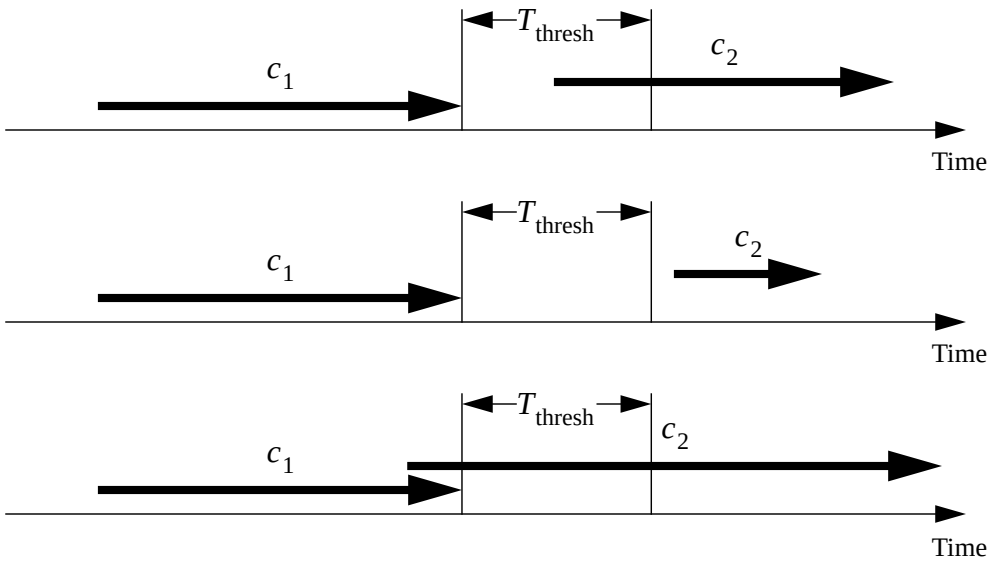


Figure A-3. Heuristic for Determining the Relation Between Two HTTP Connections. Timelines run left-to-right; TCP connections are represented by thick arrows. In the top timeline, c_2 starts within T_{thresh} time after the end of c_1; thus we judge c_1 and c_2 to belong to the same document. In the center timeline, the gap between c_1 and c_2 is greater han T_{thresh}; thus the two belong to different documents. In the bottom document, c_2 starts before c_1 finishes; in this case the two are judged to belong to the same document.

This heuristic requires the definition of a suitable value of $_{\text{thresh}}$. As $_{\text{thresh}}$ becomes very short, it may become smaller than the time necessary for an HTTP client to initiate a retrieval. In this case, connections which really belong to the same Web document will be falsely classified as belonging to different documents. Conversely, as T_{thresh} becomes large, it may become longer than the time for a user to react to the displayed document and select a new document to view. This can make files from different pages appear to be part of the same document.

The analysis in [Crovella96] required a similar classification in order to analyze the distribution of idle times between connections. This analysis classified files separated by less than one second of idle time as belonging to the same document, due to the limitations of the users' reaction time. Idle times greater than 30 seconds were deemed to separate independent documents, as few items would take longer to be processed and displayed. Idle times in the intermediate range were assumed to belong to a "transition" region. According to this reasoning, reasonable values for T_{thresh} can be found in the range $1 \text{ sec} < T_{\text{thresh}} < 30 \text{ sec}$.

We picked $T_{\text{thresh}} = 1 \text{ sec}$ for this study. The primary influence on our choice of this value is that users will generally take longer than one second to react to the display of a new page and order a new document retrieval. For HTTP clients that perform multiple overlapping file transfers, the time to process and display a file does not affect the choice of T_{thresh}, as the various components of a multipart document are downloaded, processed, and displayed in parallel.

Given our choice of an idle threshold, we characterized the number of files per document, as shown in Table A-7. We note that in the survey of HTML documents in [Bray96], slightly more than half of all pages contained either zero or one inlined image, corresponding to either one or two connections per document. Considering that some of our "documents" were actually single-file (thus, single-connection) downloads, which would tend to skew this distribution downward, we feel that our observations are consistent with this statistic.

	19 Sep 1995	11 Oct 1995	1 Nov 1995	20 Nov 1995
Mean	2.9	2.8	3.2	3.1
Median	1	1	1	1

Table A-7. Mean and Median Number of Files Per Document, $T_{\text{thresh}} = 1$ sec.

We note that although the distributions of the number of files per document varies as T_{thresh} changes, they are very similar for values around $T_{\text{thresh}} = 1$ sec. Thus, the exact choice of T_{thresh} is not critical to our analysis. Figure A-4 illustrates this fact graphically, for the set of HTTP connections recorded in the 19 September 1995 trace.

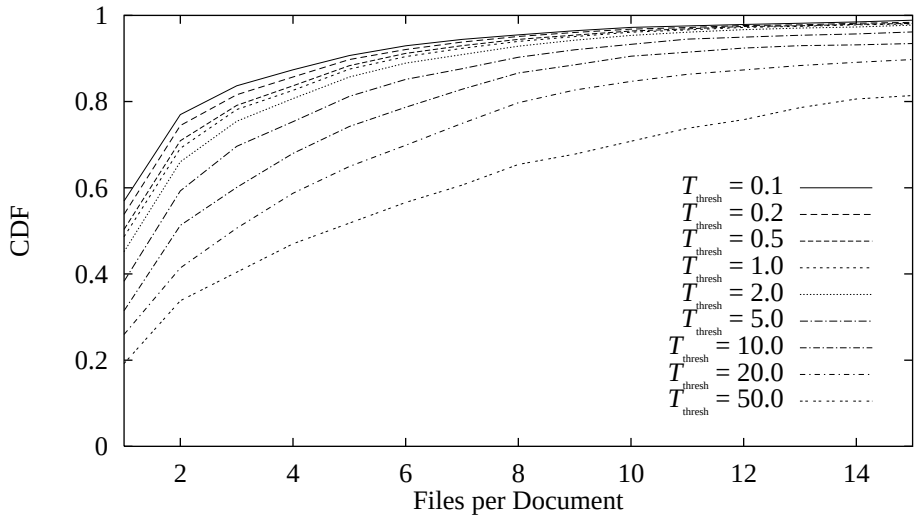


Figure A-4. Cumulative Distribution Functions of Document Length in Files, 19 September 1995. Curves correspond to varying values of T_{thresh} in seconds.

A.5.5 User Think Time

Given a selection of T_{thresh}, the empirical distribution of user think times between pages can be determined by the set of all interconnection idle times T, $T > T_{\text{thresh}}$. In Table A-8, we summarize the user think times extracted from the four Web traces. The 20 November 1995 trace had a much longer mean think time than the others. We believe this fact is due to the timing of this particular trace, which covered the American Thanksgiving holiday in late November. The University of California observes this holiday as a four-day weekend, which could conceivably account for some of the long idle times. The CDFs for the user think time distributions for all four traces is given by Figure A-5.

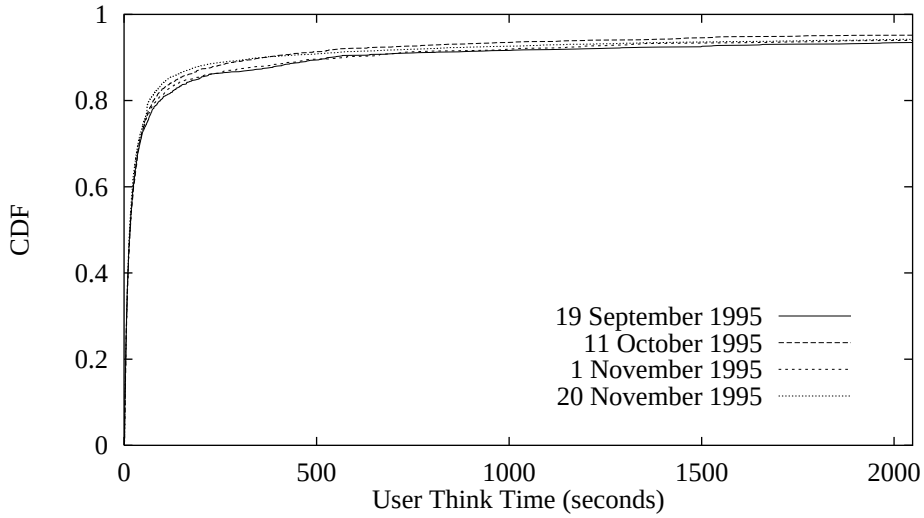


Figure A-5. Cumulative Distribution Functions of User Think Times.

	19 Sep 1995	**11 Oct 1995**	**1 Nov 1995**	**20 Nov 1995**
Number	1678	1995	1092	5692
Maximum Time	86395	80681	65914	271309
Mean Time	1313	854	837	1915.84
Median Time	15	16	16	14

Table A-8. User Think Times in Seconds.

A.5.6 Consecutive Document Retrievals

The current design of many Web document archives is such that users will frequently access documents from the same server in succession. As we saw in our discussion of various virtual circuit management policies in Chapter 6, this locality of TCP connections may be a significant influence on network performance. Table A-9 summarizes the number of consecutive document retrievals from HTTP servers during our network traces. By contrast, [Catledge95] noted that users accessed an average of ten consecutive pages per server, considerably more than the average of four to five document retrievals we observed. We believe that the difference is attributable to the interaction between user browsing strategies and client caching in Web browsers. Users tend to use a browsing strategy that has been described as “spoke and hub”, which involves frequent backtracking to already-visited pages. In browsers that implement client-side caching, revisited pages will not generate any network traffic (and thus would not appear in a network trace), but they

would be counted in a client-side event trace. Thus, we would expect our consecutive document retrieval count to be somewhat lower than the corresponding figure from a client trace by about half, as we observed.

	19 Sep 1995	**11 Oct 1995**	**1 Nov 1995**	**20 Nov 1995**
Number	253	306	171	873
Maximum Documents	37	54	37	112
Mean Documents	4.1	4.3	4.2	4.4
Median Documents	2	2	3	2

Table A-9. Consecutive Document Retrievals Per Server Access.

In Figure A-10, we show the CDF for the consecutive document retrievals distributions from our traces. As can be seen, users tend to switch between servers fairly frequently (the median number of consecutive documents retrieved is usually two). However, we noted cases in which visits to Web servers lasted for tens of consecutive documents.

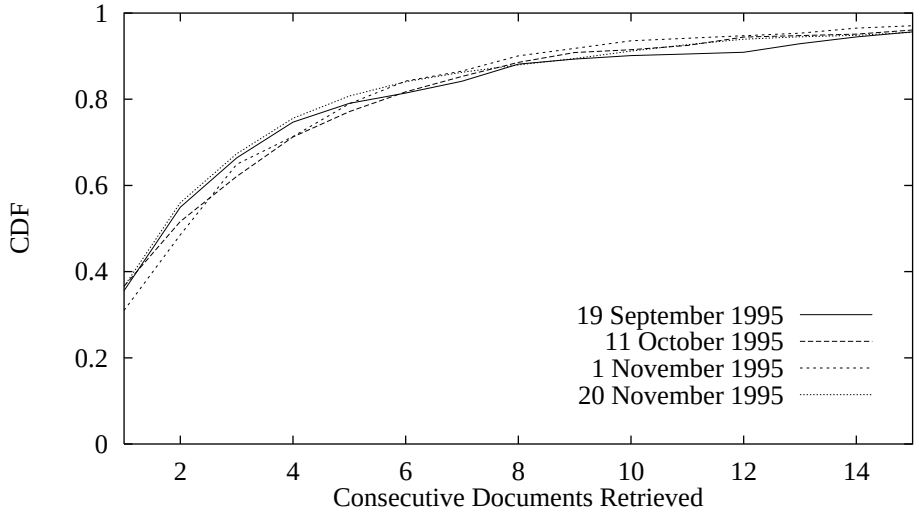


Table A-10. Cumulative Distribution Functions for Consecutive Document Retrievals.

A.5.7 Server Selection

The server selection distribution characterizes the relative popularity of Web servers. We computed the number of times that any given Web server was used for a set of one or more consecutive document retrievals. In Table A-11, we summarize the ten most popular serv-

ers for consecutive document retrievals during the 19 September 1995 trace, out of a total of 136 servers accessed as the start of 253 strings of consecutive document retrievals.

Rank	Frequency	Type
1	43	Local
2	11	Local
3	8	Remote
4	7	Remote
5	6	Local
6	6	Remote
7	6	Remote
8	6	Local
9	5	Remote
10	5	Remote

Table A-11. Top Ten Servers Observed, 19 September 1995. The frequency column shows the number of times any given server was accessed as the start of a stream of consecutive document retrievals. The type of a server reflects whether it is located locally on-site or not.

By this metric, the most-visited server in this trace (indeed, for all four traces) was the local departmental Web server. Among other items of interest, it contains homepages for the vast majority of the users of the machines attached to the network being traced, as well as the startup document for many users. We note that four of the top ten servers were located on-site.

Given these characteristics, particularly the fact that so many of the servers accessed were local to the tracing site, we believe that we have insufficient information to properly characterize this aspect of our model. We have chosen instead to approximate the server selection distribution using a Zipf's Law distribution. Zipf's Law is a discrete, heavy-tailed distribution that states that the probability of selecting the ith most popular item in a set is proportional to $1/i$. Originally, it was used to describe the frequency of words in texts, as well as other human-related phenomena [Zipf49]. More recently, this distribution has been applied to the access frequency of Web documents [Crovella96, Arlitt96]. It would seem reasonable to apply Zipf's Law, or some other heavy-tailed distribution, to the access patterns of servers as well, but confirmation of this assertion requires a larger data sample than we have available.

We note here several difficulties in attempting to measure visits to Web servers from IP-layer packet traces. The first problem is that IP-layer packet traces do not reveal the exact hostname originally used to access documents, but only the IP address of the server. Hostnames can only be obtained by performing queries to nameservers, which will return the canonical names of hosts, but not their aliases. For example, it would be very difficult to determine that the machine whose canonical name is `kohler.CS.Berkeley.EDU` is frequently accessed as `http.CS.Berkeley.EDU` or `www.CS.Berkeley.EDU`.

Another, related problem is that, in the case that a hostname maps to multiple IP addresses, it may be difficult to associate accesses to these various IP addresses with a single name. This particular situation may arise in the case of replicated HTTP servers, which rely on randomization in the Domain Name System to spread accesses to a single Web server across multiple machines, as described in [Katz94].

A.6 Model Representation

When choosing a representation for this traffic model, there were two basic approaches we considered. One was to attempt to fit the observed data to probability distributions that are easily described analytically. A simple analytic representation has the advantages of being compact and (perhaps) easier to use. This approach was discussed in [Paxson94a]. However, in circumstances where a data set cannot be described by a well-known distribution (such as the bimodal request size distributions discussed in Section A.5.2), this technique cannot easily be used.

The alternative was to represent probability distributions by their CDFs, and to use the inverse transformation method (for example, as described in [Jain91] and applied in [Danzig91]). While requiring more storage and perhaps being slower to generate random values, this approach does have the virtue of being able to represent arbitrary probability distributions.

For two reasons, we chose to maintain the CDF representations for most of our probability distributions. (However, the Zipf's Law substitute to the server selection distribution was calculated analytically.) The first reason was the ability to represent arbitrary distributions.

A more pragmatic reason was that the `tcplib` distributions from [Danzig91] already used this representation, and we had already implemented the mechanisms to generate random values using the inverse transform method. We based our distributions on the traffic gathered in the 19 September 1995 trace.

The INSANE network simulator, initially described in Chapter 3, uses this model to mimic both the activity of HTTP clients and that of HTTP servers. The behavior of a simple Web browser is illustrated via the pseudo-code in Figure A-6; an algorithm for simulating a single-threaded Web server is shown in Figure A-7. The simulation of more complex HTTP applications, such as Web browsers capable of multiple, concurrent retrievals, or multi-threaded Web servers, would be similar.

```
while (!done) {
    /* select server and number of documents to retrieve */
    /* from that server */
    server = ServerSelection();
    numdocuments = ConsecutiveDocumentRetrievals();

    /* retrieve documents in succession */
    while (numdocuments) {

        /* retrieval for document */
        numfiles = DocumentLength();
        while (numfiles) {
            requestLength = Request();
            send(requestLength);
            reply = receive();
            numfiles--;
        }

        /* wait for user to think */
        wait(UserThinkTime());
        numdocuments--;
    }
}
```

Figure A-6. Pseudo-Code for a Simple HTTP Client.

We reiterate that INSANE also models the TCP congestion and flow control mechanisms of TCP, and that any meaningful Internet simulation must account for their effects.

```
while (!done) {

    /* Get request from client */
    request = receive();

    /* Figure out length of reply */
    replyLength = Reply();

    send(replyLength);
}
```

Figure A-7. Pseudo-code for a Simple HTTP Server.

A.7 Conclusions

We have constructed an empirical model of network traffic produced by the HyperText Transfer Protocol used by World Wide Web applications. This model consists of a number of probability distributions determined by analysis of actual HTTP conversations. From packet traces, we have built up higher-layer of communication patterns, from individual HTTP retrievals to Web pages to groups of pages. This approach gives a sufficient level of detail to serve as a component of a workload generator for a packet-level simulation of an IP internetwork being used to carry Web traffic.

Our characterization of Web-generated network traffic has shown that HTTP requests exhibit a bimodal distribution, and that (as revealed in prior studies) the sizes of HTTP replies have a heavy-tailed distribution. We have shown that a simple heuristic can be used to separate HTTP transfers into different Web pages. We have characterized some aspects of user Web page selection in terms of locality of consecutive documents referenced. Where possible, we have compared the results of our measurements and analysis to other Web measurement studies and found them consistent with those prior results.

A.8 Future Work

There are, of course, areas where this model can be refined; we list several as topics for possible future work. We feel that the Zipf's Law substitute to the server selection distribution could be replaced with an empirical distribution, given an adequately-long trace of network data. It would also be desirable to investigate any correlations between the differ-

ent components of our model (for example, there may be a correlation between the popularity of a given server and the number of consecutive documents fetched from it).

The constantly-changing nature of the Web calls for updates to this model to track trends over time. The growth of Web traffic may affect the nature of documents and Web-browsing behavior. New protocol developments will force the use of new measurement and analysis methodologies.[7] Increasing use of new Web features such as Java will change the profile of files and documents being accessed.

Finally, the conversion of our empirical distributions to closed-form analytic expressions would aid in making the models adaptable to the data and workload found for different types of user communities and document archives.

7. In fact, performing new traffic measurements with current-day Web traffic requires more advanced analysis techniques than we present here, due to early support for persistent-connection HTTP on the part of some Web browsers and servers. For example, recent versions of the Netscape Navigator browser [Netscape96] and the Apache server [Apache96] support this feature.

B An Empirical Model of Internet Video

We present a model of Internet video traffic sent by the popular applications `vic` and `vgw`. The model is based on a traffic trace taken during late 1995 at the University of California at Berkeley. It captures the behavior of a `vic`/`vgw` source during the two different modes of operation (conditional replenishment and background updates) encountered during video transmission. We present both the model and the empirical distributions for the various quantities making up the model. Finally we show how it can be used to generate a stream of packets for a network simulation, such as INSANE.

B.1 Introduction

The popular video tool `vic` [McCanne95, McCanne96b] uses an encoding scheme, known as *Intra-H.261*, designed for the lossy network environment of the Internet and its Multicast Backbone (*MBONE*) virtual network [Macedonia94]. To provide resilience against network losses, this scheme encodes and transmits only intraframe-coded blocks, thus there are no temporal dependencies between block updates. Each block is encoded using the H.261 video coding standard.

Instead of performing inter-frame compression (as with video coding schemes such as MPEG), `vic` uses a scheme known as conditional replenishment to select blocks to be transmitted. Only blocks which change by more than a certain threshold amount (for example, due to motion in the video image) are transmitted. In the absence of motion, blocks not experiencing motion are updated less frequently, so that all receivers eventually receive a complete image.

vic performs open-loop rate control on a per-packet basis. After it sends each packet, the transmitter “sleeps” for a certain amount of time determined by the rate control algorithm and the rate settings specified by the user (bit rate and frame rate).

One problem with multicasting in the Internet environment is the heterogeneous nature of the various networks and hosts. The different capabilities (especially in terms of network bandwidth) can make it difficult to select a source bit rate that delivers an acceptable image quality to all parties. To address this problem (and others), an application-level video gateway vgw [Amir95b, Amir96] has been developed to convert between encoding formats and bit rates.[1] Thus, a multicast source can send a high bit rate video stream, which can then be transcoded and rate-limited by vgw to a lower-bit rate stream more suitable for low-bandwidth environments. vgw uses the same coding and rate control algorithms as vic.

vic and vgw are most visibly used in the Internet MBONE for the purpose of transmitting live video of interesting sessions to the Internet community. Past examples have included portions of Internet Engineering Task Force (IETF) meetings, portions of relevant conferences such as ACM SIGCOMM, and various seminar series. Combined with an Internet audio tool (vat) and a shared whiteboard (wb), these tools can be used to extend the audience of a presentation far beyond a local site.

In the network simulations required by our evaluation of IP-over-ATM policies, we wanted to include some instances of video applications, both to analyze their performance and their effects on other applications. To fill this need, we created a synthetic traffic source that mimics the operation of vic (or vgw).

Section B.2 describes some related and prior work. In Section B.3, we describe our methodology for capturing a sample of video traffic. Section B.4 describes our traffic model, and shows how we derived the model’s empirical probability from our network measure-

1. This software has since been renamed rtpgw, reflecting the fact that it can be used as an application-level gateway for different types of RTP sessions, not just video.

ments. We discuss the representation and usage of our model in Section B.5. Finally, in Section B.6, we make some concluding remarks.

B.2 Related Work

There are many examples of synthetic video workloads based on empirical measurements and observations of real traffic. Perhaps the best-known and most-often-used such model is the “Star Wars” workload of [Garrett93], which consists of a trace of the video frame sizes from a popular science-fiction motion picture, encoded using a VBR video coder performing intraframe compression.

Several well-known Internet-based models, based on traces of actual traffic, exist. For example, `tcplib` [Danzig91] can be used to mimic the traffic sent by a variety of common Internet applications, circa 1991. The model described in Appendix A provides a similar characterization for World Wide Web activity.

B.3 Methodology

We wanted to base our model on observed network traffic. To do this, we needed a trace of packets (with timing) sent during an actual MBONE multicast. We used the `tcpdump` [Jacobson95] utility to capture all the packet headers from `vgw` during a multicast from the Berkeley Multimedia Seminar Series on 1 November 1995. The session used `vgw` to transcode a 1 Mbps motion JPEG-encoded video stream into Intra-H.261 at a lower bit rate for transmission over the MBONE. The lower-rate MBONE multicast session we measured had a target bit rate of 128 Kbps and a target frame rate of 8 frames per second.

The session we traced lasted from 12:14 PM to 1:46 PM (local time), a total of one hour, 32 minutes. During this time the `vgw` process sent 69,135 packets, containing a total of 38.8 MB of data (this figure includes only UDP payload, not packet headers). The average bit rate, over the entire trace, was 56 Kbps.

B.4 Model

We model the stream of video packets using a two-state model, as shown in Figure B-1. Each state represents the modes of operation when `vgw` outputs conditional replenishment and background updates, respectively. We note that this two-state model is rather simplis-

tic—a slightly more accurate model would account for the fact that conditional replenishment and background updates can, in fact, take place concurrently. Such a model could be captured either by adding more states to the existing model or by using a superposition of two packet arrival processes (one for each type of update). However, our simplified model has the virtue of being easy to derive from our trace data.

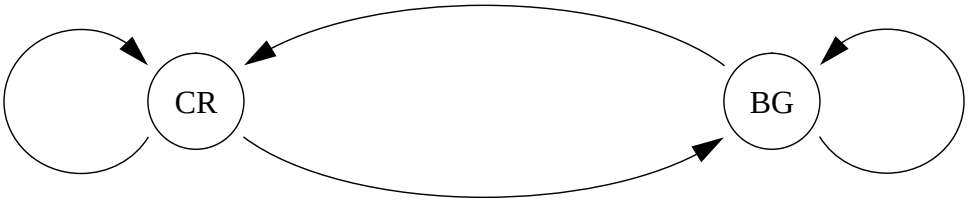


Figure B-1. Two-State Model of Video Source. The two states refer to two modes of operation; packets transmitted from each state have different characteristics.

The length of time the application transmits in each state is described in terms of the number of consecutive packets sent from each visit to that state. We name these two distributions $\text{packets}_{\text{CR}}$ and $\text{packets}_{\text{BG}}$, respectively.[2]

Within each state, the source sends UDP packets with sizes drawn from the size_{CR} and size_{BG} distributions, as appropriate. We note that selecting the packet sizes in each state is sufficient to completely describe the source's behavior while in that state, due to the rate control algorithm employed by `vic` and `vgw`. When doing conditional replenishment, a packet of size s bytes is followed by a gap of $\frac{8s}{\text{BR}}$ seconds, where s is in bytes and BR is the target bit rate in bits per second.

In the case of background updates, any packet (regardless of size) is followed by a gap of $\frac{1}{\text{FR}}$ seconds, where FR is the target frame rate in frames per second. This gap size ensures an update of at least one block per frame time. We note that each update block is much smaller than a complete video frame.

We computed the four probability distributions of our model, based on the packet sizes and arrival times captured during the MBONE multicast described in Section B.3.

2. We chose this representation, rather than using transition probabilities, because it allows us to describe two-state behavior other than a two-state Markov chain.

B.4.1 Packet Classification

For each of the packets we captured, we recorded the UDP payload size and the length of the gap separating that packet from the one following it. Figure B-2 shows a scatterplot of a representative subset of the packets in our trace.

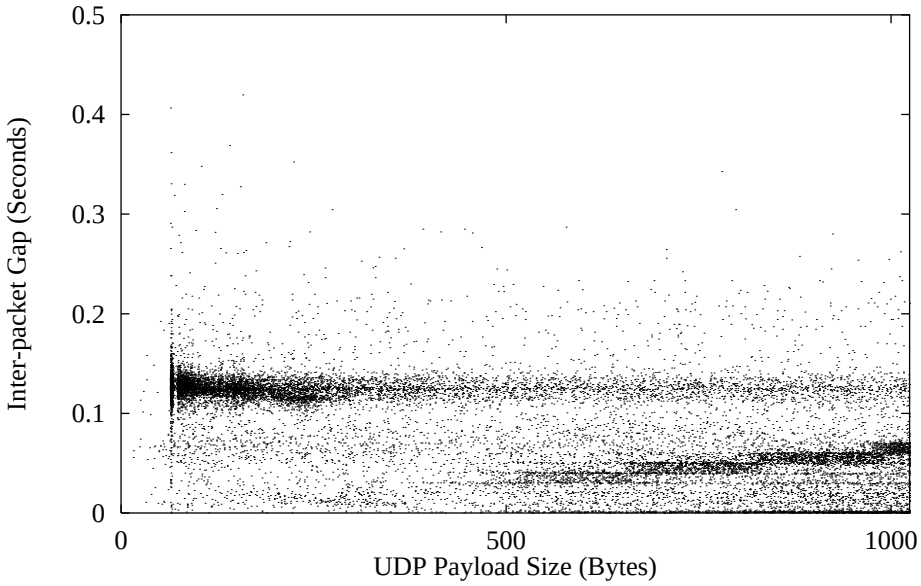


Figure B-2. Scatterplot of 20,000 Packets. This plot shows the packet payload size and following interpacket gap.

We observe that many of the packets appear to be loosely clustered into two groups. The first lies along a line through the origin, with a slope corresponding to approximately 61 microseconds per byte. These packets are conditional replenishment updates. The slope of the line corresponds to the target bit rate, 128 Kbps.

The other major grouping of packets lies around the horizontal line with a Y intercept at 0.125 seconds. These packets are background updates; 0.125 seconds is the reciprocal of the target frame rate (eight frames per second).

Packets lying far away from either line can be explained by a variety of phenomena, including queueing or media access delays, packet loss, or operating system scheduling granularity. For many of these reasons, we note that it is important to capture traffic close to the source. Our traces were captured on the same subnet as the workstation running the `vgw` transcoder. Transitions between the BG and CR states can also cause these outlying points, due to differences between our model and the `vic`/`vgw` implementation.

We used a graphical method to classify all the packets in the trace as being either conditional replenishment or background update packets. We classified each point (and associated packet) according to which of the two aforementioned clusters of packets was closer. This procedure gave us the ability to reconstruct, at least partially, the original application's behavior, even when the exact state of the source's encoder was unknown.[3]

B.4.2 Packet Sizes

Once we were able to classify packets into the two classes (CR and BG), we could examine the characteristics according to their types. The first natural measurement was the distribution of packet sizes.

Conditional replenishment packets were, in general, larger than background updates. CR packets were an average of 758 bytes long, and accounted for 27.9 MB of the bytes recorded (72% of the total). By comparison, the background updates, which made up 10.9 MB of the trace bytes, had an average length of 338 bytes. In Figure B-3 we show the cumulative distribution functions of the size_{CR} and size_{BG} packet size distributions. Clearly the two have very different distributions, and need to be modeled separately (we note that they depend very much on the threshold value used to detect changes in blocks).

B.4.3 State Times

By counting the number of consecutive packets in each of the two states, we computed the distributions of the state times (packets_{CR} and packets_{BG}). They give some measure of how long the video source remained in either the CR or BG states.

The conditional replenishment states lasted longer in general than the background updates states (on average, 5.03 packets, vs. 4.43). However, the longest background states were much longer (462 consecutive packets, as opposed to 37), due to extremely long times without any motion in the video image. Figure B-4 shows the cumulative distribution of the state times.

---

3. We actually collected several sessions' packet headers, when `vgw` was still under development and undergoing debugging. By viewing scatterplots similar to Figure B-2, the author of `vgw` was able to identify the effects of bugs in old versions of `vgw` [Amir95a].

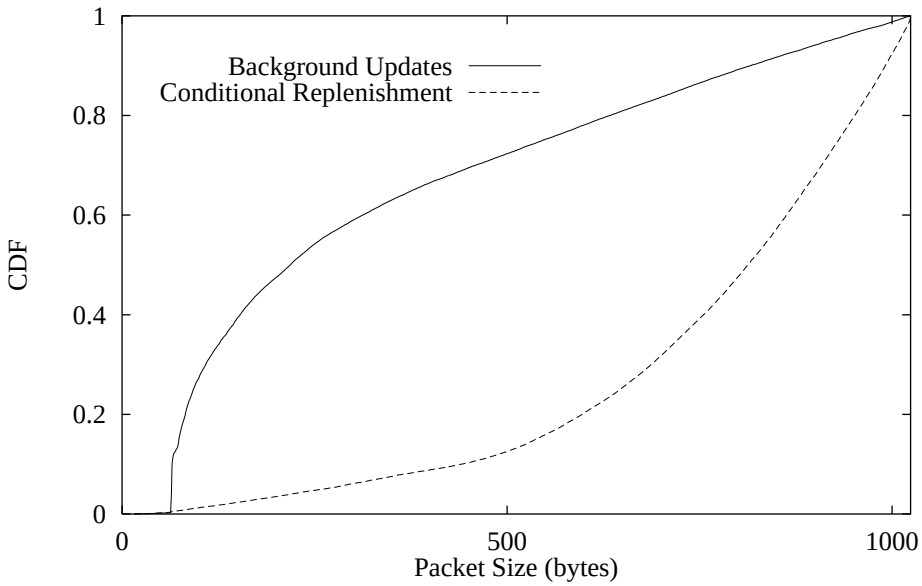


Figure B-3. Cumulative Distribution Functions of Packet Sizes.

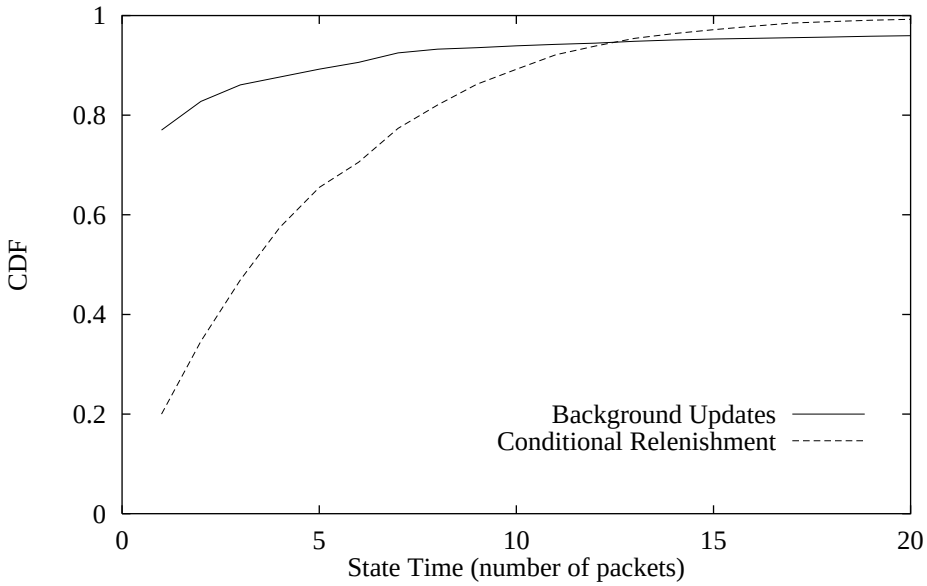


Figure B-4. Cumulative Distribution Functions of State Times.

B.5 Model Representation

As with our HTTP model, we chose to represent the various probability distributions by storing their CDFs and using the inverse transform method to compute samples from these distributions.

We implemented a simulated video source based on this model as a part of the INSANE network simulator. It transmits packets according to this model for a given target bit rate and frame rate.[4] The pseudo-code for this source is shown in Figure B-5.

```
/* initialize */
state = CR;
packets = PacketsCR();

while (!done) {

    /* compute next packet and inter-packet gap */
    if (state == CR) {
        size = SizeCR();
        delay = size * 8 / BR;
    }
    else {
        size = SizeBG();
        delay = 1 / FR;
    }

    /* switch states if necessary */
    if (--packets) {
        if (state == CR) {
            state = BG;
            packets = PacketsBG();
        }
        else {
            state = CR;
            packets = PacketsCR();
        }
    }

    /* send packet and wait for inter-packet gap */
    Send(size);
    Sleep(delay);
}
```

Figure B-5. Pseudo-code for a Simple Internet Video Source.

In our simulations, the sink of video data was a passive receiver; it only recorded statistics on received data (such as loss and delay). In reality, the receiver would also be sending control and membership information to the other members of the multicast group. A charac-

4. We believe this model to be reasonably accurate and useful, as long as the target bit rate and frame rate used by the synthetic traffic source are close to those of the original source. In all the simulations we did, we used the original bit rate of 128 Kbps and the original frame rate of 8 frames per second.

terization of this traffic, which is small in volume compared to the actual video data, is beyond the scope of this study.

Finally, we note that although we captured data from a multicast session, INSANE (as currently implemented) only supports unicast video transmission and data forwarding. As the original session had only one sender, we believe that the model is still applicable to our simulation scenarios.

B.6 Conclusions

We have derived an empirical model of Intra-H.261 coded video, as sent by the coder used by the MBONE video tools `vic` and `vgw`. This model is based on a traffic trace taken during a live MBONE multicast in late 1995. We believe that our model is simple enough to be easily used and that it captures enough interesting, meaningful properties of the traffic stream to be used in network simulations.

It would be useful to test this model against a variety of different types of video data, produced by this same application. One fairly simple study would be to examine different video contents and their effect on the output stream from the codec. For example, a “talking heads” seminar multicast would very likely generate very different traffic from an “action movie”. Even though the two could be encoded using the same coder, with the same parameters, different amounts of motion would affect the use of conditional replenishment, and hence produce different bit rates. It would also be helpful to gather data produced streams with different thresholds, frame rates, and bit rates, to provide both a validation of this model and a variety of workloads.

Another possible area of future work would to be to investigate possible short-term or long-term correlations in the video stream.

Bibliography

[Almquist92] Philip Almquist. *Type of Service in the Internet Protocol Suite*. Internet Request for Comments 1349, July 1992.

[Amir95a] Elan Amir. Personal communication, December 1995.

[Amir95b] Elan Amir, Steve McCanne, and Hui Zhang. "An application level video gateway." In *Proceedings of ACM Multimedia 95*, San Francisco, CA, November 1995.

[Amir96] Elan Amir and Steve McCanne. *rtpgw software*, 1996. This software is available at ftp://daedalus.cs.berkeley.edu/pub/rtpgw/.

[Apache96] The Apache Group. *Apache software*, 1996. This software is available at http://www.apache.org.

[Arlitt96] Martin F. Arlitt and Carey L. Williamson. "Web server workload characterization: The search for invariants." In *Proceedings of the ACM SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, pages 126–137, Philadelphia, PA, May 1996.

[ATM Forum95] ATM Forum. *ATM User-Network Interface Specification, Version 3.1*. PTR Prentice Hall, 1995.

[Banerjea96] Anindo Banerjea, Domenico Ferrari, Bruce A. Mah, Mark Moran, Dinesh C. Verma, and Hui Zhang. "The Tenet Real-Time Protocol Suite: Design, implementation, and experiences." *IEEE/ACM Transactions on Networking*, 4(1):1–10, February 1996.

[Berners-Lee95] Tim Berners-Lee and Daniel W. Connolly. *Hypertext Markup Language – 2.0*. Internet Request for Comments 1886, November 1995.

[Berners-Lee96] Tim Berners-Lee, Roy T. Fielding, and Henrik Frystyk Nielsen. *Hypertext Transfer Protocol – HTTP/1.0*. Internet Request for Comments 1945, May 1996.

[Biagioni93] Eduardo Biagioni, Eric Cooper, and Robert Sansom. "Designing a practical ATM LAN." *IEEE Network*, pages 32–39, March 1993.

[Bohn94] Roger Bohn, Hans-Werner Braun, Kimberly C. Claffy, and Stephen Wolff. "Mitigating the coming Internet crunch: Multiple service levels via precedence." *Journal on High Speed Networks*, 1994.

[Bray96] Tim Bray. "Measuring the Web." In *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, May 1996.

[Cáceres91] Ramón Cáceres, Peter B. Danzig, Sugih Jamin, and Danny Mitzel. "Characteristics of wide-area TCP/IP conversations." In *Proceedings of ACM SIGCOMM '91*, Zurich, Switzerland, September 1991.

[Cáceres92] Ramón Cáceres. *Multiplexing Traffic at the Entrance to Wide-Area Networks*. PhD thesis, Computer Science Division, University of California at Berkeley, December 1992.

[Cáceres93] Ramón Cáceres. "Multiplexing data traffic over wide-area cell networks." Unpublished technical report, Matsushita Information Technical Laboratory, Princeton, NJ, January 1993.

[Catledge95] Lara D. Catledge and James E. Pitkow. "Characterizing browsing strategies in the World-Wide Web." In *Proceedings of the Third International World Wide Web Conference*, Darmstadt, Germany, April 1995.

[Claffy94] Kimberly C. Claffy. *Internet Traffic Characterization*. PhD thesis, University of California, San Diego, 1994.

[Clark92] David D. Clark, Scott Shenker, and Lixia Zhang. "Supporting real-time applications in an integrated services packet network: Architecture and mechanism." In *Proceedings of ACM SIGCOMM '92*, pages 14–26, Baltimore, MD, August 1992.

[Cole96] Robert G. Cole, David H. Shur, and Curtis Villamizar. *IP over ATM: A Framework Document*. Internet Request for Comments 1932, April 1996.

[Crovella96] Mark E. Crovella and Azer Bestavros. “Self-similarity in World Wide Web traffic: Evidence and possible causes.” In *Proceedings of the ACM SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, pages 160–169, Philadelphia, PA, May 1996.

[Cunha95] Carlos R. Cunha, Azer Bestavros, and Mark E. Crovella. “Characteristics of WWW client-based traces.” Technical Report BU-CS-95-010, Computer Science Department, Boston University, July 1995.

[Danzig91] Peter B. Danzig and Sugih Jamin. “tcplib: A library of TCP internetwork traffic characteristics.” Technical Report USC-CS-91-495, Computer Science Department, University of Southern California, Los Angeles, CA, 1991.

[Deering96] Steven E. Deering and Robert M. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. Internet Request for Comments 1883, January 1996.

[Ferrari89] Domenico Ferrari. “Real-time communication in packet switching wide area networks.” Technical Report TR-89-022, International Computer Science Institute, Berkeley, CA, May 1989.

[Ferrari90] Domenico Ferrari. “Client requirements for real-time communication services.” *IEEE Communications Magazine*, 28(11):65–72, November 1990.

[Ferrari94] Domenico Ferrari, Anindo Banerjea, and Hui Zhang. “Network support for multimedia – a discussion of the Tenet approach.” *Computer Networks and ISDN Systems*, 26:1267–1280, 1994.

[Fielding96] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. Internet Draft draft-ietf-http-v1.1-spec-07, August 1996.

[Fraser92] A. G. Fraser, C. Kalmanek, A. Kaplan, W. Marshall, and R. Restrick. “Xunet 2: A nationwide testbed in high-speed networking.” In *Proceedings of IEEE INFOCOM ’92*, Firenze, Italy, May 1992.

[Garrett93] Mark W. Garrett. *Contributions Toward Real-Time Services on Packet-Switched Networks*. PhD thesis, Columbia University, New York, NY, May 1993.

[Gibbons85] Jean Dickinson Gibbons. *Nonparametric Methods for Quantitative Analysis*. American Series in Mathematical and Management Sciences. American Sciences Press, Inc., Columbus, OH, second edition, 1985.

[Gupta95a] Amit Gupta. *Multi-party real-time communication in computer networks*. PhD dissertation, University of California at Berkeley, December 1995.

[Gupta95b] Amit Gupta, Wingwai Howe, Mark Moran, and Quyen Nguyen. "Resource sharing for multi-party real-time communication." In *Proceedings of INFOCOM '95*, Boston, MA, April 1995.

[Handley96] Mark Handley. "Re: Port ranges assigned to video and audio." Posting to rem-conf mailing list, message 19388.837175676@cs.ucl.ac.uk, July 1996.

[Heinanen93] Juha Heinanen. *Multiprotocol Encapsulation over ATM Adaptation Layer 5*. Internet Request for Comments 1483, July 1993.

[ION96] "Internetworking over NBMA working group charter," May 1996. This document is available at ftp://ftp.nexen.com/pub/ion/ion-charter.txt.

[Ipsilon96] Ipsilon Networks. *IP Switching: The Intelligence of Routing, The Performance of Switching*, February 1996.

[Jacobson88] Van Jacobson. "Congestion avoidance and control." In *Proceedings of ACM SIGCOMM '88*, Stanford, CA, August 1988.

[Jacobson95] Van Jacobson, Craig Leres, and Steven McCanne. *tcpdump software, Version 3.0.2*, 1995. This software is available at ftp://ftp.ee.lbl.gov/tcpdump.tar.Z.

[Jacobson96] Van Jacobson and Steve McCanne. *vat software*, 1996. This software is available at ftp://ftp.ee.lbl.gov/conferencing/vat/.

[Jain91] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., New York, NY, 1991.

[Johnston95] William E. Johnston. "BAGNet: A high speed, metropolitan area, IP over ATM network testbed." In *Proceedings of IEEE Compcon 1995*, San Francisco, CA, March 1995.

[Kalmanek90] C. R. Kalmanek, H. Kanakia, and S. Keshav. “Rate controlled servers for very high-speed networks.” In *Proceedings of Globecomm ’90*, San Diego, CA, December 1990.

[Kantor86] Brian Kantor and Phil Lapsley. *Network News Transfer Protocol*. Internet Request for Comments 977, February 1986.

[Kantor91] Brian Kantor. *BSD Rlogin*. Internet Request for Comments 1258, December 1991.

[Katz94] Eric Dean Katz, Michelle Butler, and Robert McGrath. “A scalable HTTP server: The NCSA prototype.” In *Proceedings of the First International WWW Conference*, Geneva, Switzerland, May 1994.

[Keshav94] S. Keshav. “Experiences with large videoconferences on XUNET II.” In *Proceedings of INET ’94*, Prague, Czech Republic, June 1994.

[LANE95] ATM Forum, Foster City, CA. *LAN Emulation Over ATM, Version 1.0*, January 1995.

[Laubach94] Mark Laubach. *Classical IP and ARP over ATM*. Internet Request for Comments 1577, January 1994.

[Luciani96] James V. Luciani, Dave Katz, David Piscitello, and Bruce Cole. *NBMA Next Hop Resolution Protocol (NHRP)*. Internet Draft draft-ietf-rolc-nhrp-10, October 1996.

[Lund95] Carsten Lund, Steven Phillips, and Nick Reingold. “Adaptive holding policies for IP over ATM networks.” In *Proceedings of IEEE INFOCOM ’95*, pages 80–87, Boston, MA, April 1995.

[Macedonia94] Michael R. Macedonia and Donald P. Brutzman. “Mbone provides audio and video across the Internet.” *IEEE Computer*, pages 30–36, April 1994.

[Mah93] Bruce A. Mah. “A mechanism for the administration of real-time channels.” Masters report, University of California at Berkeley, April 1993.

[Mah94a] Bruce A. Mah. “Enhancements to the XUNET IP service.” unpublished technical memorandum, AT&T Bell Laboratories, Murray Hill, NJ, July 1994.

[Mah94b] Bruce A. Mah. “Measurements and observations of IP multicast traffic.” Technical Report CSD-94-858, University of California at Berkeley, December 1994.

[Mah97] Bruce A. Mah. “An empirical model of HTTP network traffic.” In *Proceedings of IEEE INFOCOM ’97*, Kobe, Japan, April 1997. To appear.

[Maher95] Maryann Perez Maher, Fong-Ching Liaw, Allisamn Mankin, Eric Hoffman, Dan Grossman, and Andrew G. Malis. *ATM Signalling Support for IP over ATM*. Internet Request for Comments 1755, February 1995.

[McCanne95] Steven McCanne and Van Jacobson. “vic: A flexible framework for packet video.” In *Proceedings of ACM Multimedia ’95*, pages 522–522, San Francisco, CA, November 1995.

[McCanne96a] Steve McCanne and Sally Floyd. *ns software*, 1996. This software is available at http://www-nrg.ee.lbl.gov/ns/.

[McCanne96b] Steve McCanne and Van Jacobson. *vic software*, 1996. This software is available at ftp://ftp.ee.lbl.gov/conferencing/vic/.

[McKusick96] Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.4 BSD Operating System*. Addison-Wesley Publishing Company, Reading, MA, 1996.

[Mogul95] Jeffrey C. Mogul. “The case for persistent-connection HTTP.” In *Proceedings of ACM SIGCOMM ’95*, pages 299–313, Cambridge, MA, August 1995.

[Mosaic95] National Center for Supercomputing Applications. *NCSA Mosaic software*, July 1995. This software is available at http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/.

[Netscape96] Netscape Communications Corporation. *Netscape Navigator software*, 1996. This software is available at http://home.netscape.com/.

[Newman96a] Peter Newman, W. L. Edwards, Robert M. Hinden, Eric Hoffman, Font Ching Liaw, Tom Lyon, and Greg Minshall. *Ipsilon Flow Management Protocol Specification for IPv4 Version 1.0*. Internet Request for Comments 1953, May 1996.

[Newman96b] Peter Newman, W. L. Edwards, Robert M. Hinden, Eric Hoffman, Font Ching Liaw, Tom Lyon, and Greg Minshall. *Transmission of Flow Labelled IPv4 on ATM Data Links*. Internet Request for Comments 1954, May 1996.

[Newman96c] Peter Newman, Tom Lyon, and Greg Minshall. "Flow labelled IP: A connectionless approach to ATM." In *Proceedings of IEEE INFOCOM '96*, pages 1251–1260, San Francisco, CA, March 1996.

[Ousterhout94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Publishing Company, Reading, MA, 1994.

[Padmanabhan94] Venkata N. Padmanabhan and Jeffrey C. Mogul. "Improving HTTP latency." In *Proceedings of the Second International World Wide Web Conference*, Chicago, IL, October 1994.

[Parulkar95] Guru Parulkar, Douglas C. Schmidt, and Jonathan Turner. "IP/ ATM: A strategy for integrating IP with ATM." In *Proceedings of ACM SIGCOMM '95*, pages 49–59, Cambridge, MA, August 1995.

[Paxson91] Vern Paxson. "Measurements of wide area TCP conversations." Masters report, University of California at Berkeley, May 1991.

[Paxson94a] Vern Paxson. "Emprically derived analytic models of wide-area TCP connections." *IEEE/ACM Transactions on Networking*, 2(4):316–336, August 1994.

[Paxson94b] Vern Paxson. "Growth trends in wide-area TCP connections." *IEEE Network*, 8(4):8–17, July 1994.

[Postel80] Jon Postel. *User Datagram Protocol*. Internet Request for Comments 768, August 1980.

[Postel81a] Jon Postel. *Internet Protocol*. Internet Request for Comments 791, September 1981.

[Postel81b] Jon Postel. *Transmission Control Protocol*. Internet Request for Comments 793, September 1981.

[Postel82] Jonathan B. Postel. *Simple Mail Transfer Protocol*. Internet Request for Comments 821, August 1982.

[Postel83] Jon Postel and Joyce Reynolds. *Telnet Protocol Specification*. Internet Request for Comments 854, May 1983.

[Postel85] Jon Postel and Joyce Reynolds. *File Transfer Protocol (FTP)*. Internet Request for Comments 959, October 1985.

[Ptolemy96] Department of Electrical Engineering and Computer Sciences, University of California at Berkeley. *Ptolemy software*, April 1996. This software is available at http:// ptolemy.eecs.berkeley.edu/.

[Rekhter96] Yakov Rekhter, Bruce Davie, Dave Katz, Eric Rosen, and George Swallow. *Tag Switching Architecture Overview*. Internet Draft draft-rfced-info-rekhter-00, September 1996.

[Reynolds94] Joyce K. Reynolds and Jon Postel. *Assigned Numbers*. Internet Request for Comments 1700, October 1994.

[Romanow94] Allyn Romanow and Sally Floyd. "Dynamics of TCP traffic over ATM networks." In *Proceedings of ACM SIGCOMM '94*, pages 79–88, London, August 1994.

[Sandberg85] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. "Design and implementation of the Sun Network Filesystem." In *Proceedings of the USENIX Summer Conference Proceedings*, pages 119–130, Portland, OR, June 1985.

[Schmidt93] Andrew Schmidt and Roy Campbell. "Internet protocol traffic analysis with applications for ATM switch design." *ACM SIGCOMM Computer Communication Review*, 23(2):39–52, April 1993.

[Schulzrinne96] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. Internet Request for Comments 1889, January 1996.

[Shenker96] Scott Shenker, Craig Partridge, and Roch Guerin. *Specification of Guaranteed Quality of Service*. Internet Draft draft-ietf-intserv-guaranteed-svc-06, August 1996.

[Stevens94] W. Richard Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley Publishing Company, Reading, MA, 1994.

[Stevens96] W. Richard Stevens. *TCP/IP Illustrated, Volume 3*. Addison-Wesley Publishing Company, Reading, MA, 1996.

[Stroustrup91] Bjarne Stroustrup. *The C++ Programming Langauge*. Addison-Wesley Publishing Company, 1991.

[Sun Microsystems88] Sun Microsystems. *RPC: Remote Procedure Call, Protocol Specification, Version 2*. Internet Request for Comments 1057, June 1988.

[Woodruff96] Allison Woodruff, Paul M. Aoki, Eric Brewer, Paul Gauthier, and Lawrence A. Rowe. "An investigation of documents from the World Wide Web." In *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, May 1996.

[Wright95] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated, Volume 2*. Addison-Wesley Publishing Company, 1995.

[Wroclawski96] John Wroclawski. *Specification of the Controlled-Load Network Element Service*. Internet Draft draft-ietf-intserv-ctrl-load-svc-03, August 1996.

[Zhang93a] Hui Zhang and Domenico Ferrari. "Rate-controlled static-priority queueing." In *Proceedings of IEEE INFOCOM '93*, San Francisco, CA, 1993.

[Zhang93b] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. "RSVP: A new resource ReSerVation Protocol." *IEEE Network*, September 1993.

[Zipf49] George Kingsley Zipf. *Human Behavior and the Principle of Least Effort*. Hafner Publishing Company, New York, NY, 1949.