UNIFYING THE TRANSPORT LAYER OF A PACKET-SWITCHED INTERNETWORK

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Song Sam Liang
March 2003

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

David Cheriton
(Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Nick McKeown

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Peter Danzig

Approved for the University Committee on Graduate Studies.

Abstract

In the layering architecture of a packet-switched internetwork, the "transport layer" provides the end-to-end data transport service between applications across the network. On the Internet, the most popular computer internetwork, TCP is the most extensively used transport-level protocol. Years of enhancement and fine tuning has made it very efficient and robust. TCP is also congestion adaptive, which contributes significantly to the tremendous growth of the Internet. Therefore, TCP demonstrates that one transport protocol can cover a large percentage of applications.

The Internet is now increasingly being used for real-time multimedia applications, such as voice communications and video streaming. In addition, for multi-participant applications such as audio or video broadcasting, multicast is needed to make efficient utilization of the network resources. Furthermore, many applications are record-based, such as database applications and remote storage applications. TCP, as it is designed, does not well support these applications. Consequently, UDP has been used for these applications. However, UDP provides basically no transport level services at all. So built on top of UDP (or IP directly), there have been a proliferation of application-specific and candidate real-time transport protocols, and a large number of reliable multicast protocols aimed to solve the problem of efficient reliable data delivery to an arbitrarily large number of receivers. However, none of these protocols has achieved the desired effectiveness and efficiency, and neither has achieved sufficient maturity and acceptance.

We have designed three extensions to TCP: real-time extension TCP-RTM, framing extension TCP-Framing, and multicast extension TCP-SMO. We have implemented these extensions in a Linux kernel. Using this implementation, we have conducted both testbed-based experiments and public Internet experiments. The resulting performance measurement data have been quantitatively analyzed, and they show that these extensions are both effective and efficient for the targeted applications.

In summary, we show that a small number of low-cost extensions to a conventional connection-oriented reliable transport protocol such as TCP allow it to support real-time, multicast and record-based applications, and we can unify the transport layer of a packet-switched internetwork by using a single transport protocol.

Acknowledgements

I would like to thank...

First and foremost, my advisor David Cheriton for his sponsoring, mentoring and guidance over the past three years. He taught me how to do research, how to identify the real problems, how to be precise and scientific and focus on results that really matter. He has always been supportive and encouraging. Although sometimes his critique was rather harsh and made me feel really embarrassed, it did stimulate my passion and creativity, and taught me a lot more. I am so grateful to be able to work with David — his pursuit of excellence is truly inspirational!

Professor Nick McKeown and Professor Peter Danzig for serving on my dissertation committee. They offered many insightful comments that significantly improved the dissertation. Nick always responded to my requests quickly. Peter read the dissertation very closely and taught me a lot about academic writing. Peter also encouraged me to bring my research work to IETF to get it standardized and generate bigger impact.

Professor Bernd Girod for serving as the Chair of my oral defense.

All the DSGers, Katerina Argyraki, Evan Greenberg, Daniel Faria, Mark Gritter, Vince Laviano, Dan Li, Bo Yang and Dapeng Zhu, for their stimulating discussions and great suggestions.

Many other friends and colleagues in Stanford, Kevin Lai, Yi Liang, Athina Markopoulou, Costa Sapuntzakis and Xinhua Zhao for their gracious assistance.

My father Liang Gencheng and mother Xi Guizhen for raising me through all kinds of hardships and always encouraging me to pursue excellence and to finish the dissertation.

My wife Lei for all the sacrifice she made to support me through this long and difficult journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the layering architecture of a packet-switched internetwork, the *Transport Layer* provides the end-to-end data transport service between applications across the network. Common transport services include connection management, loss recovery, packet sequencing, duplicate elimination, data integrity, congestion control and flow control.

In the Internet, the most popular computer internetwork, the Transmission Control Protocol (TCP) is the most well-developed, extensively used and widely available transport protocol. Years of extensive network research has resulted in numerous improvements to the protocol and its implementation, including slow-start, congestion back-off and fast retransmit. As a result of this work and extensive tuning of the implementation, TCP is fast, efficient and responsive to network congestion conditions. Its use is further encouraged by the many firewalls that exclude all non-TCP traffic. Many measurements indicate TCP now accounts for 85 to 95 percent of the wide-area Internet traffic [123]. With the current development of TCP hardware accelerators in network interface cards, TCP can be expected to be favored for more and more applications.

However, the conventional TCP is not originally designed to support certain classes of applications, such as real time applications, record-based applications and multicast applications. In this dissertation, we show that a small number of simple extensions to a conventional connection-oriented reliable transport protocol, such as TCP, allow it to support all significant applications, including real time applications, record-based applications and multicast applications. Such unification leverages all the benefits of a conventional reliable transport protocol, such as the error recovery capability, congestion control mechanisms, and efficient implementation; eliminates the need to invent specialized protocols for

each class of applications and avoids the tremendous deployment obstacles faced by these new protocols.

1.1 TCP Supports Most Applications

A series of studies have been performed by both academic and industrial researchers to measure the Internet traffic patterns. They have obtained data from MCI backbone in 1997 [123] and 1998 [28], from NASA Ames Internet Exchange in 1999 [85], and from Sprint backbone in 2001 [46]. These studies indicate that TCP accounts for 85 to 95 percent of the Internet traffic and supports in a satisfactory fashion most significant applications, such as web browsing, file transfer, email and the NNTP news network, etc. In addition, these data collected over a period of five years show that there is no significant change of such TCP dominance.

For wireless traffic, David Kotz's study [65] has traced 2000 mobile users over 476 access points on Dartmouth College campus for 11 weeks, and his data shows that 97.5% of the wireless traffic runs on TCP.

Because these studies are performed by various independent researchers and are relatively comprehensive, their results are credible and convincing. Therefore, we conclude that most significant applications are well supported by TCP.

1.2 Some Applications Not Well Supported by TCP

TCP is originally designed to support point-to-point, reliable, byte-stream applications. Certain applications have special characteristics that are not particularly well supported by conventional TCP. Next we briefly list these issues, with more detailed illustration offered in subsequent chapters.

1.2.1 Real Time Applications

Real time multimedia applications, such as voice communication and audio/video streaming are gaining popularity due to the advancement of digital media technologies and the cost-effectiveness of the Internet compared to aging, analog distribution systems. These applications can tolerate certain data loss in the network, but are relatively delay sensitive, especially for interactive voice applications. Conventional TCP provides a fully reliable

service by persistently retransmitting a lost packet, potentially forever, until the lost packet is delivered successfully to the receiver. This behavior may disrupt the timing of the media being transmitted and severely degrade the end user experience.

1.2.2 Record-based Applications

Many applications have data represented in records. For example, database applications have data stored in objects or records. Multimedia applications have data represented in media samples. Most file systems, such as NFS, store and transfer data in disk blocks. If the record boundaries are preserved in the transmission process, the receivers can easily recover the records and efficiently process them.

Conventional TCP is designed as a byte-stream protocol, preserving no record boundaries. This deficiency requires the data receivers to perform extra operations to recover the original record boundaries before they can be processed. Such operations complicate the receiving applications and degrade their performance.

1.2.3 Multicast Applications

Multipoint data delivery, either reliable or semi-reliable for real time data, needs multicast to reduce the load on the server and reduce the bandwidth usage on the downstream links. Such applications include web cache distribution [74], software distribution [49], audio/video tele-conferencing, distributed games and Internet TV.

Conventional TCP is designed for point-to-point unicast communication. Due to its connection-orientedness, the destination address cannot be a multicast address, because the meaning of a connection between the source server and a group represented by the multicast address is not well defined. In addition, the research community holds the belief that TCP cannot handle a number of challenges when used with multicast, including ACK processing, RTT estimation, packet retransmission, and scalability.

1.3 The Problem

To support these applications, User Datagram Protocol (UDP) has been used and a lot of new specialized protocols have been invented. For example, for real time applications, there have been created a lot of special real time transport protocols, such as SCP, SCTP, DCP, HPF, RTCP, etc. For multicast applications, there have been proposed a long list of special

multicast transport protocols, such as SRM, RMTP, XTP, MTP, PGM, RMP, etc. More details for these protocols are given in Chapter 3.

Because of UDP's inherent simplicity, providing basically no transport level services, these new protocols have to reinvent a lot of functionalities that are already available in TCP, such as error recovery, packet sequencing, duplicate elimination, flow control and congestion control. Most of these protocols are rather complex and hard to use. None of them has achieved the maturity and acceptance of TCP and none of them is compelling to deploy. In this dissertation, we show that a few modest extensions to TCP eliminates the need for this proliferation of specialized protocols.

1.4 Original Contributions

The thesis of this dissertation is that a small number of low-cost extensions allow a conventional connection-oriented reliable transport protocol to support all significant applications. In this dissertation, because of the pre-dominance of TCP, we use TCP as the example for general-purpose, connection-oriented transport protocols. All the work we have done on TCP can be applied to other general connection-oriented transport protocols as well.

In support of our thesis, this dissertation makes the following specific original contributions:

- With TCP-RTM, we show that a modest extension to TCP allows it to support real time applications.
- With TCP-Framing, we show that a modest extension to TCP allows it to support record-based applications.
- With TCP-SMO, we show that a modest extension to TCP allows it to support multicast applications.

Today's Internet comprises hundreds of millions of hosts and routers, and TCP is widely deployed on almost all the hosts. To facilitate incremental deployment of our new technologies and obviate the need for a "flag day" for the huge number of devices on the Internet to transition to a brand-new protocol simultaneously, we focus on extending the existing technology to meet new requirements, rather than attempting to reinventing the wheels. Our results show that such minimalist approach indeed provide great benefits.

1.5 Outline of the Dissertation

The rest of this dissertation are organized as follows.

Chapter 3 surveys related work in the area of real-time transport protocols, framing support and multicast transport protocols, and discusses their limitations.

Chapter 4 describes the design and implementation of TCP-RTM, and presents the performance evaluation results based on a test-bed emulation and public Internet experiments. It also describes the congestion control behavior of TCP-RTM and presents the SNACK (Selective-NACK) extension to TCP-RTM with its performance evaluation results.

Chapter 5 describes the framing problem, presents the solution to add TCP-framing, and presents the performance evaluation results.

Chapter 6 describes the design of TCP-SMO to tackle the challenges to handle connection management, ACK processing, RTT estimation, packet retransmission, scalability, etc. It addition, this chapter presents the performance results of TCP-SMO regarding ACK-processing, reliable data distribution, real-time multicast applications and scalability.

Chapter 7 presents our conclusions and suggestions for future work.

Chapter 2

Background

This chapter introduces some background information about transport protocols, in particular the basic functionalities they provide, using TCP as an example. The extensions we describe later are all based upon these basic functionalities. In addition, this chapter briefly describes the unique characteristics of real-time applications and some special techniques they use, providing some background information about why some of the extensions are useful and effective. Finally, this chapter discusses the deployability of new network technologies.

2.1 An Overview of Transport Protocols

Transport protocols provide end-to-end transport services between applications across the network. The most common services include reliability, flow and congestion control, and connection management. In this section, we present a brief introduction of these transport services and describe briefly how these services are typically supported using TCP as an example.

TCP is a general purpose transport protocol in the DARPA Internet protocol suite. As mentioned in Chapter 1, it is the most extensively used transport protocol in the entire Internet, with 85% to 95% of the traffic running on top of it.

Although TCP is a transport protocol in the Internet protocol suite, it is an independent general purpose protocol that makes few assumptions of the underlying network technology. What it depends on is just a best-effort packet delivery service, without reliability or ordering requirements. It can work on the Internet Protocol (IP) network or other best-effort

networks such as those based on ISO's CNLP or Novell's Internet Packet eXchange protocol (IPX) or AppleTalk. The services provided by TCP and the way they are implemented can be applied to other general purpose transport protocol on a packet-switched internetwork. Throughout this dissertation, we just use TCP as an example to demonstrate how a general purpose transport protocol can be extended to support real-time applications, record-based applications and multicast applications.

2.1.1 Reliability

Reliability refers to the function to transmit all data from sender to receiver in order, without duplication and free of error. In TCP, reliability is achieved by using sequence numbers, positive acknowledgement (ACK), checksum and retransmission. Sequence numbers are used to detect missing packets and duplicates, and to reorder out-of-sequence packets. ACKs are used by the receiver to report to the sender the data that has been correctly received. The sender uses duplicate ACKs and a timer to detect packet loss and retransmit missing packets. Because the ACK as initially designed does not report packets received out-of-order, to improve the utilization of the network bandwidth by avoiding the retransmission of already received data, the Selective ACK (SACK) option has been added as an optimization to report dis-contiguously received packets.

Real-time multimedia applications are delay sensitive but can tolerate certain packet loss [1]. So these applications do not require full reliability. However, this dissertation shows that real-time multimedia applications can improve their playback quality by exploiting the reliability function provided by a reliable transport protocol.

2.1.2 Flow Control and Congestion Control

Flow control adjusts the sender's transmission rate to avoid overwhelming the processing speed of the receiver. TCP uses a sliding window mechanism to implement flow control to ensure that the sender does not send data faster than the receiver can process. TCP also uses this window mechanism to transmit multiple packets at the same time and try to saturate the network to get high throughput.

Congestion control detects and responds to network congestion. This is an important function to ensure fair sharing of the network bandwidth and to avoid congestion collapse.

[1] More details on this issue are presented in sub-section 2.2.

To implement congestion control, TCP uses a congestion window to control the amount of outstanding data that has been transmitted but not acknowledged. When congestion happens in the network, packet loss occurs and the sender is notified by either a timeout event or three duplicate ACKs from the receiver. In this case, the sender would shrinks its congestion window to reduce its transmission rate, so that the congestion condition on the network can be relieved.

TCP's congestion control function has contributed significantly to the success of the Internet. Unfortunately, more and more applications are using UDP to transmit real-time data. Because UDP does not perform congestion control, congestion-insensitive applications pose a big threat to the health and stability of the Internet. To avoid congestion collapse on the Internet, all traffic should perform congestion control. This dissertation shows that real-time applications can leverage the congestion control functionality of TCP to become congestion adaptive and thus become good citizens on the Internet.

2.1.3 Connection Management

Connection management refers to the opening, maintaining and terminating of connections between two communicating entities. At both the sender and the receiver, certain state information is maintained and dynamically updated for each connection, such as the estimated round-trip time, flow control window size, next sequence numbers to send and receive, etc. Such connection state is necessary to support reliability, flow and congestion control.

Using TCP, before the data communication starts, the two end-hosts perform a "three-way handshake" to set up a virtual connection. This three-way handshake makes sure that the connection is authorized and both sides are ready to communicate. It also exchanges some parameters, such as the maximum segment size, which is used in the subsequent data transmission; in addition, some options are negotiated to make sure that both sides agree on certain optional features. After the connection is set up, data can be exchanged between the two endpoints. Finally, when the data transmission is finished, the connection is tore down by exchanging some control messages.

Currently, most real-time applications, for example IP-telephony and video streaming, despite their connection-oriented nature, are using UDP, which is a connectionless transport protocol. To maintain connection state information, such as round-trip times and sequence numbers, they are forced to implement their own connection management features. This

dissertation shows that they can take advantage of the connection management support in a conventional connection-oriented transport protocol.

In addition, multicast applications are traditionally connectionless. However, this dissertation shows that by using a multicast extension of a conventional connection-oriented transport protocol, multicast applications can keep track of the state of each receivers and offer many useful functionalities, such as efficient error recovery, multicast flow control, receiver talk back and session relay.

2.2 Real-Time Applications

Real-time multimedia network applications, such as Internet telephony, online distance lecturing, Internet TV, have different characteristics from traditional bulk-data transfer applications, such as remote file transfer or World-Wide-Web applications. Due to the perception limitation of human beings, real-time applications may tolerate low level of data loss. For example, for an Internet telephony application, the occasional loss of a packet that is worth 20 milliseconds of voice may not be noticeable by the listener. Also, such occasional loss can usually be concealed by various application-level techniques, such as reusing the data in previous packet or interpolating the data in the previous packet and the next packet.

On the other hand, real-time multimedia applications are in general delay sensitive due to the intrinsic continuous characteristics of the audio or video media. The data in each packet has limited useful lifetime. If the data has not been received by the time it is needed for playback, it is in general not useful any more.

Therefore, real-time multimedia data is sometimes informally termed redundant but "perishable". A conventional reliable transport protocol is not suitable for such applications because it favors reliability over timely delivery of data, and when packet loss happens, it always attempts to retransmit the lost packet persistently until it is finally successfully retransmitted. This may conflict with the real-time requirements and degrades the media playback quality.

There has been a misconception that because real-time data is redundant, error recovery is not needed. Although the raw multimedia data contains a lot of redundancy, including both spatial redundancy and temporal redundancy, to save network bandwidth and storage space, the data usually goes through a compression process before being transmitted or

stored. The compression process substantially reduces the redundancy rate of the data, and makes them very vulnerable to packet loss during transmission. For example, using MPEG2, a popular video compression standard, video data can be compressed by 100 or more times. Some studies [86, 93, 114] have shown that even a network loss rate of 1% may render the quality of the streaming video intolerable. Therefore, compressed multimedia data becomes very sensitive to packet loss, and error recovery is very important to achieve good playback quality.

2.2.1 Playback Buffer and Playback Delay

Given the time-sensitivity of real-time data, there is another misconception that it is not possible to retransmit the lost packets.

The receiver of a real-time multimedia application usually uses a playback buffer to temporarily store the received data, and delays the playback of the data for certain amount of time. The playback buffer is used to smooth out the jitter in the network, so that the playback of the data in each packet can be continuous and isolated from the network's latency variation. The playback delay can range from 150ms to tens of seconds, depending on the classes of the application, and generally provides enough time to retransmit most lost packets. Chapter 4 shows that even for interactive applications, most lost packets can be recovered.

2.2.2 Classification

There are in general two major classes of real-time multimedia applications. The first class is *interactive applications*, including Internet telephony, online tele-conferencing, etc. Due to the interactive nature, this class of applications have higher latency requirements. To certain extent, the shorter the playback delay, the better the interactivity. However, there is a trade-off between interactivity and playback delay. When the network is lossy, too short a playback delay leaves no time for error recovery, which also hurts the quality of the interactive communication.

The second class is *uni-directional streaming applications*, of which *media-on-demand* is a subset. Internet TV, for example, belongs to this class. This class of applications do not require live interaction between the sender and the receiver, therefore, they are less delay sensitive, and the receiver can use much longer playback delay (of several seconds to tens

of seconds) and much larger playback buffer to absorb network jitter. Such longer playback delay provides plenty of time to retransmit lost packets and enable better playback quality.

2.3 Deployability

As a large system, the Internet evolves gradually and incrementally. Such evolutional advancement is further mandated by the Internet's tremendous scale, with hundreds of millions of network elements (hosts and routers) [55], its vast heterogeneity, and its high degree of distribution all over the world. Therefore, any new network technology, in addition to being able to solve some significant problems or enabling some compelling applications, needs to be based on the existing infrastructure and compatible and inter-operable with existing systems, to have any chance to be eventually adopted. In other words, it needs to have a viable path to go from here to there, or from the present to the future, in order to be deployed in the Internet. One factor of being "viable" is that this technology can be incrementally deployed, and that benefits can be obtained when it is initially deployed on only a small number of nodes. Any mechanism that requires all or most of the nodes be upgraded before benefits accrue is hard to deploy. One example of such a mechanism is RSVP [133], which requires to be deployed on all the routers between the two communicating hosts in order to be useful.

At the transport level, TCP has the biggest install base. It is implemented in almost all operating systems on a wide spectrum of platforms, ranging from hand-held computers, personal computers to large mainframe computers. As shown in Chapter 1, TCP accounts for 85 to 95 percent of the Internet traffic. Presumably, such a majority of the Internet applications are well supported by TCP, that is why they choose TCP. So TCP is the winner of the *natural selection* in the Internet world. In the remaining 5 to 15 percent of the traffic, a large fraction is DNS and routing traffic [85], which can and should use TCP as well to be reliable and congestion adaptive.

In addition, due to the maturity and vast popularity of TCP, it is being built into network interface card hardware. Furthermore, many routers capable of layer 4 switching have built into hardware the capability to parse TCP headers to support wire speed load balancing, access control, QoS or accounting [118, 120].

Therefore, if TCP can be slightly extended without changing the header format to meet additional application requirements, such as real-time or multicast requirements, there will

be much stronger motivation to deploy such extensions because the deployment will be simple and easy by just modifying slightly the current implementation in either software or hardware, compared to deploying a new protocol.

To deploy a new network protocol, such as SCTP or some other multimedia transport protocol or one of the reliable multicast protocols, the deployment process includes the precise definition of the protocol, standardization, and implementation in all kinds of operating systems, which take a long time and are very costly; in the mean time, there is no significant technical or economic benefits that make such huge investment worthwhile. Therefore, most new protocols we survey in this Chapter 3 are not easily deployable. In contrast, an extended TCP has a viable and much simpler deployment path.

2.3.1 How Many Transport Protocols Are Needed?

Most applications today require reliability, flow and congestion control, and connection management, and these requirements are well supported by one transport protocol – TCP. For every new popular application requirement such as real-time data delivery or one-to-many data distribution, do we need to create a brand new transport protocol from scratch? Most previous work surveyed in Chapter 3 take exactly this approach. They try to design a new protocol for every changed or new requirement, and this creates a proliferation of various protocols. This dissertation shows that this is the wrong approach. We demonstrate that there are simple ways to extend an existing conventional reliable transport protocol to meet all the new requirements. Therefore, one transport protocol can support all the significant applications, and that is all we need at the transport level.

Chapter 3

Related Work

This chapter introduces the readers to some related work in transport protocols for real-time applications and multicast applications, as well as related work in providing framing support in the transport layer. In Section 3.1, we consider other transport level efforts to support real-time applications. In Section 3.2, we examine other approaches to provide framing support in the transport layer. In Section 3.3, we survey the large number of multicast transport protocols in both research field and industry. In Section 3.4 we discuss other efforts to extend TCP to meet new requirements. Finally, we conclude with a summary in Section 3.5.

3.1 Real-Time Multimedia Applications

As explained in section 2.2, real-time multimedia applications have different characteristics from traditional bulk-data transfer applications. In this section, we survey some transport protocols that are designed specifically for real-time applications, and argue that they are not deployable. Next we look at efforts to extend existing transport protocols to support real-time applications. Then we review techniques to improve the performance of real-time applications, such as Forward Error Correction. Finally we examine the issue of performing congestion control for real-time applications and application level rate adaptation.

3.1.1 Congestion Control for Real-Time Applications

Congestion control is important to avoid congestion collapse and maintain the stability of the Internet. All applications should perform congestion control, whether they are traditional

bulk data transfer applications or real-time multimedia applications.

The conventional approach of using UDP for real-time communication has led to network problems in which UDP-based applications not sensitive to network congestion problems have caused problems for other applications, particularly the TCP-based applications. Here, TCP backs off in response to excessive UDP traffic, sometimes signaled through an active queue management mechanism such as RED, allowing UDP traffic to deny service to TCP-based applications.

To address this problem, many newly proposed protocols attempt to duplicate TCP's congestion control mechanisms such as SCTP, SCP, HPF, etc. [1] We see no necessity in creating new protocols from scratch and duplicating the same congestion control mechanisms in TCP. Through TCP-RTM, we advocate using TCP's congestion control mechanisms directly.

Another approach to add congestion control for real-time applications is proposed by Balakrishnan, et al. [11]. Called the Congestion Management Architecture, it acts as a shared module between the transport layer and the network layer on each endhost to maintain congestion control state and perform congestion control functions for all application flows running on that host. However, the advantage of such a new shared module in the operating system over traditional TCP congestion control is not clear, and because it is adding a new layer between the transport and the network layer, its deployment is going to be difficult.

Application-Level Rate Adaptation

Many researchers have worked on adjusting the data rate of multimedia applications to adapt to network congestion conditions. For example, Sisalem and Schulzrinne [117] present the loss-delay based adjustment algorithm (LDA) for adapting the transmission rate of multimedia applications to the congestion level of the network. The LDA algorithm was designed to reduce losses and improve utilization in a TCP-friendly way that avoids starving competing TCP connections. It relies on the end-to-end Real-Time Transport Protocol (RTP) for feedback information. For another example, Jacobs and Eleftheriadis [58] describe Dynamic Rate Shaping (DRS), which is a signal processing technique that adapts the rate of compressed video to dynamically varying bandwidth constraints. It provides an interface between the sender and the network, with which the encoder's output

[1] More details of these protocols are presented in subsection 3.1.2

can be matched to the network's available bandwidth. It uses a TCP like congestion control mechanism to adjust its congestion window to estimate the available bandwidth. We think these application level rate adaptation techniques can be used on top of TCP-RTM, instead of relying on new protocols that emulate TCP to provide feedback information, whose effectiveness has not been proven.

3.1.2 Special Real-Time Protocols

Following section 2.2 and section 3.1.1, a transport protocol needs to provide the following services to support real-time multimedia applications:

1. Timely data delivery.
2. Efficient error recovery.
3. Congestion control.

To address these issues, a number of new protocols, such as RTCP, HPF [75], SCP [20], SCTP [107], etc., have been proposed. Next, we briefly survey some representative protocols designed for real-time multimedia applications.

RTP and RTCP

Real-time Transport Protocol (RTP) [112] is an application level protocol that attempts to provide support for the transport of real-time data such as audio and video streams. RTP header includes timestamp, sequence number and payload type identifier to provide services such as time reconstruction, loss detection and content identification (including the payload format as well as the encoding/compression schemes). Default payload types, such as PCM, MPEG2 audio and video, H.261 video, etc., are defined in RFC 1890 [111]. More payload types can also be defined by providing a payload profile specification according to RFC 1890 [111]. RTP can be used for both unicast and multicast.

RTP does not provide mechanisms to perform error recovery and congestion control. These have to be implemented by applications running on top of RTP, which is a difficult problem for the applications.

Currently RTP is usually used on top of UDP, but not TCP. In this dissertation, we show that RTP can run perfectly on top of TCP to support real-time applications and take advantage of TCP's error recovery and congestion control capabilities.

When used on top of TCP, which provides much stronger error detection and error recovery, RTP's sequence number is redundant. However, when used on TCP-RTM, the real-time extension to TCP, RTP's sequence number is useful when TCP-RTM skips over a packet.

Real-Time Control Protocol (RTCP) is designed to work together with RTP to provide periodic feedback on the quality of data transmission. An RTCP packet carries control information such as *receiver report* and *sender report*. The *receiver report* contains reception quality feedback about data delivery, such as the latest sequence number received, the number of lost packets, and the timestamps for the sender to calculate round-trip time. The *sender report* contains, in addition to the information in a receiver report, total number of packets and bytes sent and some cross-media synchronization information. Note the similarity between the information in a receiver report and a sender report and a TCP packet. Each TCP packet contains sequence number, ACK sequence number. The TCP SACK option and the time-stamp option also provide additional information to report lost packets and time information for RTT estimation. So TCP can provide most of the functionalities of RTCP, so RTCP is not needed.

Heterogenous Packet Flow (HPF)

Similar to TCP, Heterogenous Packet Flow (HPF) [40, 75], designed by Dwyer, Ha, Li and Bharghavan, is a connection-oriented protocol. It allows applications to specify blocks of data called "frames", and to provide frame-specific policies for reliability, priority and timing. If a frame is specified as reliable, it will be retransmitted until successfully delivered. Otherwise, the frame is transmitted on a best-effort basis. If all packets are specified as un-reliable, HPF performs similarly to UDP. In addition, it allows the application to specify the deadline of each frame. If a frame cannot be delivered by its deadline, it is discarded. Although they present some data to show that their protocol may have some advantage over TCP when the network drop rate is high, such as 20%, it is not clear when the network drop rate is low, such as lower than 2%, whether all the extra mechanisms and complexity are still compelling. When the network loss rate is too high, for example higher than 10%, the quality of the real-time media being transmitted will be low no matter what the hosts at the two ends do. The real problem in this case is with the network itself, and the network itself needs to be upgraded or better managed to reduce the network drop rate and in turn improve the transmission quality.

HPF also attempts to use TCP-like congestion control mechanisms to provide congestion control to real-time applications, which seems to be a duplicated effort.

Therefore, it is not compelling to deploy HPF.

Streaming Control Protocol (SCP)

Streaming Control Protocol (SCP), designed by Cen, Pu and Walpole [20], runs on top of UDP. It focuses on making streaming applications live in harmony with each other and with TCP-based traffic. It uses congestion control mechanisms similar to TCP, such as slow start at the beginning of a connection and after a long pause, and exponential back-off of the congestion window in the case of network congestion. Unlike TCP, SCP does not retransmit data that are dropped in the network, in order to avoid transmitting possibly late packets and avoid potential long latency. getting rid of packet retransmission completely is not always a good choice. With TCP-RTM, we show that by taking advantage of the playback buffer at the receiver, many lost packets can be retransmitted in time, and they can considerably improve the quality of the media being transmitted without introducing unwanted latency. In addition, we show that TCP's congestion control mechanisms can be leveraged directly to make real-time applications congestion-adaptive. So there is no need to create a new protocol to do exactly the same thing.

Stream Control Transport Protocol (SCTP)

Stream Control Transport Protocol (SCTP) is a recently proposed protocol designed to support Public Switched Telephone Network (PSTN) signalling messages over IP networks, and it is claimed to be suitable for other applications, including real-time applications. Basically, this new protocol mimics TCP in terms of its use of sequence number, packet retransmission, control flow and congestion flow. In addition, it provides the notion of association and streams and TCP-like congestion control.

An SCTP *association* is similar to but broader than TCP's notion of *connection*. An association represents the connection session between the sender and the receiver, but it can contain multiple *streams*. Packets that belong to different streams of the same association can be delivered to the user independently. Also, an SCTP endpoint is identified by a port number and a list of IP addresses to support multihoming.

The new functionalities SCTP added are of limited use because they can be easily supported by TCP. For example, the multiple streams in an association session can be

easily supported by multiple TCP connections. And considering the ever decreasing cost of memory, the memory overhead of two or three TCP connections is of little concern. The multihoming support can be added to TCP by extending TCP in a similar way to TCP-SMO, by transferring data transmitted to each IP address on a host to a specific TCP connection.

In summary, given the rich feature set of TCP and the wide deployment of TCP, there is no compelling reason to deploy SCTP.

We will talk about SCTP a little more in section 3.2.3 regarding framing support.

Summary

Although these protocols have studied and explored various protocol techniques and helped us better understand the real-time data delivery problem, they are not designed with the consideration to fit into the existing standard network infrastructure. They have not achieved the efficiency, maturity and robustness of TCP, and their deployment will be difficult.

3.1.3 Extending Existing Protocols

For incremental deployment in the Internet, we focus on extending existing protocols, in this case TCP, to meet expanded application requirements. Krasic, et al. [66] and Mukherjee, et al. [89] concur with us on this general approach.

QoS-adaptive Video System over TCP

Krasic, Walpole, Li and Goel [66] present the position that a substitute transport protocol for TCP to support multimedia streaming will provide marginal benefits. In particular, for video streaming applications, they argue that the delay involved in TCP's packet retransmission and the rate variation resulting from TCP's congestion control mechanism do not affect the video playback quality because the large client side playback buffer can hide the retransmission latency and smooth out the rate variation. To support this position, they describe an architecture for content preparation and delivery intended to demonstrate effective streaming of stored content over TCP, and they present preliminary results from implementing a QoS-adaptive video system based on the proposed architecture. Such QoS-adaptive video system does application level rate adaptation on the sender side based on

the feedback from the receiver. Basically, when sustained congestion happens, the sender application will choose to send only lower quality data and reduce the data rate. However, they have not addressed the issue of timeliness vs. reliability for interactive applications.

Time-lined TCP

To address trade-off between reliability and timely delivery, Mukherjee and Brecht's Time-lined TCP (TLTCP) [89] associates a deadline with each packet at the sender side, and if a packet is not acknowledged by the deadline, it is discarded. In addition, because a packet may be discarded by the sender before it is transmitted successfully to the receiver, there may be discontinuity in sequence number at the receiver. Therefore, TLTCP adds a new TCP option in every packet to inform the receiver what the next sequence number it should expect to receive. In the case that a packet is expired and discarded by the sender, this TCP option informs the receiver the sequence number of the subsequent packet. The receiver is modified to handle this new TCP option, and updates its next expected sequence number based on the value in the new TCP option. They perform ns-based simulations and find that TLTCP performs time-lined data delivery and shares the network bandwidth fairly with competing TCP flows under most circumstances.

Because of the new socket option to specify the lifetime of each packet and the new TCP option for each TCP packet, this approach seems to be more complicated, compared to the TCP-RTM approach we are going to describe. In addition, there is a potential problem after an expired packet is discarded and the packet that carries the next expected sequence number notification is lost in the network, in which case the receiver does not know that a packet has been discarded, and keeps waiting for it, which will never arrive. Furthermore, due to the network latency between the sender and the receiver, when the sender discards an expired packet, it does not know exactly whether that packet has been received by the receiver or it has been dropped by the network. In the latter case, there is congestion in the network, and the sender should respond to that and scales back its congestion window. This problem is not addressed by TLTCP.

Another issue is that TLTCP does not address how the sender determines the lifetime of each packet. We think the receiver, rather than the sender, has a better idea as to what the lifetime of a packet should be. For example, in the case of video streaming, if the receiver's playback buffer is large, the lifetime of each packet can be much longer than the case that the playback buffer size is small.

In summary, TLTCP's approach to extend an existing transport protocol is in the right direction, however, simpler extension is feasible, as demonstrated by TCP-RTM presented later.

3.1.4 Forward Error Correction

An alternative approach to retransmission for error recovery is Forward-Error Correction (FEC). FEC [18, 102] transmits some redundant data, called parities, along with the original data. At the receiver, the redundant data can allow the reconstruction of the original data if certain amount of data is lost or corrupted in transmission. The redundant data is derived from the original data using various techniques based on the coding theories [78], such as exclusive OR (XOR) or Reed-Solomon codes.

FEC has been used for error recovery for both unicast and multicast communications, for example the Tornado codes [16]. However, there are two major problems with the FEC approach. First, it requires higher network bandwidth due to the added redundancy, which aggravates the network congestion condition and may cause higher loss rate. Second, FEC requires batch processing of large number of packets as part of the encoding so is not suitable for live real-time streams in general due to the extra delay introduced.

On the other hand, where appropriate, for example, for one-way streaming, when the application is less delay sensitive, FEC can be used on top of TCP-RTM, which reduces the application-level loss rate and allows FEC to use fewer extra bytes for error correction to reduce data redundancy and achieve higher encoding/decoding efficiency.

3.2 Framing Support

Many applications are record-based, such as database applications, multimedia applications and network file systems. Back in 1987, David Cheriton [23] defined the data access model in the Uniform I/O Interface (UIO) in terms of the sequence of data blocks associated with a UIO object. The unit of data access is a block. To quote [23]: for record-based applications, "a byte-stream abstraction is difficult to implement and obscures record boundaries that may have important semantics to the client." Therefore, framing support is important in the transport layer for record-based applications to improve the performance and simplify the implementation.

3.2.1 Fragmentation Considered Harmful and PMTU Discovery

Talking about framing, it is necessary to know how to determine the size of the data frames.

The Internet is composed of many different types of networks. Each type of network may have different Maximum Transmission Unit (MTU). In general, when the maximum possible packet size is used, the optimal throughput can be achieved. However, if a router receives a packet that is larger than the MTU on the interface that the packet needs to be forwarded to, the router needs to perform fragmentation. Kent and Mogul [64] show that such fragmentation is harmful, because it can result in poor performance (such as unnecessary retransmission of all the packets for one datagram even when only one fragment is lost) or even complete communication failure in the case of some systematic packet loss.

To avoid such fragmentation, it is recommended that endhost performs Path MTU (PMTU) [88] discovery before transmitting data. The PMTU is equal to the minimum MTU of all the network links along the path between the sender and the receiver. The PMTU is discovered by using the IP "Don't Fragment" (DF) bit. Basically, the sender chooses an initial packet size and sets the DF bit. When a router that needs to do fragmentation on a packet with the DF bit set, the router discards the packet, and returns an ICMP "Destination Unreachable" message to the sender. When the sender receives such a message, it knows that the packet size it is using is too big, so it reduces the packet size and retransmits the data, again with the DF bit set. This process continues until the sender receives no more ICMP "Destination Unreachable" message. After the PMTU is discovered, the sender should limit all subsequent packets to be smaller than the PMTU.

3.2.2 Application Level Framing (ALF)

Another important question related to framing is who should do the fragmentation. The current model used by TCP is that TCP treats the data from the application as an unstructured byte stream, and it breaks it down to TCP segments, which are passed to the network layer and may be further fragmented by the network layer. This dissertation shows that this is a bad model, and shows how to add framing support into TCP, which preserves application level frame boundaries. This framing support illustrates the Application Level Framing (ALF) design principle.

ALF was proposed by Clark and Tennenhouse [29] as an architectural principle for designing network protocols. Its major motivation is to allow the application to be able

to handle data packets independently one from the other. This is particularly important when packet losses occur, because this enables the losses be expressed in terms meaningful to the application and gives the applications maximum flexibility to handle the packet loss. To achieve this goal, ALF advocates that the lower layer protocols, such as the transport layer protocol, should handle data in the units that are meaningful to the application. For example, the transport layer protocol should preserve the application level frame boundaries as they process the data. The new TCP framing mode created in this dissertation is a good example of applying the ALF principle.

3.2.3 Transport Protocols with Framing Support

Next we briefly survey a few transport protocols that provide framing support.

UDP

UDP provides a simple application interface to the IP datagram service. It is a datagram-based protocol. Data in each packet is processed independently and delivered independently to the application. So if the sender places one record in one datagram, the receiver is able to get the record and recognize the record-boundary easily.

However, UDP is such a "null" protocol that it provides no reliability, no in-sequence data delivery, no control flow and no congestion control. Applications need to implement all these features themselves, which is very complicated and difficult.

XNS SPP: Sequenced Packet Protocol

The Sequenced Packet Protocol (SPP) of the Xerox Networking Systems (XNS) protocol suite is a reliable, connection oriented protocol. It provides byte stream service and optional message boundaries on top of the Internet Datagram Protocol (IDP), a counterpart to the IP protocol.

IDP/SPP is the predecessor to Novell's NetWare IPX/SPX protocol suite (Internet Packet Exchange/Sequenced Packet Exchange), which has been widely used for some time in the personal computer environment.

In terms of data representation, SPP has a three-level hierarchy:

1. Byte: same as all the other transport protocols, bytes are the basic entity.

2. Packet: a packet consists of zero or more bytes (corresponding to TCP segments).

3. Message: a message consists of one or more packets, corresponding to application-level frames.

In terms of the services it provides to the applications, it offers three interfaces to the user process:

1. Byte-stream interface: it provides in-order delivery of bytes, and preserves message boundaries. It does not preserve packet boundaries.

2. Packet-stream interface: it provides in-order delivery of packets, and it preserves packet boundaries and message boundaries.

3. Reliable-packet interface: it provides order-of-arrival delivery of packets, which may be out-of-order, and preserves packet boundary. Message boundaries are not preserved.

The two communicating entities can use two different interfaces. For example, one entity can use the packet-stream interface, while the other uses the byte-stream interface.

SPP provides a rather comprehensive range of services as far as message boundaries are concerned. Its design could be applied to other general purpose transport protocols.

However, the benefit of its reliable-packet service is questionable. Very few, if any, applications require the out-of-order delivery service. Also, the IPX/SPX successor is mainly used in a local area network, where the network latency is very low, meaning that retransmission can be done very quickly, and the likelihood that packets are re-ordered is low. So the order-of-arrival delivery service has very limited benefit.

SCTP

We introduced SCTP in section 3.1.2. One difference it has from TCP is that it is a message-based protocol, rather than a byte stream protocol. Such message-orientedness is useful for a general transport protocol.

However, this brand new protocol derives most of its features and designs from TCP. Because TCP meets almost all of the requirements of most significant applications, and the few functionalities lacking in TCP, such as framing support, can be easily added into TCP, we see no compelling reason to deploy such a new protocol.

Reliable Data Protocol (RDP)

RDP [126, 98, 97] is designed to provide high speed transfer of bulk data. It preserves data boundaries and allows both in-sequence and order-of-arrival delivery of packets. It preserves data record boundaries by sending each application record in a single packet, and each packet contains just one application record. It is connection-oriented, and it uses cumulative acknowledgement, and an extended-acknowledgement (EACK) scheme to acknowledge packets received out of sequence. EACK is similar to TCP's SACK option. It does timer-based retransmission to achieve reliability. In addition, for congestion control, it does exponential back-off when packet-loss happens.

Partridge [97] found that ordered delivery is actually as efficient as un-ordered delivery when using ring buffers, although the ordered delivery does need queuing.

In many respects, RDP is a simplified TCP, with the exception that it preserves record boundary and allows out-of-order packet delivery. Its technique of preserving application record boundary can be applied in a general transport protocol. But in general, there is no compelling reason to choose RDP over TCP for any significant application, as evidenced in the past 18 years. RDP was initially designed in 1984, but was not deployed anywhere. In addition, with the new TCP framing support, TCP can preserve record boundary as well.

3.2.4 Adding an Extra Layer for Framing

A new approach taken recently to adding framing support is to add a framing layer on top of existing transport protocols. This is motivated by network storage applications.

In the IP storage area, people have developed techniques, for example RDMA [110], to efficiently transmit data over TCP into a receiving host's memory, eliminating any extra data copying (this is also termed as "zero copy"). To achieve this goal, it is required that the header of the upper level protocol, such as iSCSI [68], in a data stream be efficiently identified, and this demands an efficient while simple framing technique, which is hardware implementable.

One notable proposal to provide such framing support is called "ULP Framing for TCP" [130], where ULP stands for "Upper-Level-Protocol". This framing mechanism is designed as a "shim" layer between TCP and higher level protocols, which attempts to preserve higher level protocol record boundaries if the record is less than or equal to the path MTU. It is designed to enable hardware acceleration of data movement operations, such as directly

placing incoming TCP segments into higher level protocol buffers, even when some previous TCP segments are temporarily missing due to packet loss.

Although out of the right intention, their approach to add a new layer between TCP and upper layer protocols to add framing support is not compelling. As shown in Chapter 5, framing support can be added easily in TCP directly, so adding a new layer is not necessary, in addition to being less deployable and potentially suffering extra header overhead.

3.2.5 A Generic Framing Mechanism: Byte Stuffing

Byte stuffing is a process to encode a sequence of data to be transmitted by using some special reserved marker as frame delimiters, so that the receiver can use the special marker to retrieve individual frames. In this process, if the special marker appears in the original data to be transmitted, it needs to be re-encoded to differentiate it from the frame delimiter. In general the result of the byte stuffing process will be longer than the original sequence of data. For many byte stuffing algorithms, such as those used by Serial Line IP (SLIP), the Point-to-Point Protocol (PPP) and AX.25 (Amateur Packet Radio) [62, 116, 6], the worst case can double the original data size. Using these algorithms to achieve framing has the obvious disadvantage of degrading throughput due to the worst potential doubling of the data size. One interesting byte stuffing algorithm called Consistent Overhead Byte Stuffing (COBS) [25] guarantees that in the worst case adds no more than 1 byte in 254 bytes in any packet. COBS can be used together with TCP-Framing to get fully reliable framing support even in the presence of resegmentting TCP proxies.

However, similar to FEC, byte stuffing involves constant computation overhead, because it requires scanning all the data in the encoding/decoding process.

3.2.6 Limitations of Previous Efforts

Previous efforts to provide framing support are either too weak, like UDP, or limited to local area network, like XNS SPP (and its successor SPX), or not compelling to deploy, like SCTP and RDP. The conclusion from this survey is that there is no widely deployed effective transport protocol that provides good framing support. In this dissertation, we show that adding framing support to a widely deployed reliable transport protocol is simple, effective and deployable.

3.3 Multicast Transport

Multicast is a very important technique for distributing information to multiple points simultaneously. It is useful for many important applications, including IPTV, multimedia tele-conferencing, and file and cache distribution, because it significantly saves both network bandwidth and server work load by transmitting just one copy of the data at the source instead of multiple copies.

IP-multicast is a network level multicast mechanism proposed by Deering and Cheriton [34, 35, 33]. It provides a best-effort service to transmit data from one point to multiple destinations. Similar to the necessity of a reliable transport protocol, such as TCP, which provides a reliable data transmission service over the best effort IP service for point-to-point communication, it is necessary to have a multicast transport protocol that provides reliability over the IP-multicast service for one-to-many communication.

Multicast transport has been a hot research topic for more than a decade. Many reliable-multicast (RM) protocols ([39, 92]) have been designed and studied, such as SRM [45], XTP [10], LBRM [54], OTERS [73], PGM [124], RMP [1], etc. This tremendous effort has obtained much better understanding of numerous issues and achieved significant conceptual progress on this topic (as shown in the following survey). Some of these protocols perform well for certain applications. For example, SRM has been deployed on the M-Bone and supports a large scale white-board application. On the other hand, despite such effort, academia and industry still lack a consensus on a standard deployable protocol that is able to meet the requirements of common multicast applications.

Back in 1988, Crowcroft and Paliwoda [60] described a theoretical multicast transport protocol that "provides a service equivalent to a sequence of reliable sequential unicasts between a client and a number of servers, whilst using the broadcast nature of some networks to reduce both the number of packets transmitted and the overall time needed to collect replies". Although they briefly described their design on handling acknowledgements from each receiver and how to do retransmissions, they stayed at an abstract conceptual level without offering detailed design and implementation, neither did they provide performance evaluation results. In other words, they expressed the goal but not the means to get there.

TCP-SMO is designed as a straightforward multicast optimization over multiple unicasts in terms of data delivery. However, we explore the relevant issues in much more depth, describe the design and implementation in more details, perform large number of experiments

and present lots of performance data. Details of TCP-SMO are given in Chapter 6.

Next, we describe the main issues facing a reliable multicast protocol, and survey the approaches taken by representative reliable multicast protocols. We also introduce two major multicast models, the host group model and the channel model.

3.3.1 Major Issues Facing Reliable Multicast Transport Protocols

The main question to answer to design a reliable multicast protocol is how to achieve reliability. There are generally two major approaches for reliability — using either retransmission or forward error correction (FEC).

For the retransmission-based approaches, the first problems to solve include determining how to detect errors, who is responsible to detect errors and how the errors are signalled. Solutions to these problems include ACK-based sender-initiated approaches and NACK-based receiver-based approaches, which are surveyed in detail in subsections 3.3.2 and 3.3.3.

To report errors, large number of feedback signals could be generated by the receivers, and such signals could cause an associated problem called *feedback implosion*, which may overwhelm the source server. Various mechanisms to address the feedback implosion problem are also explored next.

After the error is detected, the next problem to solve is to determine how the lost packet is retransmitted, what kind of error recovery structure is used, whether to use unicast or multicast, and whether router assistance is needed.

These issues and FEC are discussed in the following subsections.

3.3.2 Sender-initiated Approach

The sender-initiated approach puts the responsibility to detect lost packets on the sender. It usually requires each receiver to send back to the sender positive acknowledgement (ACK) for every (other) packet it has received. The sender uses either duplicate ACKs or a timer to detect lost packets. TCP is the best-known protocol that uses the sender-initiated approach to detect lost packets for unicast. For multicast, except for a few attempts, such as [21], few protocols use this approach. The main reason is the fear for the so called *"ACK-implosion"* problem [38] when there are thousands or even millions of receivers, a type of the feedback-implosion problem. One representative prior attempt using sender-initiated approach is SCE, which is described next.

SCE: Single Connection Emulation

Single Connection Emulation (SCE), proposed by Talpade and Ammar [121] uses a SCE sublayer between the unicast transport layer and the multicast-capable network layer, which emulates the single destination network layer interface to the transport layer. It addresses a number of interesting design issues for a multicast transport service. For example, it is connection-oriented and uses a closed feedback loop with the SCE layer aggregating the control packets. Also, the source server keeps track of the state of each group member, such as its address, port number, current window size, etc. These techniques can be applied to a general transport protocol and are also adopted by TCP-SMO.

However, unlike TCP-SMO, it uses a server-initiated connection model and does not allow asynchronous joins, which is needed for real-time multimedia multicast. Also, it does not support TCP fast-retransmit and can only do multicast retransmission, which is inefficient in the case of independent packet loss and receiver host drops. Finally, SCE does not allow a receiver to talk back to the source server through the SCE connection for the session relay purpose, which is useful for multi-sender applications such as teleconferencing.

3.3.3 Receiver-initiated Approach

The receiver-initiated approach puts the responsibility of detecting lost packets on the receiver side. Each receiver only sends the sender negative acknowledgements (NACK) when it detects lost packets. Because it is assumed that packet loss happens infrequently, the number of NACKs is much smaller than the number of ACKs in the sender-initiated approach, so it is believed that this approach can alleviate the feedback-implosion problem.

However, if a packet loss happens at a link near the source sender, it may affect a large number of receivers. So a large number of NACKs may be generated and transmitted to the sender and may cause the NACK-implosion problem. To address this problem, a technique called "slotting and damping" is usually used. First, when a receiver detects a packet loss, it does not unicast the NACK to the sender, instead it multicasts to the entire group, so all receivers can receive it. If a receiver receives the NACK from another receiver for a particular lost packet, it does not generate another NACK even if it detects the same packet loss. Second, when a receiver detects a packet loss, it does not generate the NACK immediately. Instead, it waits for a random delay, and only if it does not receive any NACK from other receivers for this packet during this delay does it send out its own NACK. XTP

and SRM are two representative protocols using these techniques.

Another notable receiver-reliable multicast protocol is LBRM. It uses positive ACKs between the source and the primary logging server, and uses NACKs between the receiver and the logging servers to provide reliability. Details are presented next.

XTP: Express Transport Protocol

XTP was ambitiously designed to support a wide range of applications, attempting to provide all the functionality of TCP, UDP and TP4, including transport-level multicast. It provides some orthogonal features that can be configured. For example, either positive acknowledgement or negative acknowledgement can be sent by a receiver.

When used for multicast, each receiver sends an NACK to the multicast group when it detects out-of-order packets. A "slotting" mechanism is used by each receiver to wait a random delay before sending the NACK packet. Also, the receiver uses a "damping" mechanism to refrain from sending NACK packets if it has received the NACK packet from another receiver. Retransmission is done through multicast.

In general, XTP is a protocol that attempts to do too much. For example, it tries to be able to run on top of the MAC layer, the IP layer or even ATM. Such un-focused attempt results in a protocol that is overly complicated. Although at the beginning the XTP effort attracted the attention of a large number of commercial and research organizations and a large XTP forum was formed, the forum became quiescent and irrelevant.

SRM: Scalable Reliable Multicast

Scalable Reliable Multicast (SRM), proposed by Sally Floyd, et al., was designed specifically for white-board-like applications, which have low data rate and do not require ordered data delivery. SRM chooses to use receiver-based reliability, i.e. when a receiver detects packet loss, usually from a gap in the sequence number, it waits a random delay before multicasting a repair request to the group. The random delay is similar to XTP's "slotting" mechanism. Other receivers that also miss the same data do not multicast the "repair request" again if it has received one request for the same data. Any node in the group that receives the "repair request" can multicast the requested data to the group. A prototype of SRM has been developed on MBone to support a large scale white board application.

The "slotting and damping" technique multicasts the NACK and the retransmission to

a group address from any receiver, so it suffers from problems of the host group model [2], such as the address allocation problem and the multicast routing problem, and won't work with the channel model.

In addition, the assumptions used by SRM basing on white-board like applications are too specialized (such as no ordering requirement), and are not applicable to most popular multicast applications, such as software distribution or Internet TV, which require in-order delivery. Also, SRM requires that every node stores all packets, or that the application layer store all relevant data. This is not suitable for real-time streaming, because for streaming, there is usually no need to store the data after it is played out.

LBRM: Log-based Receiver-reliable Multicast

LBRM, proposed by Holbrook, Singal and Cheriton [54], provides reliability by primary and secondary logging servers. There is one secondary logging server at each client site, which is a topologically localized part of the network. Each site's logging server logs packets from the multicast source, and if a packet fails to reach a site's logging server, it retrieves the packet from the primary logging server. Each receiving application requests retransmission from its local secondary logging server, instead of from the primary logging server.

LBRM is receiver-reliable, i.e., the sender only makes it possible, via the logging server, for the receiver to recover a lost packet. Receivers do not have to receive each packet and the sender does not check to confirm that each receiver has every packet. [3]

LBRM uses a statistical acknowledgement scheme for selecting a retransmission strategy, which either unicasts or multicasts a missing packet. The multicast transmission is divided into epochs. Before the start of each epoch, the source chooses a number of positive acknowledgements that are desired for each data packet, and statistically selects a number of Designated Ackers from the secondary loggers. After transmitting a data packet, if the source receives acknowledgements from all its Designated Ackers, it assumes that multicast retransmission is not needed, and waits for individual retransmission requests.

LBRM is designed for applications where the update rate is low, but a high degree of freshness is required even if packets are lost. So it is suitable for applications like cached WWW page invalidation, distributed file caching. But it does not seem suitable for real-time

[2] See Section 3.3.10 for more details on the host group model and the channel model.

[3] This property may make it unsuitable for certain applications such as software distribution, in which case the source does want to ensure that every single receiver has reliably received all the data correctly and completely.

multimedia streaming, because the update rate is high and constant. Also, for real-time streaming applications, the data usually have limited life-time and do not need to be logged.

3.3.4 Error Recovery Structures

After the error is detected, the next problem is to retransmit the lost packet. The error recovery structure can be either un-organized or organized. SRM uses an un-organized structure, because it allows each receiver that loses a packet to multicast an NACK to the entire multicast group, and it allows any receiver that receives the NACK to retransmit the lost packet. In contrast, LBRM uses an organized error recovery structure. It organizes the receivers in a hierarchy. A receiver missing a packet only sends an NACK to its local secondary logging server, which retransmits the missing packet.

The main advantage of the organized approach is its scalability, because it avoids the "crying baby" problem [51]. In the SRM case, if there is a receiver that is behind a particular lossy link, it keeps multicasting NACKs to the entire group, which forces every single receiver to handle such retransmit requests. However, with the organized approach, the handling of the crying baby is localized to one local logging server.

This hierarchical tree based technique can be applied to a general purpose transport protocol. TCP-SMO also adopts such hierarchical scheme to achieve scalability.

Two other representative protocols that use tree-based error recovery structure are RMTP and TMTP, which are described next.

RMTP

Reliable Multicast Transport Protocol (RMTP), proposed by Lin and Paul [77], provides sequenced, lossless delivery of bulk data from one sender to a group of receivers. RMTP ensures reliability with a receiver-driven approach. It uses a selective repeat retransmission scheme, in which each acknowledgement packet carries a sequence number and a bitmap. It uses statically chosen Designated Receivers (DRs) in each region of the multicast group to process acknowledgements and perform retransmissions when needed. The DRs are organized into a control tree. The DRs decides whether to unicast or multicast the retransmission depending on the number of receivers that have lost the same packet.

At a higher level, RMTP's DR tree is similar to TCP-SMO's hierarchical structure to support large scale multicast applications. However, to support multiple receivers that reside in the same local region, RMTP attempts to reinvent lots of functionalities of TCP,

which is not only totally unnecessary, but also inferior to TCP. For example, RMTP's flow control and congestion control mechanisms are rather primitive.

TMTP

Tree-based Multicast Transport Protocol, proposed by Yavatkar, Griffioen and Sudan [132], focuses on the information dissemination from a single source. It dynamically organizes the participants into a hierarchy of subnets or domains. Normally, all the interested receivers in the same subnet belong to a domain and a single domain manager acts as the representative for that domain. The domain manager is responsible for local error recovery in that domain. In addition, a domain manager is also responsible for error recovery for other domain managers in its neighborhood. The domain managers in the domain hierarchy form a control tree. The control tree employs limited scope NACKs with suppression and an "expanding ring search" to distribute the functions of state management and error recovery among many members. Each subtree (or domain) is represented by its root, termed the "domain manager", which retransmits lost packets by limited-scope (using a restricted Time-To-Live(TTL) field) multicast within the domain.

Although the control tree, if formed correctly and stays operating, can help scaling both flow control and error recovery, the whole control management protocol is rather complicated and is not fault tolerant. It is not clear how a new domain manager is chosen and how the control tree is repaired if one domain manager in the middle of the control tree fails. In addition, TMTP uses fixed rate flow control, which is rather inflexible.

3.3.5 Using Router-Assistance and Subcasting

Due to the lack of good practical reliable multicast protocols, another trend is to attempt to add reliable multicast support into routers. A number of proposals [27, 70, 71, 81, 124] try to add soft-state into routers and cache multicast packets in routers and have routers do retransmission. Other proposed support include having routers fuse ACK or NACK from the receivers. Most of these proposals have serious deployability problems, because they require too many changes to the router design. One exception is OTERS, which requires simple changes to the network layer. Next we introduce briefly OTERS and PGM, one notable protocol in this class.

OTERS: On-Tree Efficient Recovery using Subcasting

OTERS, proposed by Li and Cheriton [73], organizes receivers into a fusion tree that matches the multicast delivery tree and uses this tree to fuse NAKs and subcast retransmissions. They have done extensively simulations and shown that OTERS performs significantly better than many other RMs, including TMTP and SRM in terms of network traffic and loss recovery latency. OTERS requires two extensions to the network layer. The first is multicast route backtracing, which allows a receiver to determine the sequence of routers between a source S and itself. This could be achieved by an extension to IGMP. The second extension required is subcasting, a facility to multicast a packet over a subtree of the multicast delivery tree for a particular group, specified by the multicast address G and the router R at the root of the subtree. When the channel model is used, the subcasting can be implemented through loose source routing. Due to the use of subcasting, OTERS can repair correlated loss very efficiently.

OTERS is a simple and efficient reliable multicast protocol. But its future does depend on new facilities in the network layer and requires new router support.

PGM: Pretty Good Multicast

Pretty Good Multicast (PGM) [124], also known as Pragmatic General Multicast, was designed for applications that require ordered, duplicate-free, multicast data delivery from one source to multiple receivers. The source periodically sends Source Path Messages (SPM) to the multicast group address. A SPM provides the up-stream path from a receiver to the source sender. When data loss is detected, the receiver sends a Negative-ACK (NAK) to the next upstream hop - the address which it has received in the Source Path Message (SPM). On receiving a NAK, the next upstream hop router sends a multicast NAK confirmation (NCF) to the receiver that sends it. This message is also sent to all the other receivers below this router by multicast, and they will not send NAK again. After receiving a NAK, the source or a local receiver retransmits the lost data.

The major problem with PGM is that it needs router support and is very difficult to deploy.

3.3.6 FEC and The Digital-Fountain Approach

As mentioned in subsection 3.1.4, Forward Error Correction can be used to implement reliable multicast. It is useful to reduce the amount of feedback generated by the receivers for the sender. Some studies [91, 48, 90] also show that FEC efficiently recovers temporally correlated losses (packet losses that are close to each other).

The digital-fountain approach [16] employs FEC and performs constant broadcasting of information. It injects a stream of distinct encoding packets into the network, from which a receiver can reconstruct the original source data if the receiver receives any subset of the encoding packets equal in total length to the source data.

However, as pointed out by Li and Cheriton [72], pro-active use of FEC requires that the data be blocked and encoded in large transmission groups to reduce its bandwidth consumption overhead. This behavior adds delay to the transmission, making it not suitable for real-time applications. In addition, reactive use of FEC as part of the recovery mechanism complicates the RM protocol significantly and can delay the recovery as well.

For bulk file transfer or non-interactive streaming applications, TCP-SMO can be used together with FEC. Because TCP-SMO reduces the application-level packet loss rate, it can reduce the redundancy required in the FEC coding and the associated encoding/decoding overhead.

3.3.7 Receiver-driven Layered Multicast

Receiver-driven Layered Multicast (RLM) is not a transport level protocol. Instead, it is a higher level rate adaptation scheme that defines how the sender application encodes the streaming data, and how the receiver application selects the right layers based on its connection bandwidth.

RLM moves the rate adaptation from the source to the receiver to accommodate heterogeneity. To quote [83]: it "combines a layered coding algorithm with a layered transmission system. By selectively forwarding subsets of layers at constrained network links, each user receives the best quality signal that the network can deliver". The selective forwarding is achieved by using multiple IP-multicast groups for multiple layers of coding, and each receiver specifies its level of subscription by joining a subset of the groups.

This approach addresses both the static heterogeneity of link bandwidths and the dynamic variations in network capacity (i.e. congestion).

It "combines a layered compression algorithm with a layered transmission scheme [115, 122]. In the approach, a signal is encoded into a number of layers that can be incrementally combined to provide progressive refinement. By dropping layers at choke points in the network – i.e. selectively forwarding only the number of layers that any given link can manage – heterogeneity is managed by locally degrading the quality of the transmitted signal."

3.3.8 Commercial Reliable Multicast Products

TIBCO Inc. provides a product called Information Bus, which uses a reliable multicast transport for distribution of (delay intolerant) financial information.

GlobalCast Communications Inc. [4] offers RMP (Reliable Multicast Protocol), GSRM (Generic Scalable Reliable Multicast), RMTP, RMTP-II and PGM. RMTP-II is derived from RMTP, and is designed to support network monitoring and resource management from a central location.

RMP [129], proposed by Whetten, Montgomery and Kaplan, is targeted at realtime, collaborative environments. A rotating token scheme, called Post-Ordering Rotating Token, is used to ensure reliability. Under this scheme, a single token is passed between sites and designates the site to multicast an ACK for the recently received packets. NAKs are multicast to avoid NAK implosion. Lost packets are multicast to all receivers. RMP may have good performance on a low latency and reliable LAN, but on a less reliable WAN with higher latency, the use of the token ring scheme is not robust. For example, RMP does not address how to handle the case when the token is lost or the site holding the token crashes.

Lucent Technology Co. has patented RMTP and built its "e-cast" product on RMTP.

MFTP

StarBurst Communications, Inc patented Multicast File Transfer Protocol (MFTP) [63]. MFTP was designed for non-real-time delivery of one-to-many data and intended as an extension to the point to point FTP. MFTP provides receiver-oriented reliability with aggregated selective NAKs used to identify lost packets. It uses FEC in retransmission of lost packets. It also provides three modes of group management, which are Closed, Open Limited and Open Unlimited. For the first two modes, the source server maintains states

[4] GlobalCast has been acquired by Talarian Co, which has been later acquired by TIBCO Inc.

of all the receivers, and full reliability is provided. One problem with MFTP is that the transmission rate has to be pre-defined by the sender application at the beginning, and does not easily adjust to the receivers' processing speed and the network condition.

The main problem with these commercial product is that they are all specialized and proprietary protocols designed for specific applications, rather than general purpose transport protocols. Therefore, they are expensive and cannot be widely deployed.

3.3.9 Stream Splitting

In practice, because IP-multicast has not been widely deployed, a technique called *stream splitting* [79, 67, 95, 87, 5, 3, 2] has been used to achieve some benefits of multicast. When multiple recipients need to get the same media stream at the same time from a remote media server, instead of letting each local recipient connect to the remote media server individually and making multiple remote connections, one local proxy server is used to make one connection to the remote server. It gets one stream of the media of interest, splits it and relays it to each recipient in the local network, as illustrated in Fig. 3.1.

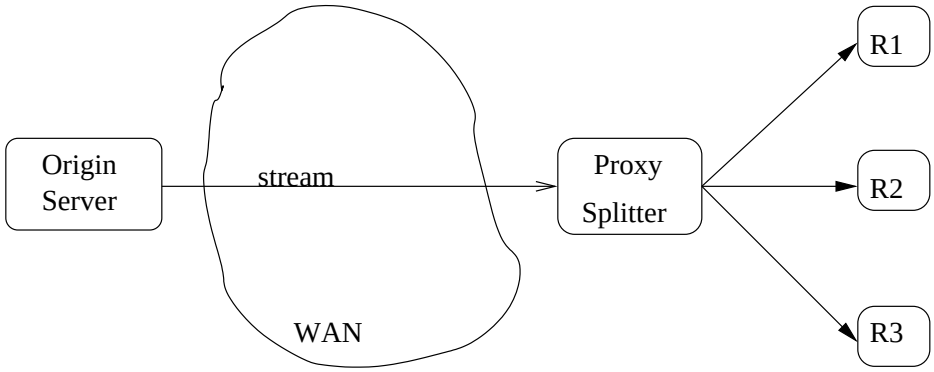


Figure 3.1: *Stream Splitting*

Stream splitting is usually used for serving live media or pre-scheduled playback to multiple recipients simultaneously. To serve multiple recipients asynchronously, a related technique called *stream caching* [79, 67, 95, 87, 5, 3, 2] can be used. When the first user requests the content, a connection is made to the origin server to retrieve the data. The data is then cached at the proxy server, and it is used to serve other users at various times later.

Stream splitting saves the upstream bandwidth usage and reduces the load at the origin server. In addition, it is relatively easy to deploy, because it does not require any change to any of the end-receiver systems and does not require the IP-multicast upgrade of the routers in the local area network. Also, multiple proxy servers can be deployed at bottleneck points in the network and perform hierarchical splitting to achieve better scalability.

When IP-multicast is deployed in the local network, stream splitting can work together with TCP-SMO to cut the bandwidth usage to downstream users of the local network (in addition to cutting the upstream bandwidth usage) and reduce the load on the proxy server by transmitting just one copy of the data to multiple recipients. In addition, the hierarchical splitting scheme is analogous to TCP-SMO's hierarchical relay model.

Furthermore, due to licensing restrictions, some origin servers do not allow local proxy servers to perform stream splitting, because it cuts the hit-rates at the origin servers and may hurt their hit-rate-based revenues. Using TCP-SMO, by taking advantage of its connection-oriented model, the proxy server can easily keep track of the number of receivers and the connection duration of each receiver (at the same time of performing multicast), and it can report such statistics to the origin server through an accounting connection to relieve the concern over the loss of revenues resulting from stream-splitting. [5]

3.3.10 Host Group Model and Single Source Channel Model

Deering and Cheriton [34, 33, 35, 36] defined a host group model for multicast. In this model, a range of host addresses are reserved as multicast group addresses. A potential receiver who desires to receive data joins a host group by issuing some command to the network. The network elements set up appropriate states in the network to ensure that any host on the network can transmit data to the group address and the data can be delivered to any host that has joined the group.

The channel model, proposed by Holbrook and Cheriton [53, 52], provides one-to-many delivery, in contrast to the many-to-many delivery of host groups. Each channel is identified by an address pair (S, G), where S is the address of the source server and G is the group address. The channel model solves two problems inherent with the host group model. The first problem is the difficulty of providing scalable, enforceable allocation of host group addresses. The second problem is the difficulty to scale the multicast routing for the host group model.

[5] Of course, the proxy server has to be honest in reporting the statistics.

In our work on extending TCP to support multicast, we adopt the channel model to take advantage of its scalability and deployability.

3.3.11 Limitations of Previous Approaches

Most of these reliable multicast protocols try to target very large scale multicast applications, having possibly millions of receivers. We think such lofty goal introduces substantial complexity into their design. In contrast, we target some common medium-scale multicast applications, and rely on a hierarchical structure to service a larger scale audience when needed.

Also most of these protocols (with the exception of RMP and a few others) are mainly designed for fully reliable applications only. TCP-SMO, however, is designed to support both fully reliable multicast of bulk data and semi-reliable multicast of real-time multimedia data.

As far as we are aware, only a few of the RMs (such as SRM and RMTP) have real implementation. Many of the RMs only have a simulated implementation based on some network simulators such as ns [125], so their performance results on a real operating systems and in a real physical network are not available. In addition, we are not aware of any published results obtained from experiments conducted over a thousand receivers.

Many RMs require the complicated setup of a DR (designated-receiver) tree and data recovery tree. In contrast, TCP-SMO is very simple operationally and does not involve any of these special processing. In addition, TCP-SMO is based on the mature protocol TCP. Therefore, we believe it is more robust than many other RMs.

In addition, many RMs require retransmission from a DR or some other end-host other than the source server in the case of packet loss. This introduces serious security problems. Furthermore, for real-time streaming applications, since no receiver actually stores the data, so retransmission from any host other than the original server is difficult.

In summary, none of these protocols meets all the major application requirements and is deployable.

3.4 Other Work to Extend TCP

The most notable is Berkeley's Daedalous project [13, 12], whose general philosophy agrees with ours, which is, in the context of today's Internet, to extend an existing widely deployed

protocol TCP, which is effective for most applications, to meet new application requirements, instead of inventing brand new protocols, such as SCTP, which will be extremely difficult to deploy.

TCP's performance suffers significantly in lossy wireless networks, because TCP treats packet loss as congestion signal and reduces its congestion window which limits its throughput. The Daedalous project has designed a TCP-aware link protocol called the *snoop protocol*, which isolates the wired sender from the lossy characteristics of a wireless link. A snoop agent running on the base station maintains a cache of the TCP packets that the wired TCP sender sends to a wireless host, and that have not been acknowledged by the wireless host. If a packet is corrupted and dropped, the snoop agent detects that by either duplicate ACKs or local timeout, and then retransmits it. Thus, the congestion window on the wired host is not reduced due to corrupted packets, and TCP can achieve higher throughput.

On networks with asymmetric bandwidth, such as Direct Broadcast Satellite (DBS) Networks and Asymmetric Digital Subscriber Loop (ADSL) networks, the forward bandwidth is much higher than that of the reverse path. TCP's performance not only requires that the data packets be reliably and quickly delivered, it also relies on that the acknowledgements are returned in a timely fashion, so that the send window can move forward, and congestion window can be enlarged. On an asymmetric network, the acknowledgements are more likely to be lost due to buffer overflow on the reverse path, thereby TCP's throughput can be affectedly adversely. To solve this problem, they have designed two major techniques: Ack Congestion Control (ACC) and Ack Filtering (AF). ACC uses ECN (Explicit Congestion Notification) to mark ACK packets on the reverse path, in addition to marking the data packets on the forward path. When the receiver receives a packet with ECN set, it reduces the frequency of ACK generation, thereby reducing ACK congestion. In addition, AF is implemented in a gateway on reverse path. It takes advantage of the fact that TCP ACKs are cumulative, and remove some previous ACKs for the same TCP connection in the gateway's queue to free up buffer space and avoid dropping new ACKs.

Many Web connections are short. If a packet is lost, and fewer than three packets are sent after the lost packet, the sender will not receive three duplicate ACKs to trigger fast-retransmit, because each duplicate ACK is triggered by a new data packet. Thus such packet loss has to be recovered by a coarse timeout, which is inefficient. They have proposed a technique called *right-edge recovery*, which has the sender send out a new data segment,

with a sequence number higher than any outstanding packet, when each duplicate ACK arrives. When this packet arrives at the receiver, it will generate an additional duplicate ACK, which assures the sender that the previous packet is lost and can be retransmitted.

These work further support our thesis to unify the transport layer of the Internet.

3.5 Summary

In this chapter, we have surveyed transport level efforts to support real-time applications, record-based applications and multicast applications. We showed that a proliferation of special protocols try to provide real-time specific capabilities in the transport layer and try to provide reliability for multicast applications. However, none of these protocols have achieved the maturity and acceptance of TCP.

In contrast, we find that TCP is very flexible and extensible. In this dissertation, we take a different approach from the majority of the existing approaches and show that a small number of low-cost deployable extensions allow TCP to support these applications well.

Chapter 4

Real-Time Mode

The Internet is increasingly used to deliver real-time video and audio, for asynchronous playback for entertainment and education, and interactive applications such as teleconferencing. The increasing capacity of the Internet and its cost-effectiveness compared to conventional analog distribution networks suggests this trend will accelerate. So effectively transmitting real-time contents is an important problem.

The conventional wisdom is that TCP is un-suitable for this type of traffic because it favors reliable delivery over timely delivery, the wrong trade-off for real-time traffic. This thinking dates back at least to the famous separation of TCP from IP prompted by the work of Danny Cohen [30] and others [61]. Consequently, most real-time protocols have been built on top of UDP; operation over TCP is only considered as a compromise to enable communication through firewalls.

Conventional wisdom says that real-time applications can tolerate loss more than jitter. However, in real-time streaming with video and audio, the content is compressed to reduce data rate, making it more sensitive to loss. For example, video is compressed as an MPEG-2 stream such that a single packet loss corrupting an I-frame can disrupt the video reception for significant fractions of a second. Moreover, these applications often use a significant playback buffer on the order of hundreds of milliseconds (or even larger in the case of one way streaming), masking the network jitter to the application. Recognizing the reality that these streaming real-time applications are in fact more sensitive to loss than jitter, at least within bounds, TCP error recovery provides significant benefit. [1]

[1] The alternative of sending raw data to make a real-time stream less loss sensitive is not cost-effective, because the raw data requires significantly higher network bandwidth, thus increases significantly the cost

With all the clear benefits for using TCP, it seems attractive to investigate whether there are modest TCP-level modifications and application-level techniques that would allow real-time applications to use TCP.

In this chapter, we describe techniques to allow TCP to be used for real-time streaming applications such as audio and video delivery, including interactive applications. We conclude that these techniques, including providing a real-time mode (RTM) extension to TCP, make its use superior to the conventional UDP-based approach for these real-time applications. In addition, TCP-RTM is much easier to deploy than special new transport protocols.

The next section describes the TCP-RTM modifications. Section 4.2 describes application-level techniques to use TCP-RTM for real-time applications. Section 4.3 presents the performance results obtained from testbed-based and public Internet experiments. Section 4.4 describes the congestion behavior of TCP-RTM and the Selective NACK extension. We close with some conclusions.

4.1 TCP-RTM Modification

In this section, we describe the few simple extensions we make to TCP to make it suitable for real-time applications.

4.1.1 TCP-RTM Packet Reception

TCP is modified to support real-time mode (RTM) by two simple changes to the TCP packet reception algorithm, namely:

1. On application-level read of the TCP connection, if the in-sequence data queue is empty, but one or more out-of-order packets are queued for the connection, the first contiguous range of out-of-order packets is moved from the out-of-order queue to the receive queue, the receive pointer is advanced beyond these packets, and the resulting data is delivered to the application.

of the entire system. For example, to transmit a video stream with a raw data rate of 10Mbps, using the approach of compression plus retransmission on loss only needs to use a bandwidth of about 1.01Mbps, assuming a compression ratio of 10 and a network loss rate of 1%. So sending the raw data is almost 10 times more expensive in this case.

2. On reception of an out-of-order packet with a sequence number logically greater than the current receive pointer (rcv_next_ptr) and with a reader waiting on the connection, the packet data is delivered to the waiting receiver, the receive pointer is advanced past this data and this new receive pointer is returned in the next acknowledgment segment.

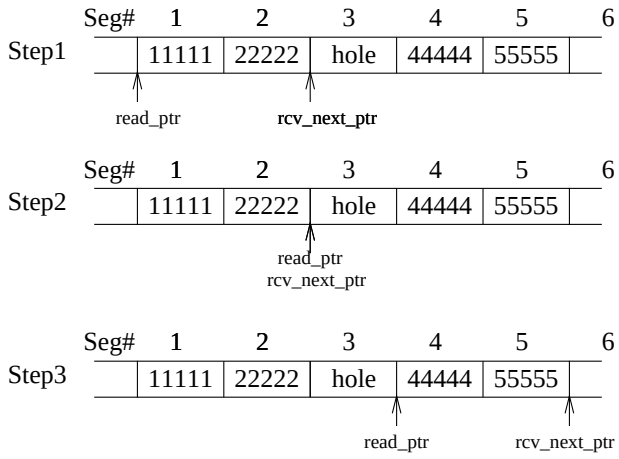


Figure 4.1: *Scenario in TCP receive buffer.*

Figure 4.1 illustrates this reception behavior. Here, segments 1, 2, 4 and 5 have been received and stored in the buffer, segment 3 has been dropped and segment 6 has not arrived. When the user process is requesting data in segment 1 and 2, TCP-RTM behaves exactly the same as regular TCP (Step 1 in Fig.4.1). After data in segments 1 and 2 are consumed, the read pointer points to the beginning of a "hole" (Step 2 in Fig.4.1) . If the hole is not filled by retransmission of segment 3 by the time the user process requests new data, TCP-RTM skips over the hole and moves the read pointer to the beginning of segment 4 and starts returning data from there (Step 3 in Fig.4.1). In the mean time, TCP-RTM sets the receive-next pointer to the end of segment 5 plus 1, which is used when an ACK is sent to the sender.

With this modification, the receiver thread is never blocked from receiving the latest data by packet loss and retransmission. Consequently, if multiple data packets are lost, defeating fast retransmission, the receiver is not blocked waiting for retransmissions. Similarly, if the round-trip increases to exceed that allowed by the playback delay, the receiver is able to continue. Thus, the receiver does not experience significant periodic delays which would

otherwise occur with high packet loss or long round-trip times.

In our Linux implementation of TCP-RTM [2], this modification required changes to only 50 lines of code, representing far less than one percent of the TCP implementation. This includes a new socket option that the receiver uses to turn on TCP-RTM mode, enabling the above modifications to TCP behavior.

The TCP-RTM modification requires the application to deal with recognizing and handling losses out of the TCP byte stream. Given that most such applications transmit data in records, this requires some form of application-level framing that works with TCP-level packet loss, as discussed in Chapter 5.

4.1.2 Sender-side TCP-RTM

In general, the sender's send buffer should be large enough to accommodate all the data segments that have not been acknowledged. However, in the case that the sender's send-buffer is full due to large amount of backlogged data, TCP-RTM discards the oldest data segment in the buffer and accepts the new data written by the application. TCP-RTM also advances its send-window past the discarded data segment. This way, the application write calls are never blocked and the timing of the sender application is not broken.

At the receiver side, because TCP-RTM uses the skip-over hole technique, old packets that are discarded by the sender will not cause any problem, and will be "skipped-over" by the receiver.

In addition, the sender TCP-RTM keeps track of a per-connection *loss-counter* that counts the number of segments which have failed to reach the receiver, which are either discarded by the sender or lost in the network and skipped-over by the receiver. The number of receiver-skipped-over packets will be reported to the sender by the SNACK option introduced in 4.4.3. The value of the loss-counter can be queried by the sender application and used by the application to adjust its encoding scheme and transmission rate to the network condition.

In this section, we described extensions to TCP to make it suitable for real-time applications. In the next section, we describe new techniques that real-time applications can use to take advantage of TCP-RTM.

[2] We made our modifications based on the Linux 2.2.12 kernel, using a TCP-Reno implementation with SACK.

4.2 Using TCP-RTM and Application-level Techniques

A few simple application level techniques enable real time multimedia applications to operate well over TCP-RTM. We describe these techniques in the following subsections.

4.2.1 TCP-sized Playback Buffer

Real-time streaming applications use a playback buffer to insulate the playback from *jitter* arising from the variable delay through the Internet. A receiver application normally delays processing to operate with a *playback delay* of *pbd* milliseconds later than the source so that a packet is only late in its processing if it is delayed by more than *pbd* milliseconds.

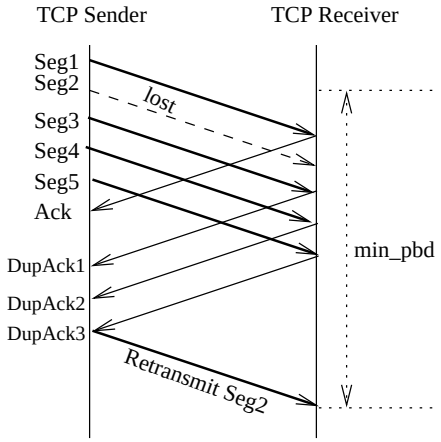


Figure 4.2: *Derivation of the minimum pbd.*

For real-time applications using TCP-RTM, as illustrated in Fig.4.2, the receiver thread is programmed with $pbd \geq (3/2 * RTT + 3 * period) + delta$ [3], where RTT is the TCP round-trip time and *period* is the typical interval between transmissions of data on the stream and *delta* is some modest extra time. For instance, for a typical audio stream with $period = 20ms$, $RTT = 60ms$ and $delta = 10ms$, then $pbd \geq 160ms$.

A real-time application using TCP-RTM is programmed to use the TCP receive buffer for this playback buffer by delaying its play-out from TCP after stream initiation for this time period, and then only reading from TCP at the real-time rate of the stream (so it remains *pbd* milliseconds behind the sender). [4]

[3] For asymmetric connections, it should be refined as $pbd \geq (RTT + forward\_latency + 3 * period) + delta$.

[4] Of course, the application can also use an application-level playback buffer in the mean time and can

With the playback delay set as above, TCP can recover from a packet loss by fast retransmit within the playback delay. In particular, after a data packet loss, the receiver receives the next 3 packets as out-of-order packets, causing 3 duplicate ACKs to be sent back to the sender. The sender thus receives the third duplicate ACK after $RTT+3*period$, causing it to retransmit the last un-acked portion of the stream, which corresponds to the dropped packet.

With this behavior, the application processing is normally not delayed and experiences no additional jitter by using TCP. Moreover, the application is oblivious to the packet loss, providing better quality play-out. In audio and video applications, especially with content compression, packet loss can seriously degrade the play-out quality otherwise. Even a 1% loss rate, especially on "important" packets, such as MPEG I frames, can make the result intolerable [86, 93, 114] for video streaming applications.

Burst drop from tail drop in overflowing network queues is well known as a behavior to avoid. Single packet loss is the most common drop situation in a well-managed network because it is commonly used to signal congestion to TCP flows that are ramping up to maximum throughput [44]. The combination of single-drop signaling and TCP congestion control has been shown to allow networks to operate efficiently while avoiding congestion collapse. Routers using active queue management, such as DBL [41] or other RED-like mechanisms [44, 76], instead of tail drop, are already being deployed [43, 119]. These routers normally only drop a single packet for each flow to notify it of congestion. Thus, it seems reasonable to expect TCP-RTM to primarily encounter single packet drop by the time it is deployed.

In some real-time streams, such as an MPEG video, a multi-packet unit such as an I-frame may be delivered as a burst. With a packet loss in this case, there are typically several subsequent packets received within a short-time, causing three duplicate ACKs within a shorter time than the normal $3 * period$, causing the fast retransmission to occur even more quickly. Moreover, packet loss from congestion is in fact more likely in the case of such bursts. Thus, many recoveries are expected in less time than *pbd*.

4.2.2 Application-Level Heartbeat

A TCP-RTM real-time application is programmed to periodically send an application-level packet back to the source if it has not received any data within the heartbeat time period.

read all in-sequence data as soon as it is received by TCP.

The *heartbeat period hbp* is set to $hbp = 3 * period$ ensuring that, if a single ACK is lost, an extra ACK is generated by the heartbeat packet.

The heartbeat message is like the "uh-huh" in human conversation in some aspects. Here, however, it prompts TCP to piggyback the ACK information that reassures the source that the data are being received and keeps the data stream flowing. It can also make sure the sender can increase its congestion window size in a timely fashion during slow-start. In addition, it can also carry application-level information, such as an indication from the receiver on the amount of loss it is observing, providing the source with additional information on which to act. For instance, the source may change its encoding and/or report the loss behavior to the user.

The application can easily check for receipt of new (in-order) data by querying the TCP buffers of its connection each time that it wakes up. Because the heartbeat is only sent when the receiver has not received new in-order data, it imposes low overhead on the network in the expected case of low packet loss and, as a small packet, it imposes minimal percentage overhead in any case.

Paxson's study [100] showed that Internet packet loss is usually uncorrelated between the forward path and the reverse path. Our experimental results presented later show that in the case when packet loss only occurs in the forward path, just using TCP playback buffer already provides good performance under reasonable network loss rates. In the uncommon case that both the forward path and the reverse path are losing packets, the loss of some critical ACKs may cause unnecessary delay on the data sender side.

For example, in Fig. 4.3, segment A is retransmitted due to RTO timeout. Because of slow-start or other congestion window restrictions, no more packets can be sent out until the ACK to segment A comes back. Although this retransmitted packet is received successfully by the receiver, the loss of the ACK forces the sender to double its RTO value, block until after a second timeout and retransmit segment A again. This loss of ACK introduces unnecessary long delay and another retransmission.

Therefore, it is important to ensure that important ACKs are received at the sender in a timely fashion, and the application level heartbeat technique is designed to achieve this goal.

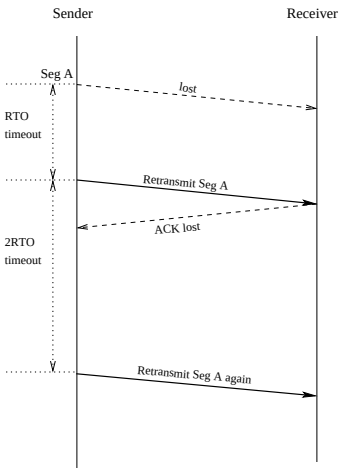


Figure 4.3: *When the ACK to the retransmitted packet is lost, the sender cannot proceed and has to timeout and retransmit that packet again, causing unnecessary 2nd retransmit and long delay.*

4.2.3 Framing Techniques

For applications with fixed-sized frames, typically true for audio applications, the following method is generally sufficient. The `TCP_NODELAY` socket option is used at the sender to turn off the Nagle algorithm, so that each application level frame is transmitted immediately after it is written by the sender application (this avoids delay and avoids combining the frame with another frame in a TCP segment) [5]. In addition, the TCP MSS (Maximum Segment Size) is set in such a way that each TCP segment contains one or integral multiple number of application level frames. Consequently, each TCP packet loss corresponds to one or integral number of application-level frame loss. At the receiver, each read request uses the fixed frame size as the size of the read. Therefore, each read request gets one application-level frame.

Further application-level sequencing can make the packet loss evident to the receiver. For instance, the RTP sequence number can be used for this purpose with TCP-RTM

[5]The Nagle algorithm was mainly designed to avoid small packets for applications like rlogin.

applications using RTP [112]. Note that this is not an extra burden on the TCP applications because UDP-based applications are already doing similar things to deal with missing frames.

For applications with variable-sized frames, TCP *framing mode*, as described in Chapter 5, needs to be used.

4.2.4 RTM Receiver

A TCP-RTM receiver application uses non-blocking reads and always has the read complete immediately, only returning no data when no data has been received since the last read.[6] The application does, however, have to be prepared for there being data missing from the stream because of packet loss. For instance, an audio application may interpolate between the next sample and the previous sample to compensate for the loss of the current sample, playing out this interpolated sample and saving the next sample for the next time period. In this way, it can attempt application-specific recovery to loss and still maintain the same playback delay.

As the round trip time and the loss rate of the connection increase, the application confronts a trade-off between the playback delay and the level of application-visible packet loss. With unidirectional streaming applications, the playback delay is not significantly constrained. However, with interactive applications, excessive playback delay starts to interfere with interactivity. For instance, in the context of a teleconference, excessive playback delay means longer gaps between speakers plus delay in interrupting one speaker with a comment from another. On the other hand, packet loss also detracts from interactivity. Packet loss means that you may not understand all the words spoken by a speaker, causing you to ask for retransmission at the human level. In either case, the communication is less efficient. It seems appropriate in these applications to allow user control of this trade-off when the network environment is fundamentally this difficult.

The combination of the application level techniques plus the TCP-RTM extension appears to allow real-time applications to operate well over TCP over lossy networks. We describe our evaluation of this hypothesis next.

[6]The application can also delay slightly the issuance of the read request when it senses that there is no in-sequence data available by checking the size of the receive socket buffer. This technique allows the application to receive and play back data that is delayed slightly beyond the *pbd*.

4.3 Performance Evaluations

We performed extensive experiments to evaluate the performance of the techniques discussed above. In the following subsections, we first present the performance results of just using the application level techniques. Then we show the results of applying the TCP-RTM extension from both testbed-based experiments and public-Internet experiments. Finally, we present the user-level quality of service evaluation results.

4.3.1 Performance Evaluation of the Application-Level Techniques

To evaluate the performance of application-level techniques, we simulated an audio-like real-time application[7]. An audio application was selected because audio is the most delay-sensitive, is well-characterized, and the short periodic audio samples are the most challenging case to handle. As observed earlier, bursty traffic like video can cause earlier fast retransmit so it is less demanding in general. It is also less sensitive to delay from a human factors standpoint. To emulate packet loss and delay, we used the Nistnet emulator [19] installed on a router between the sender machine and the receiver machine, as shown in Fig. 4.4.

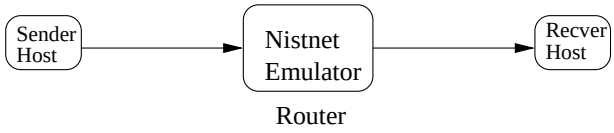


Figure 4.4: *Simulation topology.*

For these simulations, we used a playback delay of 250 milliseconds, which is generally considered acceptable for interactive audio applications such as teleconferencing [86, 101] [8]. This allows a round-trip time of up to roughly 120 milliseconds, sufficient for operation across North America in general. In practice, a TCP real-time application could adapt the playback delay to the actual TCP-determined round-trip time.

Our simulations focused on single packet losses because this case is the most common. Burst losses of two or more packets are an order of magnitude less common [50, 101, 131].

[7] A data frame stream of 100 bytes per frame at 20ms intervals was used, which resembled typical audio traffic. 5000 frames were transmitted in each of the simulations, which corresponded to 100 seconds of streaming medium.

[8] For interactive video applications, the playback delay can be higher, up to 500 milliseconds [18], which will result in even better error recovery performance.

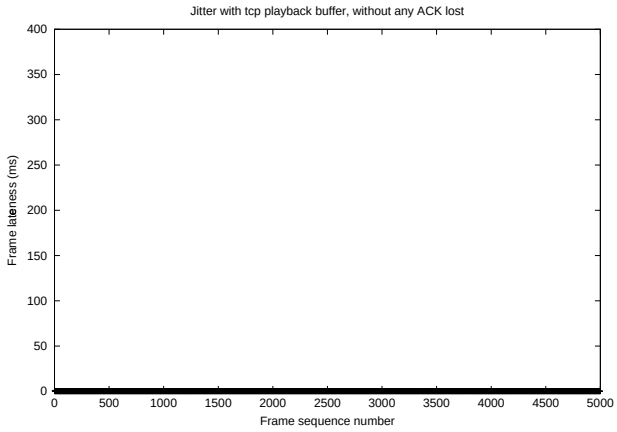


Figure 4.5: *Playout jitter with the TCP playback buffer, with RTT=100ms, and network drop rate=4%. No ACK on the reverse path is lost. Note that almost all data points have value zero and fall on the X axis.*

Also, as mentioned in section 4.2.1, routers using AQM that provides early congestion notification by single packet drop is being deployed. However, to be more robust, simulations under burst losses were also performed and the results, though a little degraded, were basically in line with the results under single packet losses. Such results are presented in section 4.1.

In the following experiments, we first simulated a TCP real-time application with just the TCP-sized playback buffer and then added each new technique to determine the benefit of each.

In the following figures, *Frame Lateness* is defined as:
$MAX(frame\_arrival\_time - (frame\_transmit\_time + pbd), 0)$. So if a frame is received before *pbd*, in time for playback, its lateness=0; otherwise its lateness is a positive number. *Jitter* is loosely defined as the overall lateness values of all the frames. Fig. 4.5 shows almost no jitter observed with just the TCP playback buffer, when the data packet loss rate is at 4%. Fig. 4.6 shows the jitter observed when the ACK packets suffer 4% loss rate. Fig. 4.7 shows that the application level heartbeat technique substantially reduces the jitter.

In a reasonably well managed network, it is the common case that the packet loss rate stays below 4%. As noted before, a 1% loss rate may render the quality of some real-time applications intolerable, while using just these two simple application-level techniques can enable such applications to achieve good performance, in terms of recovering all lost packets

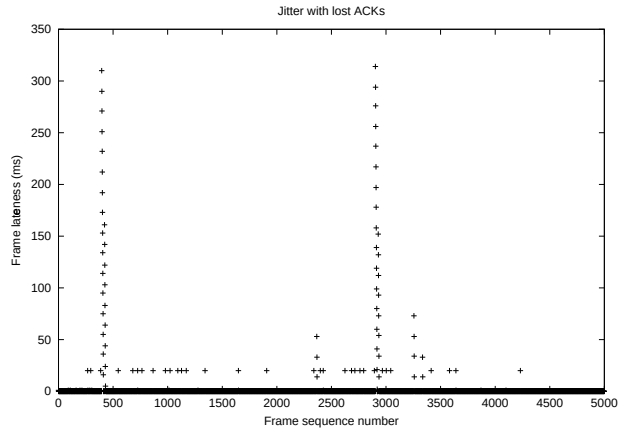


Figure 4.6: *Playout jitter with the TCP playback buffer, with RTT=100ms, and network drop rate=4%. ACKs on the reverse path also suffers 4% loss rate.*

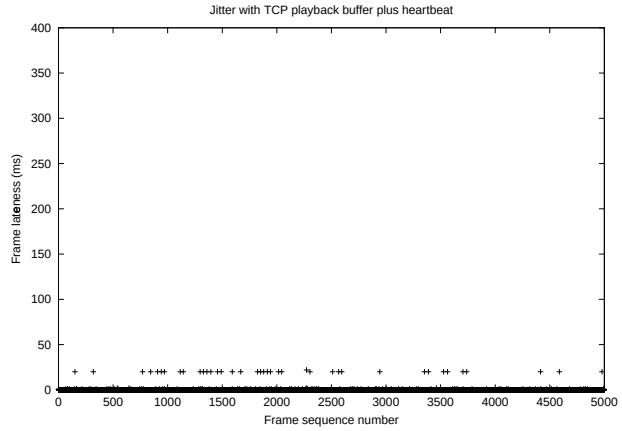


Figure 4.7: *Playout jitter with the TCP playback buffer and application-level heartbeat, with RTT=100ms, and network drop rate=4%. ACKs on the reverse path also suffer 4% loss rate.*

and introducing low jitter, under loss rate as high as 4%. With these two techniques, TCP real-time applications perform well in networks with low levels of packet loss provided that the round-trip time remains relatively stable and within the range used to compute the *pbd*. To address situations in which the packet loss is higher and round-trip times can temporarily spike, TCP-RTM techniques need to be utilized. We next evaluate the performance of the RTM techniques.

4.3.2 Performance Evaluation of RTM Techniques

We implemented these RTM extensions in a linux kernel, using TCP-Reno and SACK. In the evaluation process, we followed the guidelines proposed by Allman, et. al. [8] by combining emulation-based testbed experiments with real Internet experiments across the North America continent to perform the evaluation more realistically. In addition, we built a simple telephone system on top of TCP-RTM to perform voice communication between two PCs over a network and evaluated the audio quality in the presence of network latency and packet loss. Furthermore, TCP-RTM's congestion behavior in the presence of competing traffic was evaluated; see Section 4.4. In the following, we describe our experiments and measurement.

Testbed-based Experiments

Using the same topology as shown in Fig. 4.4, more experiments were performed to evaluate the TCP-RTM read-over-hole technique, using a wide range of network delay and packet loss rate parameters to emulate the complex and diverse conditions of the real Internet.

Section 4.3.1 has shown that when the network drop rate is below 4%, the two application-level techniques enable real-time applications to operate well over TCP. However, increasing the packet loss rate further causes fast retransmit to fail, causing jitter at the receiver. Our simulations show that a packet loss rate at 5 percent and higher causes this to be a significant problem. Fig. 4.9 shows the lateness-deviation [9] as a function of packet loss rate. It shows that the lateness-deviation is significantly reduced by the heartbeat technique, and further cut in half by the RTM read-over-hole technique.

Introducing the TCP-RTM modification to TCP, jitter is further reduced, as shown in Fig. 4.8 and 4.9, even with these higher loss rates. Moreover, even with high loss rates, TCP-RTM manages to recover a substantial number of dropped packets compared to a conventional UDP-based approach, as illustrated in Fig. 4.10.

As this figure illustrates, application-perceived packet loss rises exactly with increasing network packet loss with a UDP-based approach. However, using TCP-RTM, the packet loss rate is substantially lower, especially at the shorter round-trip times ($\leq 60ms$) where more than 90% of the packets dropped by the network can be recovered, indicating that applications that may tolerate 1% loss rate can actually operate well over lossy networks

[9] Lateness-deviation is defined as $\sqrt{\sum_{i=1}^{n} L_i^2/n}$, where L_i is the lateness of the i-th frame, and n is the total number of frames.

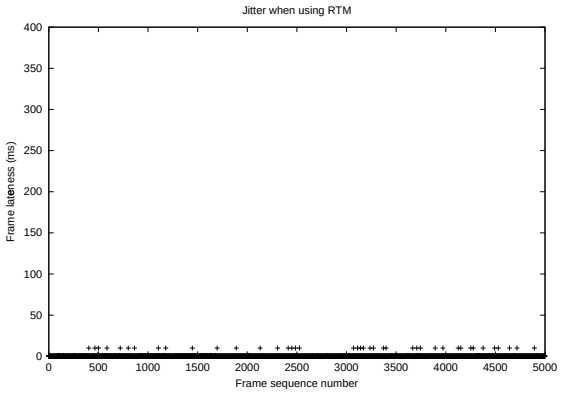


Figure 4.8: *Play out jitter with TCP-RTM and TCP playback buffer and application-level heartbeat, using a RTT= 100ms and network drop rate=4%.*

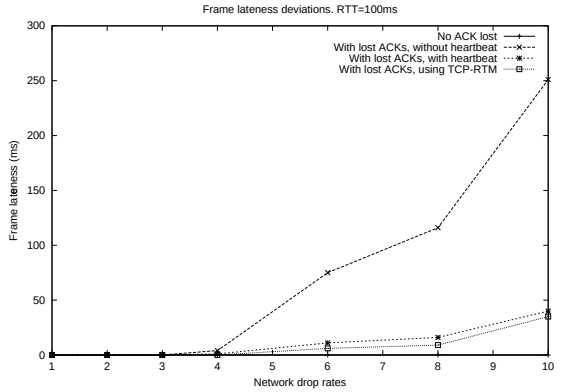


Figure 4.9: *Deviation in lateness as a function of packet loss rate, using a RTT= 100ms. Note that the curve for the case when no ACK is lost falls completely on the x-axis, indicating no jitter at all in this case.*

with near 10% loss rate. The jitter at the application level is also significantly lower (Fig. 4.11, 4.9).

Thus, the application behavior is substantially better with TCP-RTM, especially when the network drop rate is below 4% or RTT$\leq 60ms$, which is the common case in the current Internet. Of course, the same behavior could be accomplished by duplicating all the TCP packet retransmission mechanism on top of UDP but this seems hard to justify, given the good performance of TCP.

Although we concentrate on a loss model of single packet loss, we also evaluated RTM with some bursty traffic. Fig. 4.12 and 4.13 show the results under the same condition

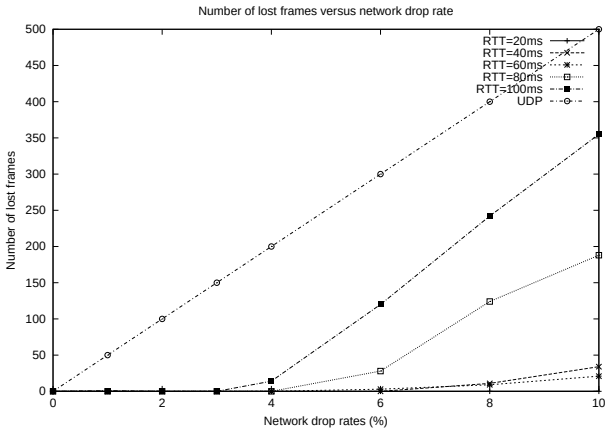


Figure 4.10: *For RTM, number of lost frames (out of 5000 frames) vs. network drop rates with playback delay = 250 ms.*

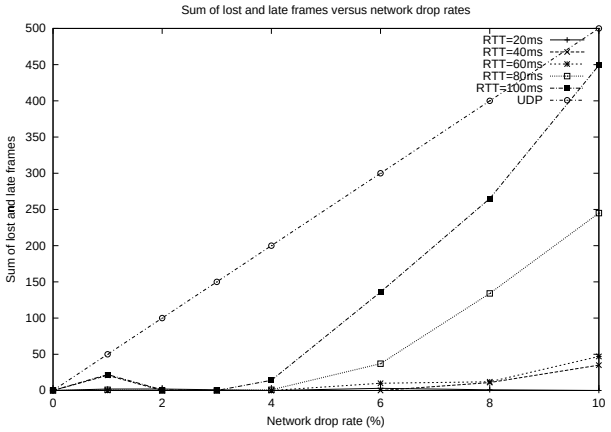


Figure 4.11: *For RTM, sum of the numbers lost frames and delayed frames vs. network drop rates with playback delay = 250 ms.*

except that the network packet loss may be bursty, with near 10% of the packet loss having a burst length of 2 or longer. These figures show that RTM performs also well.

Public Internet Experiments

In our emulation-based experiments, we used real TCP implementations on Linux and emulated a wide range of network link latencies and loss rates. Although the topology was simple, we believe these experiment configurations were good abstraction of the public Internet, and we believe the experiment results were valid. However, to further evaluate

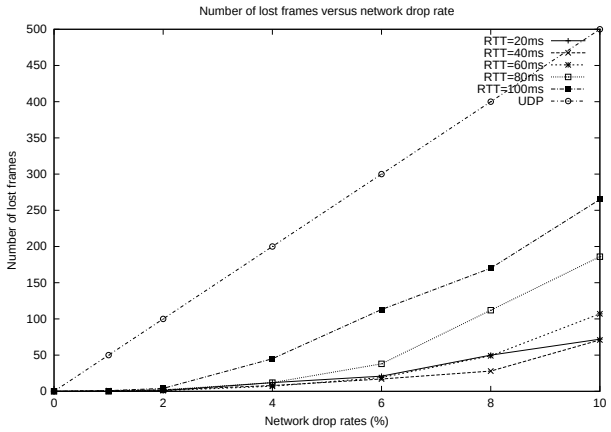


Figure 4.12: *For RTM, number of lost frames vs. network drop rates with playback delay = 250 ms and bursty data packet loss.*

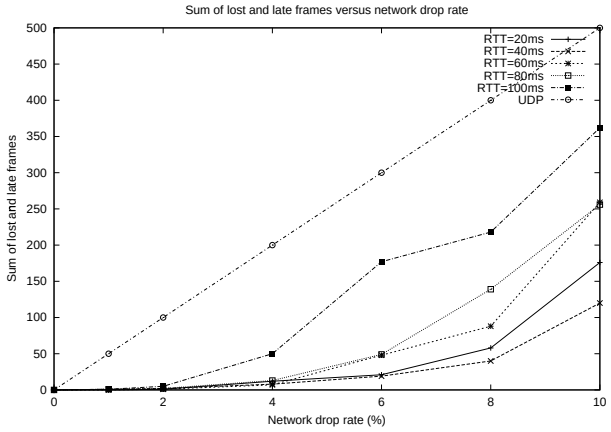


Figure 4.13: *for RTM, sum of the numbers of lost frames and delayed frames vs. network drop rates with playback delay = 250 ms and bursty packet loss.*

TCP-RTM in real environments, we also conducted public Internet experiments.

A series of experiments were conducted across the North America continent between a Solaris machine in Atlanta near the east coast and a Linux machine in Stanford on the west coast. The two machines were connected by the public Internet through 16 hops, and the round-trip time between them varied normally between 58ms and 70ms but sometimes went up over 100ms or even 1000ms briefly. The experiments were conducted during the daytime between 8am and 6pm at a 2-hour interval on a business day to expose TCP-RTM to realistic Internet traffic.

Audio-like data were sent by the Solaris machine to the Linux machine. Fig. 4.20 shows that the application level perceived jitter were very low all day. The only exception was that at 10:00am there were several spikes in lateness, which were probably caused by transient congestion and increased long queuing delay, which were out of the control of TCP at the endpoints. [10] In addition, the application-level perceived loss rate was less than 0.02%. Another series of tests were done using video-like data at 300 kbps at odd-numbered hours. The results were similar, except that there were no spikes as seen in Fig. 4.20(b).

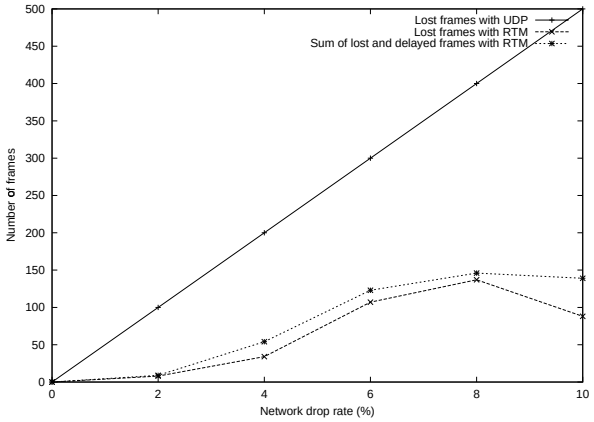


Figure 4.14: *Application-level perceived packet loss between the Solaris machine and the Linux machine across North America continent, with artificially introduced packet loss for stress-test.*

To further stress-test TCP-RTM, we introduced artificial packet loss into the communication between these two hosts. Fig. 4.14 shows that the result was also good for the loss rates ranging from 2% to 10%.

Another series of experiments were conducted between two Linux machines connected by LAN and commercial ADSL, whose bandwidth was low. [11] The RTT between them varied between 30ms to 40ms. The results on jitter and application level loss rate were also good and omitted due to space limitation.

More public Internet experiments under diverse conditions should be performed to further evaluate TCP-RTM. However, we think the Atlanta experiment was representative of a long distance call, and the DSL experiment was representative of a local, but off-campus

[10] Note that UDP-based approach could not avoid such queuing delay either.

[11] Down-stream bandwidth: 384 kbps. Up-stream bandwidth: 128kbps. Experiment data were sent up-stream.

call.

Note that the data shown so far were obtained by using a relatively short playback delay buffer of 250ms to accommodate interactive audio applications. For unidirectional non-interactive applications, such as video streaming, a much larger playback delay buffer (for example one to five seconds [18]) can be used, which gives TCP much longer time to recover lost packets and hides much more network jitter, thus much better performance can be achieved. We performed experiments that used a playback delay of 2 seconds; the results were indeed much better.

4.3.3 User-level Performance Evaluation

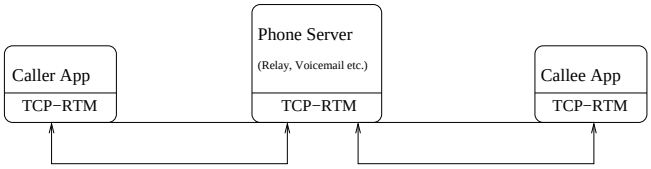


Figure 4.15: *Phone Server Model based on TCP-RTM.*

To evaluate RTM's performance in terms of user-level quality of service, in addition to transport-level loss rate and jitter, we built a simple telephone system on top of TCP-RTM. In the system, both the caller and the callee used the modified Robust Audio Tool [4] running on a PC to communicate. RAT was modified to use TCP-RTM instead of UDP to transmit voice data. A series of experiments were performed over a wide range of RTTs and loss rates and the performance results were very encouraging. We only present some representative data here. We used a sampling frequency of both 8kHz and 16kHz, with a packet interval of 20ms and 10ms respectively, and a data rate of 128kbps and 256kbps respectively.

To evaluate the audio quality, we used Mean Opinion Score (MOS), which is an ITU standard subjective quality measure [94]. MOS is given on a scale of 1 to 5. A MOS rating above 4.0 matches the toll quality in PSTN. A rating between 3.6 and 4.0 means the quality is acceptable for most users. A rating between 3.1 and 3.6 means the quality is not acceptable for most users.

When the RTT was at 70ms and the (one-way) network loss rate ranged from 1% to

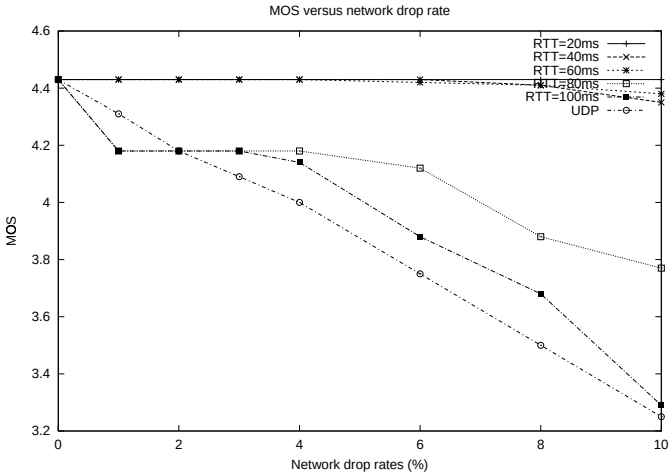


Figure 4.16: *Expected user level quality measurements in terms of MOS.*

5%, the subjective audio playback quality at the receiver had a MOS value of 4.3 (very satisfied), in the sense that the playback was continuous without noticeable crackle. In contrast, using UDP, the MOS value was at 3.88 when the loss rate was 5%.

In addition, based on the simulation data presented in subsection 4.3.2, we derive the expected MOS values (figure 4.16) for a range of loss rate and RTTs by using the mapping from loss rate and delay to MOS values presented in [9]. These MOS values show that using RTM gives significantly better user-level quality in most situations, taking into account the minor MOS degradation caused by the playback delay.

These results showed that in addition to reducing transport level loss rate and jitter, TCP-RTM indeed provides good performance in terms of user-level quality of service over lossy networks, which is the ultimate goal of a transport protocol. In addition, the modification to RAT to switch to TCP-RTM from UDP was simple and minimal, requiring fewer than 100 lines of code, which demonstrated TCP-RTM's ease of deployment.

In summary, TCP-RTM allows TCP to recover lost packets transparently to the application given an adequate playback buffer for the round-trip time without increasing the application-layer jitter and without the application requiring coding beyond that required in the simple UDP-based approach, given such applications conventionally use a playback buffer as well.

4.4 Congestion Control for the Real-Time Mode

This section studies the congestion control behavior of TCP-RTM and introduces the Selective-NACK option.

4.4.1 Problem: Lack of Congestion Control for Real-Time Applications

As mentioned earlier in Chapter 3, most real-time applications today do not perform congestion-control. This is very harmful for the Internet, especially when the percentage of real-time applications keep growing. When using TCP-RTM, however, real-time applications automatically get proven congestion control functionalities from TCP.

There are two aspects to congestion behavior with TCP-RTM: 1. How does TCP-RTM co-exist with conventional TCP? 2. How does a TCP-RTM application deal with the TCP congestion back-off behavior? Let's consider these issues in turn.

4.4.2 TCP-RTM versus TCP

TCP-RTM as described responds to congestion in a similar fashion as conventional TCP. It performs slow start, congestion back-off and congestion avoidance as regular TCP. However, it has potentially more aggressive behavior under severe congestion condition. When the application receiver thread reads over a missing segment, it generates a TCP ACK that effectively disguises the loss from the sender. Thus, compared to conventional TCP, the TCP-RTM sender may not back off as much.

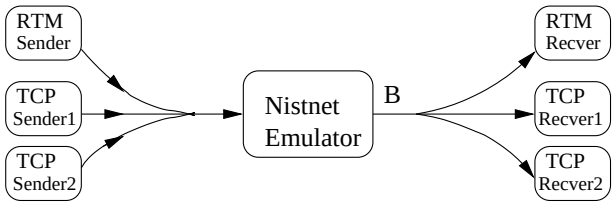


Figure 4.17: *Simulation topology to evaluate how RTM sender competes with conventional TCP. The RTM sender sends audio-like data, while the bottom two TCP senders send bulk data in a greedy fashion.*

To evaluate this effect, we did two simulations. In the first one, a TCP-RTM connection competed with two conventional TCP connections over a link with limited bandwidth B, as illustrated in Fig. 4.17. This link was controlled by Nistnet using the DRD algorithm [47],

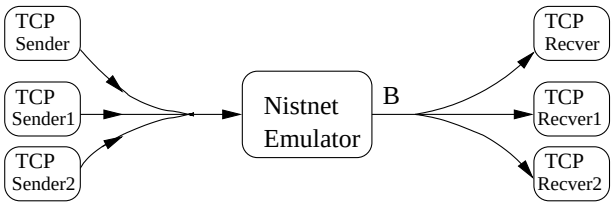


Figure 4.18: *Simulation topology to observe how unmodified TCP sender competes with conventional TCP. The TCP sender on the top sends audio-like data, while the bottom two TCP senders send bulk data in a greedy fashion.*

a variant of RED, with $DRD\_MIN = 5, DRD\_MAX = 15$ [12]. In this simulation, the TCP-RTM connection sent audio-like streaming data at a constant rate of 64kbps (including all the TCP/IP header overhead). The two TCP connections have unlimited amount of data to transmit and do so as fast as allowed. The link bandwidth B was varied from 320kbps to 1.28Mbps, which resembled the bandwidths of current commercial DSL lines. Since the two conventional TCP connections sent data as fast as possible, congestion would result at the shared link and congestion-induced packet loss would be caused. We observed the average congestion window sizes [13] (abbreviated as *cwin*) of the TCP-RTM to evaluate its aggressiveness. The *cwin* size (along with the receiver's advertised window size) limits the rate a TCP sender can transmit data, therefore it is a good indicator of the aggressiveness of the TCP sender.

In the second simulation, as illustrated in Fig. 4.18 we replaced the TCP-RTM connection with an unmodified regular TCP connection, which transmitted the same streaming data at the same rate, competing with the same two other TCP connections. We again observed the average *cwin* sizes of the unmodified TCP connection, and compared them with those of TCP-RTM observed in the first simulation, as shown in Table 4.1. Note that the two simulations used exactly the same parameters for both the network link and the transmitters, except for the swap of the transport protocol from TCP-RTM to regular TCP to transmit the audio-like streaming data.

Table 4.1 shows the average *cwin* sizes for each connection and the percentage difference for each data rate. This table indicates that when the available bandwidth is relatively

[12] When the number of packet queued is less than DRD_MIN, no packets are dropped. Once the number of packet queued reaches DRD_MIN, 10% of the incoming packets are randomly dropped. This percentage ramps up with queue length, reaching 95% when DRD_MAX is reached.

[13] We measured the cwin size in segments instead of bytes.

	TCP-RTM	TCP	Difference
B=1280kbps	3.93	3.91	0.5%
B= 960kbps	3.95	3.78	4.5%
B= 640kbps	4.16	3.87	7.5%
B= 320kbps	4.31	3.78	14.0%

Table 4.1: *Comparison of average cwin sizes of TCP-RTM and TCP. B is the bandwidth of the bottleneck link.*

high, e.g when $B = 1280kbps$, there is not much difference between TCP-RTM and TCP's congestion behavior. However, the TCP-RTM as described is more aggressive than conventional TCP under serious congestion condition, with the difference increasing as the available bandwidth decreases further below the offered load. The reason that TCP-RTM's average *cwin* size increases when the available bandwidth decreases is that the receiver TCP-RTM skips more missing segments and "disguises" the sender more often. When the receiver sends back an ACK that acknowledges packets past the missing packet when it skips over a "hole", the sender is not aware of the missing packet and does not scale back its *cwin*, which an un-modified TCP would do.

TCP-RTM as described has similar congestion control behavior to normal TCP but may claim more bandwidth when the congestion condition is severe, so we have further extended TCP-RTM to better respond to network congestion using a very slight modification to SACK [82] handling, which we refer to as the SNACK (Selective-NACK) extension. As [8] points out, the SACK option has received wide acceptance, so we expect the deployment of SNACK will be straightforward.

4.4.3 SNACK Extension

TCP-RTM includes the selective-negative-ACK (SNACK) extension to respond to network congestion. In particular, when the receiver reads out-of-order data, it piggybacks a SACK option in the ACK it sends back to the sender. The SACK option indicates (as usual) the out-of-order data that the receiver has received. However, the ACK acknowledges beyond this point, allowing the sender to infer that packets have been lost yet still advance the acknowledgment window so it does not attempt to retransmit these packets.

For example, consider, as illustrated in Figure 4.19, the case of packet 1 having been received containing up to sequence number A with packet 2 containing up to sequence number B dropped and packet 3 containing up to sequence number C received.

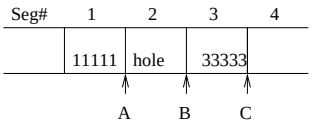


Figure 4.19: *Illustration of SNACK.*

	TCP-RTM	TCP	Difference
B=1280kbps	3.85	3.91	-1.5%
B= 960kbps	3.77	3.78	0%
B= 640kbps	3.76	3.87	-3%
B= 320kbps	3.75	3.78	0%

Table 4.2: *Comparison of average cwin sizes of TCP and TCP-RTM with SNACK.*

If the receiver reads over the hole between A and B, it sends an ACK with acknowledgment sequence number $ack\_seq\_num = C$, piggybacking a SACK block of B-to-C in the same packet. The sender can immediately infer that the packet(s) from A to B are lost and cut its congestion window size by half as usual. Thus, as far as the congestion control is concerned, this modification causes the sender to do exactly what it would do if it received 3 duplicate ACKs, only without retransmitting.

This extension requires minimal modification to TCP's SACK code. In particular, on receiving of a SACK, if the *ack_seq_num* is greater than the beginning of the SACK-specified block, it reduces its `cwin` by half, if no such reduction has been done due to these lost packets, but does not retransmit the missing data.

This behavior is consistent with TCP acknowledgment and SACK semantics. The *ack_seq_num* fundamentally indicates to the sender the buffered data it is free to release because it is no longer needed for retransmission, exactly the semantics required here. Further, the SACK option communicates the out-of-order data received at the receiver, again the semantics we desired.

This situation of the *ack_seq_num* being greater than the beginning of the SACK-ed block should only arise with TCP-RTM. Moreover, our proposed behavior here seems suitable to deal with a misbehaving or malfunctioning TCP receiver that is simply sending an erroneous SACK. Thus, it seems feasible to modify TCP to behave in this fashion in all cases so no explicit setting or mode is required on the sender end with TCP-RTM applications.

Another simulation was done using TCP-RTM along with the SNACK extension; see Table 4.2. It indicates that TCP-RTM using the SNACK extension competes fairly with unmodified TCP, without being more aggressive.

	Competing with RTM	Competing with TCP	Difference
B=1280kbps	4.50	4.58	-1.7%
B= 960kbps	4.60	4.57	0.7%
B= 640kbps	4.48	4.44	0.9%
B= 320kbps	4.41	4.35	1.3%

Table 4.3: *Comparison of average cwin sizes of conventional TCP flows when competing with RTM audio-like traffic (with SNACK) and TCP audio-like traffic respectively.*

Table 4.3 compares the average *cwin* sizes of the two bulk data TCP connections when competing with the RTM audio-like connection (with SNACK) and the TCP audio-like connection respectively. This table shows that from the perspective of the conventional TCP's, its *cwin* size cannot distinguish whether a RTM connection or a TCP connection is competing.

Simulation results using video-like data stream with data rate at about 400 kbps gave similar result as the audio-like data stream.

Thus, TCP-RTM appears to coexist fairly with conventional TCP. Also, analytically, due to the minimal extension to TCP, the only behavior of RTM that may affect congestion control is the skip-over-hole operation. Because the SNACK extension makes sure that the sender accounts for such packet loss and backs off properly, TCP-RTM's congestion control behavior should be the same as regular TCP. Note that this property of TCP-RTM offers a big advantage over using UDP, which requires the application developers to reinvent the congestion control mechanisms to be TCP-friendly.

With this modification, TCP-RTM reduces its congestion window and therefore its data rate in response to network congestion. Thus, the sender is not allowed to continue sending at the same rate and must adapt, as described next.

4.4.4 Responding to Congestion in a TCP-RTM Application

The TCP-RTM sender can detect network congestion by checking the TCP buffering of its output data periodically or checking the TCP-RTM loss-counter introduced in 4.1.2. If large data is queued or the loss-counter has high relative value, the sender can infer

that the network (or the local host) is congested and react with some application-specific solution. For example, with layered media encoding, the source can drop a layer to a more basic encoding, thereby reducing the data rate and the network congestion. Alternatively, the source can switch to a lower-data rate encoding of the application-level data. Possible responding schemes have been extensively studied as in [57, 56, 84, 105, 113, 117, 127], and are beyond the scope of this paper.

In summary, with the SNACK behavior described above, a TCP-RTM application coexists fairly with conventional TCP, responding to network congestion and reducing its data rate as necessary. More explicitly, TCP-RTM fully conforms to the additive-increase/multiplicative-decrease (AIMD) principles [26, 59] of conventional TCP. Further, it does so without duplicating the TCP congestion control mechanism using some application-specific means, which a UDP-based application would require. Thus, a "good citizen" real-time application is substantially simpler to build using TCP-RTM than UDP.

4.5 Summary

TCP-RTM provides a new mode for TCP that allows it to support a range of real-time applications with performance superior to using a basic datagram service such as UDP. TCP-RTM makes all the benefits of TCP available to these real-time applications while allowing application developers to use one transport protocol, not two or several. It also ensures real-time applications are responsive to network congestion in a way compatible with established TCP dynamics. In addition, TCP-RTM requires modification to less than 1 percent of the TCP code, a trivial software cost compared to supporting a whole separate transport protocol. This modification is interoperating with existing TCP implementations, making incremental deployment straight-forward.

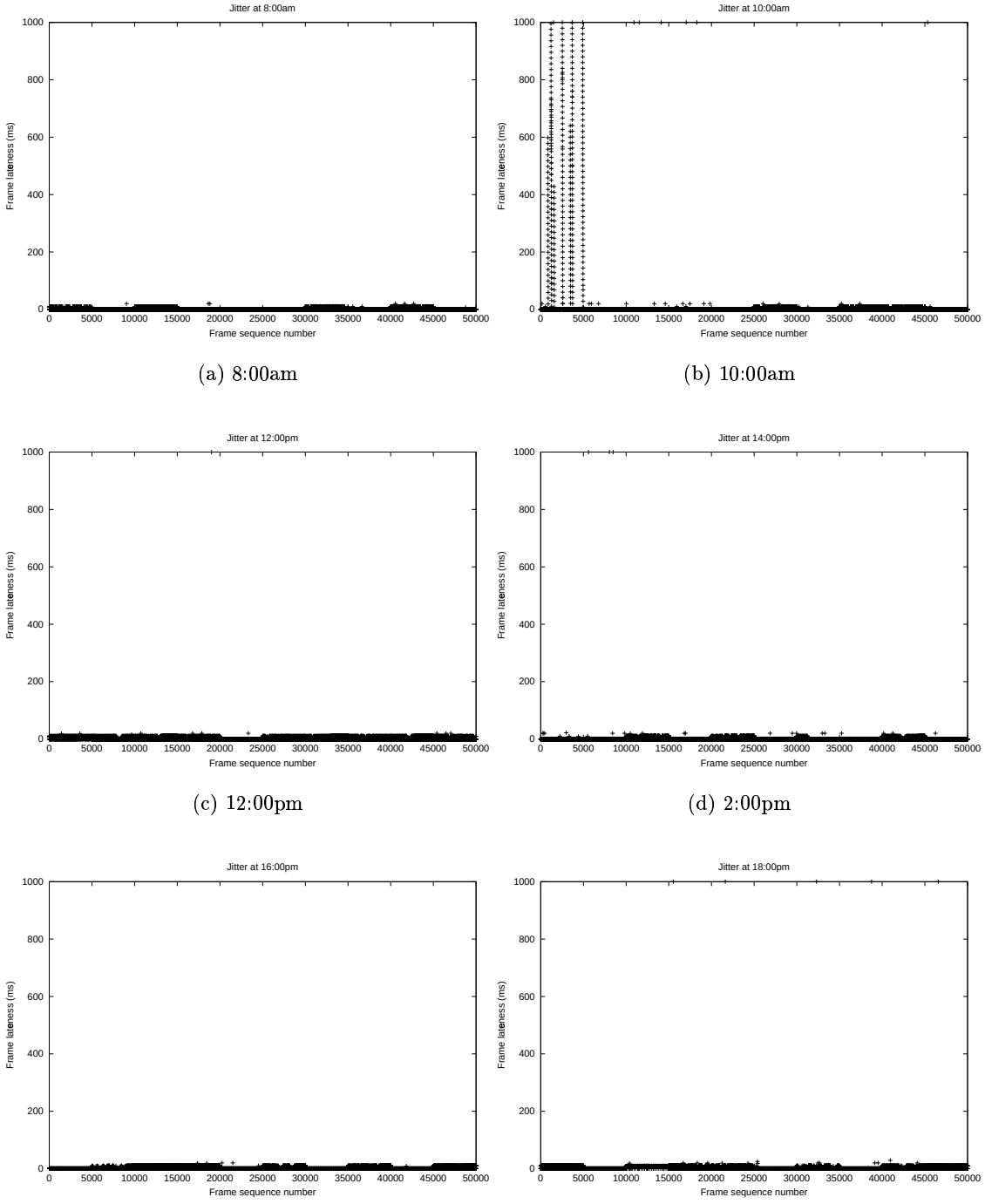
Several aspects of our work are of note. First, we showed how structuring an application to use the TCP buffering for the playback buffer together with fast retransmit allows retransmission on packet loss without introducing jitter at normal levels of network packet loss. The resulting recovery of lost packets is important with applications such as video and audio streaming that are made loss sensitive by their use of compression techniques. We argued that is far more effective to retransmit in this fashion than to send the less loss-sensitive but far larger raw data.

Second, we showed that the RTM receiver behavior of stepping over missing packets

and the sender behavior of advancing the window independent of acknowledgments ensures a real-time application seldom stalls by excessive levels of packet drop.

Finally, we showed that a minor modification to the SACK code causes TCP to provide congestion response that is compatible to regular TCP, whether in RTM mode or not.

Overall, we conclude that it is a better engineering decision to extend TCP to support real-time than to require real-time applications to use a separate specialized transport protocol, given the many benefits of using TCP and not having to support and use a separate protocol. We see TCP-RTM as a promising definition of such an extension and hope to see it incorporated in the standard implementation of TCP.



(a) 8:00am

(b) 10:00am

(c) 12:00pm

(d) 2:00pm

(e) 4:00pm

(f) 6:00pm

Figure 4.20: *Cross-continental Experiment results. At each time, 50000 frames were transmitted. Each figure shows the lateness in ms for each frame.*

Chapter 5

Framing Support

TCP was originally designed as a byte-stream protocol, preserving no record boundaries. This is sufficient for applications like telnet or bulk file transfer. However, many applications are record-based, such as database applications, multimedia applications, network games and distributed file systems, where the data to be transmitted has some intrinsic record-structure, such as database data objects, media samples, gamer positions/actions, RPC requests/replies, etc., and it is desirable to use a transport protocol that can preserve such record-structure during transmission, so that the receiver can easily and efficiently recover [1] the records.

This chapter discusses the framing support in a reliable transport protocol. Such framing support preserves higher level frame boundaries, thus it simplifies frame recovery and improves performance. In the next section, the problems with the lack of framing support are explained, then Section 5.2 presents a simple solution to add framing support to TCP to solve these problems, and Section 5.3 presents the performance evaluation results. Finally, Section 5.4 summarizes this chapter.

5.1 Framing Problems

Record-based applications have data represented in records or frames[2]. When such data is transmitted, the receiver needs to recover each frame before it can be processed. A

[1] The term "recover" in this chapter refers to the operation to determine the boundaries of a frame and identify the data that belongs to the frame. It is not used to mean retrieving data from lost packets.

[2] Note that for generality, in this chapter we only discuss variable sized records or frames; fixed sized frames are easier to process.

byte-stream transmission model obscures the frame boundaries, and makes it difficult and inefficient for the receiver to recognize the frames and recover them. In this section, we explain the framing problems in more detail, including the extra buffering and gathering problem, the problem with network storage applications and the problem with semi-reliable applications.

5.1.1 Extra Buffering & Gathering

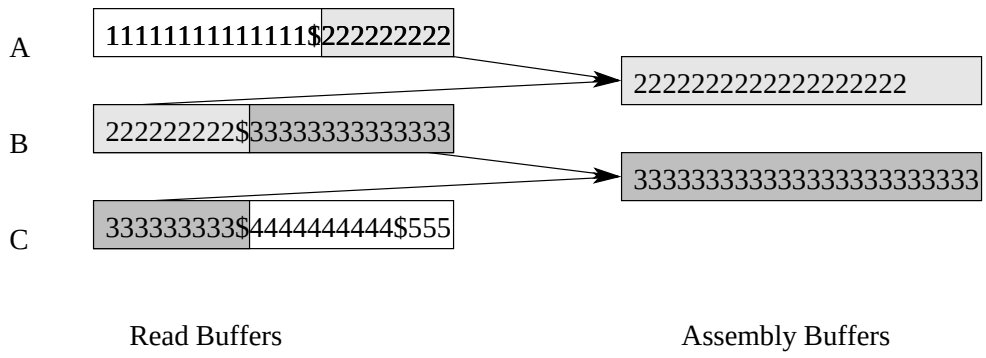


Figure 5.1: *Illustration of the extra buffering and gathering problem (assuming the '$' sign represents the frame delimiter).*

Because a byte-stream transport protocol obscures the frame boundary, the receiver of a record-based application usually needs to perform extra data buffering and gathering to recover each individual frame before it can be processed. One common way of doing this is to use several read buffers and each time read fixed number of bytes from the transport layer into one read buffer, search for frame boundaries in the read buffers to recognize each individual frame (for example, using a special frame delimiter to indicate frame boundaries and searching for it), then copy the data that belongs to one frame from the read buffers to another assembly buffer, and finally the receiver application can use the data in the frame reconstructed in the assembly buffer.

Figure 5.1 shows an example of this problem. In this example, the receiver application reads data from the transport layer into the read buffers. Read buffer A contains data that belongs to `frame#1` and `frame#2` (for ease of illustration, assuming all the `'1'` characters belong to `frame#1` and all the `'2'` characters belong to `frame#2` and so on), read buffer

B contains data for `frame#2` and `frame#3`, and read-buffer C contains data for `frame#3`, `frame#4` and `frame#5`. Because the data for many frames is not contiguous and is spread out in multiple read buffers, to use the individual frames separately, the receiver must search for frame boundaries in the read buffers, and then copy the data that belongs to `frame#2` to the first assembly-buffer, and copy the data that belongs to `frame#3` to the second assembly-buffer. Only after such reconstruction of `frame#2` and `frame#3` in the assembly buffer can the receiver properly use them.

This process is very memory intensive. First, it requires extra data buffers (read buffers and assembly buffers), costing more memory. In addition, it needs extra copying operations, which slows down the receiving process. Furthermore, the logic to recognize and recover the frames is quite complicated.

One common way to reduce the extra memory used is to use just one data buffer (assuming some maximum frame size), and shift data in place, we call this the *wrap-around* approach. Using this approach, the receiver uses only one read-buffer. Each time, it reads enough data to fill the read-buffer, it searches for the frame boundary and consumes the frame. Next it shifts the data at the tail of the buffer to the head of the buffer, and repeats this read-search-shift cycle.

```
Loop
  Read from transport layer,
  Scan and find frame boundary,
  Process the frame,
  Shift-forward (data movement).
end
```

Figure 5.2 shows an example of this method. At first, the receiver reads enough data to fill the entire read buffer. Then it searches for the end of `frame#1` starting from the beginning of the read buffer and uses `frame#1`. At step 2, it shifts the remaining data in the read buffer after `frame#1` from the tail of the read buffer to the beginning of the buffer, next it reads more data to fill the buffer, and then searches for the end of `frame#2` and uses `frame#2`. At step 3, it repeats this process by shifting the remaining data after `frame#2` from the tail of the read buffer to the beginning, reading more data to fill the buffer and then searching for the end of `frame#3` and using it.

Although this method avoids using extra memory, it still needs to do extra data copying

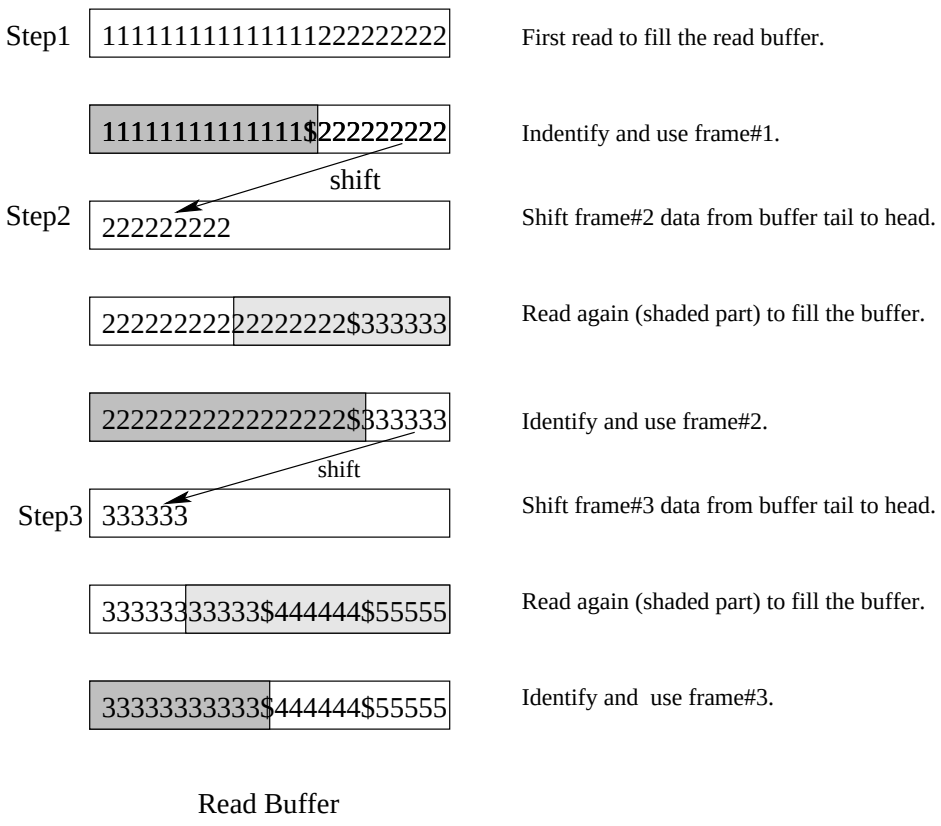


Figure 5.2: *Illustration of the wrap-around approach (assuming the '$' sign represents the frame delimiter).*

when it shifts data from the tail of the read buffer to the head, in addition to searching for frame boundaries.

For both methods, we assume the sender application uses a marker to delimit the frames, and the receiver searches for this marker to find the frame boundaries. Such searching requires the receiver to scan every byte of data in the read buffer. If this scanning is implemented in addition to any other application-level data scanning or De-Marshalling operations, if there is any, it further increases the load on the memory and CPU.

5.1.2 Byte-Counting

One approach to avoid the extra buffering and gathering described in the previous section is to use a technique called *byte-counting*. Basically, the sender writes the length of the application level frame to the network first, then writes the actual frame data (alternatively, these two writes could be combined into one write to reduce the number of system calls). At the receiver side, the receiver first reads the length of the frame, then reads the frame data using the frame length. For example, suppose the data frame length is 100 bytes. The sender first writes the frame length 100 in two bytes to the network, then writes the 100 bytes of frame data to the network. The receiver first reads the two bytes of the length field and gets the frame length 100, then it reads 100 bytes to get all the data for the frame.

In addition to adding a slight data overhead with the length field, which is not desirable especially for applications using small frame sizes, such as voice applications due to latency penalty, the receiver has to issue at least two read system calls to obtain each frame, and this increases the CPU load on the receiver and also reduces the transmission performance. Furthermore, this method only works with fully reliable transport. With semi-reliable transport or best-effort transport, once one packet containing a length field is lost, none of the subsequent frames can be recovered as the consequence of the chain-reaction effect.

We compare the performance of byte-counting with that with framing support in Section 5.3.3.

5.1.3 Problem with RDMA Direct Data Placement

While processor speed and network bandwidth have been increasing tremendously, computer memory has not kept pace with that of processors and network bandwidth. Therefore, for some applications that operate on high speed networks, such as network storage applications, memory speed on the endhost limits data throughput [99, 32, 106].

To reduce the amount of data copying on the receiver improves data throughput. A technique called RDMA is being developed to have the hardware place a data packet's payload directly into the memory buffer of the receiving application, which is also referred to as *Direct Data Placement*. A header called Region ID (RID) is placed at the beginning of the payload in the application level frame, based on which, the receiving network hardware is able to determine where in the application memory to place the payload data. (For details of RDMA, please refer to [110, 106].) Framing support is required for the receiver to find

the RID header of each frame and use the information in the header to determine exactly where to place the packet's payload data.

Without efficient framing support, the performance of the RDMA receiver is greatly harmed. For data delivered in order, RDMA may not be able to directly place the data in each incoming segment immediately, because one application-level frame may be spread across several TCP segments and it may suffer the problems described in Section 5.1.1. The performance is particularly hurt after one or more segments are lost in the network. Because the receiver cannot find header information for out-of-order segments received after a missing segment, it cannot determine where in the application memory buffer to place the data.

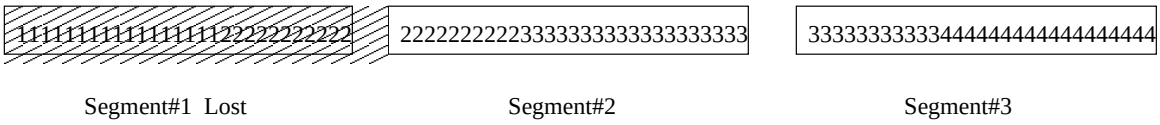


Figure 5.3: *Illustration of the framing problem for RDMA.*

Figure 5.3 shows an example of the problem with out of order segments. Four data frames are packetized in three TCP segments. Assume `segment#1` is temporarily lost, and `segment#2` and `segment#3` are received out of order. Because `segment#2` contains only the rear part of `frame#2`, the receiver cannot find its header. Although `segment#2` contains the header of `frame#3`, it is difficult to find its beginning.

To handle out-of-order segments, the receiver may have the following alternatives:

1. Save the out-of-order segments on the NIC, which requires extra RAM and increases the cost of the NIC. In addition, after the missing segment is retransmitted, there will be a large burst of data movement to transmit all the segments saved on the NIC to the application memory. Because the memory speed is slower than the network speed, such bursty data movement slows down the overall data throughput.

2. Save the out-of-order segments in temporary kernel memory, requiring extra kernel memory and one extra data copy, and similarly, after the missing segment is retransmitted, there is also a big bursty data movement.

3. Drop the out-of-order segments, which is inefficient, because it requires the retransmission of all the out-of-order segments, and it tremendously shrinks the congestion window at the sender.

Obviously, none of these alternatives is particularly attractive.

5.1.4 Problem with the Semi-reliable Mode

For real time multimedia applications that use the proposed semi-reliable mode of TCP, TCP-RTM, there are some special framing problems.

First, when a missing segment is skipped, it is difficult to find the header of the next frame; it is slow to scan the data and examine every byte to find the frame boundary. This is similar to the problem of finding `frame#3`'s header in `segment#2` in figure 5.3, except that in the RTM case, `segment#1` will not be retransmitted.

Second, partial frames are usually useless, so certain packetization may cause higher application-level loss rate than the actual network level loss rate.

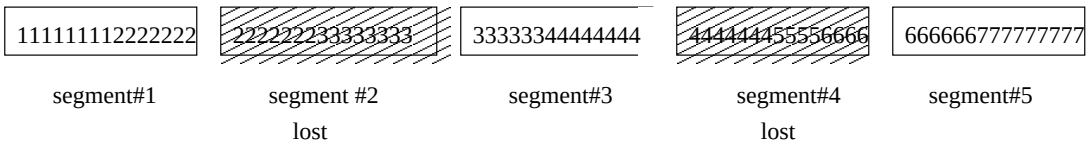


Figure 5.4: *The loss of segment#2 causes the application to lose both frame#2 and frame#3, and the loss of segment#4 causes the loss of both frame#4, frame#5 and frame#6, causing the application-level loss rate to be more than twice as high as the network level loss rate.*

For example, in figure 5.4, the loss of `segment#2` causes the application to lose both `frame#2` and `frame#3`, and the loss of `segment#4` costs the application to lose both `frame#4`, `frame#5` and `frame#6`. In this case, the application-level loss rate is more than twice as high as the actual network level loss rate.

5.1.5 Summary

In summary, the lack of framing support causes significant overhead for the applications to recover record boundaries, including overhead in both space and speed. Next we describe a simple solution to provide framing support in the reliable transport layer.

5.2 Adding Framing Support to TCP

As mentioned earlier, the way the current TCP byte-stream model works is that the transport layer receives data from the sender application, treating it as an unstructured byte stream, and performs segmentation to place data into segments and transmits them. This model obscures the application frame boundaries, and makes frame recovery complicated and inefficient. To make frame recovery simple and efficient, we apply the Application Level Framing (ALF) principle and provide framing support in the transport layer.

ALF was proposed by Clark and Tennenhouse [29] as an architectural principle for designing networking protocols. ALF advocates that "the application should break data into suitable aggregates, and the lower level should preserve these frame boundaries as they process the data." They call these aggregates Application Data Units, or ADUs, and "ADUs will then take the place of the packet as the unit of manipulation."

We let the application do the segmentation, and have the lower-levels preserve the higher-level frame boundaries by aligning the application frame boundaries with transport layer segment boundaries, so that application frames can be easily and efficiently recovered.

With our framing support, if the application's frame size is larger than the PMTU (Path MTU), we require the application to use a higher level framing protocol and break the application-level frame into a number of ADUs that fit the PMTU[3]. Consequently, the head of the first ADU aligns with the beginning of the application-level frame, and the end of the last ADU aligns with the end of the application-level frame. During transmission, the application writes one ADU to the transport layer at a time, and the transport layer aligns the ADUs with the transport segments.

If the application's frame size is smaller than the PMTU, for example, frames of voice applications are usually small, then each application-level frame corresponds to one ADU, and a higher-level framing protocol can be optimized out.

We add a new TCP framing socket option which makes each TCP segment contain only one ADU. Effectively, we ensure that the application level frames always align with the transport level segments. Simple changes are made in the transport layer at the sender-side and the receiver-side, which are explained as follows.

[3]Obviously, the last ADU may be smaller than the PMTU.

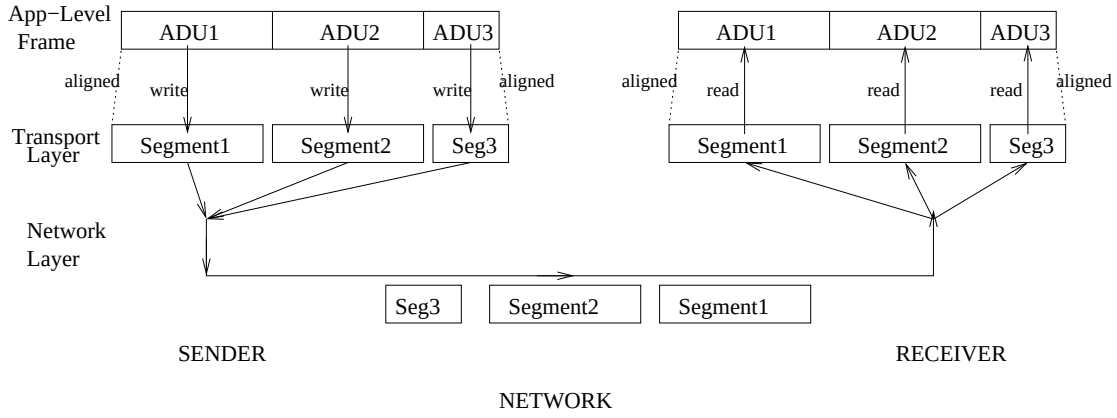


Figure 5.5: *Illustration of the TCP framing support. On the left, the Sender's application-level-frame is broken into 3 ADUs, each of which is written to the transport layer separately, and each is carried by a separate transport layer segment. When these segments reach the Receiver on the right, each is returned to the receiving application in one read. Because the higher level frame header is at the beginning of ADU1, the receiving application can immediately recover the higher level frame header and reassemble the higher level application frame easily.*

5.2.1 Sender-Side Framing Changes in TCP

At the sender side, the transport layer treats the data in each application write call as an independent application data unit (ADU), and it encapsulates each ADU in one TCP segment. At transmission, each application-level frame is transmitted in one TCP segment. At retransmission after packet loss, no repacketization is performed which may packetize different ADUs into the same segment. The result is that the ADU boundaries always align with the transport level segment boundaries and so are the application-level frame boundaries. See Figure 5.5 for a visual illustration.

5.2.2 Receiver-Side Framing Changes in TCP

At the receiver side, for each application read request, the transport level returns all the data in one and only one TCP segment[4]. Because each TCP segment contains just one ADU, each read request obtains one complete ADU. And because the application level frame

[4]If the data buffer provided by the application receiver is not big enough for the entire TCP segment segment, an error would be returned.

headers align with ADU boundaries, the application level frame headers can be trivially recovered, without the need to do complicated searching and reassembly.

5.2.3 TCP-Framing Rationales

Despite its simplicity, this new "framing mode" represents an important paradigm shift. Since its origin, TCP has always been a byte stream protocol, preserving no record boundaries. However, the new "framing mode" adds a new mode to TCP that changes this byte-stream model, and significantly simplifies the applications and improves their performance.

Conceptually, this TCP framing solution complies with the end-to-end design principle [109], which suggests that "functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level". With traditional byte-stream model, functions performed at the lower transport layer to concatenate application level frames (such as the repacketization operation at retransmission) and the resulting obscuring effect on the frame boundaries degrades the performance of upper layer record-based applications. So the byte-stream model is not only "of little value" by doing unnecessary operations, but actually hurts application's performance. The new framing solution rectifies this violation of the end-to-end principle.

In addition, the TCP framing solution also complies with the layering principle [31] for network design: data coming out of layer `n` at the receiver should be the same as the data going into layer `n` at the sender. In our case, layer `n` is the transport layer. Basically, the data that comes out of the transport layer at the receiver should be exactly the same as the data that is sent into the transport layer at the sender, with no properties of the data changed. With the byte-stream model, the data coming out of the transport layer at the receiver loses the record delimitation information, which is an important property of the data being transmitted.

5.3 Implementation and Performance Evaluation Results

5.3.1 Trivial Extension

The framing solution is implemented in the Linux kernel. The implementation is very straightforward, requiring less than 50 lines of code change. Compared to other approaches

to providing framing support, such as designing a new transport protocol, for example SCTP or RDP, or adding another layer on top of TCP, our TCP framing approach offers clear advantage. It is easier to implement, and because it is based on TCP, the transport protocol with the largest install base, it is much easier to deploy.

5.3.2 Higher Performance for Record-based Applications

Framing support improves the performance for record-based applications to recover frames, because it frees the applications from scanning data to find record boundaries and enables them to avoid extra buffering and copying.

To determine the performance benefit, we performed simulations to compare the throughput to recover frame boundaries and reassemble the frames with and without the framing support. In the case without framing support, the application receiver used the wrap-around approach described in Section 5.1.1 to recover each frame. In the case with framing support, the receiver recovered each frame directly without the need to do extra data copying and data scanning to search for frame boundaries. In each case, we measured the throughput to process 400MB of data, with frame size ranging from 600 bytes to 1400 bytes. The measurement results are presented in Figure 5.6 and Table 5.1.

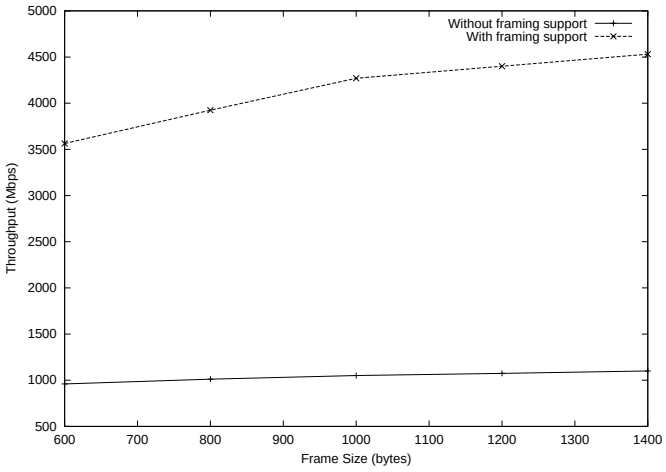


Figure 5.6: *Comparison of frame recovery throughput.*

Figure 5.6 and Table 5.1 show that with framing support, the frame recovery performance can be four times as good as the case without framing support. The larger the frame

FrameSize(bytes)	No Framing (Mbps)	Framing (Mbps)	Framing/NoFraming Ratio
600	960	3564	3.70
800	1011	3925	3.86
1000	1051	4270	4.06
1200	1074	4401	4.10
1400	1100	4531	4.12

Table 5.1: *Comparison of frame recovery throughput.*

size, the bigger the advantage the framing support provides.

The first reason for the performance difference is that without framing support, the application has to do extra data copying to move data around. Secondly, to find frame boundaries, the application needs to search for the special frame delimiters, and it has to read data byte-by-byte or word-by-word and examine each byte/word. This requires more read system calls and the large number of individual reading of each byte/word from the memory is time consuming. (See Figure 5.2 for the illustration of the extra data copying and data scanning.)

These simulations assumed the data had already arrived at the receiver machine [5]. They measured the data frame recovery throughput on the receiver machine.

We also performed data transmission experiments over a gigabit network connection between two hosts and measured their end-to-end throughput. As mentioned before, the growth of memory speed is slower than the growth of network speed, so the bottleneck is usually at the receiver. As 10 Gbps Ethernet gets wider deployment, this situation of relatively slower receiver compared to the fast network will persist. In these experiments, to demonstrate the effect of slow receiver with respect to fast network, we chose to use a relatively slow host as the receiver. We transmitted one million frames, with frame size ranging from 600 bytes to 1400 bytes, over a gigabit connection between two PCs.

Figure 5.7 and Table 5.2 show that with framing support, the overall data transmission throughput can be 8% to 52% faster than without framing support.[6]

[5]For these simulations, we used a commodity Linux PC with a 930MHz PentiumIII CPU and 512MB RAM.

[6]When the frame size is smaller, such as 600 bytes, using no framing support appears to have slightly higher throughput than using framing support (79.1Mbps vs. 75.5Mbps). This is mainly because when the frame size is small, it is possible for the sending TCP to packetize more than one frames into one TCP segment, so in total it is transmitting fewer number of segments, and also the relative header overhead is smaller. However, when the application is trying to optimize throughput, it usually should not use small frame size.

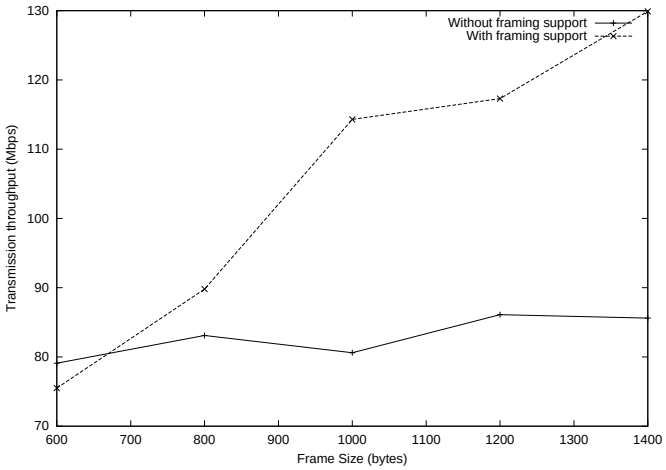


Figure 5.7: *Comparison of frame transmission throughput.*

FrameSize(bytes)	No Framing (Mbps)	Framing (Mbps)	Framing/NoFraming Ratio
600	79.1	75.5	0.95
800	83.1	89.8	1.08
1000	80.6	114.3	1.42
1200	86.1	117.3	1.36
1400	85.6	129.9	1.52

Table 5.2: *Comparison of frame transmission throughput.*

5.3.3 Better performance than Byte-Counting

We also performed data transmission experiments to compare the throughput between using framing support and byte-counting. Figure 5.8 shows that with framing support, the data throughput can be 19% to 27% higher than that with byte-counting. This is mainly because with byte-counting, more read system calls need to be performed to obtain each frame and these increase the CPU load on the receiver and slows down the process. We do notice that when the frame size is less than 850 bytes, the throughput using byte-counting can be higher than that with the framing support. This is because with framing support, it is not packetizing separate frames in one TCP segment, so in total, it needs to transmit more TCP segments. This can be easily overcome by allowing the TCP sender to packetize multiple complete application-level frames, which still maintains the boundary alignment between the ADU frames and the TCP segments. As shown in Figure 5.8, the throughput

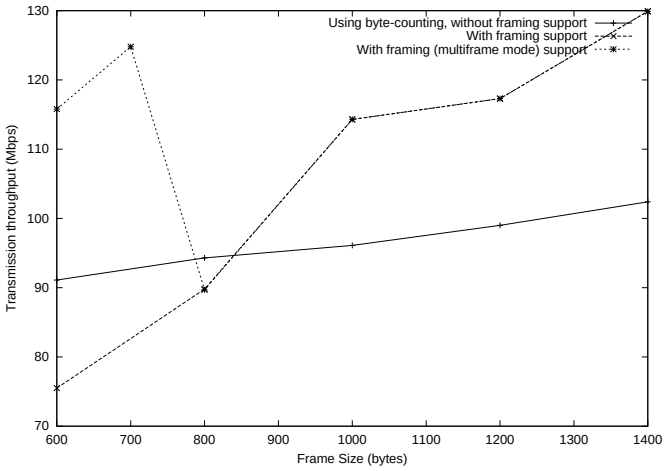


Figure 5.8: *Comparison of frame transmission throughput between using framing support and byte-counting.*

of this multi-framing mode is much better than byte-counting, even for smaller frames. [7]

5.3.4 High Performance for RDMA Applications

For RDMA applications, without framing support, all the data goes through the memory bus on the receiver host at least three times:

1. From NIC to kernel memory using DMA.

2. From kernel memory to CPU registers.

3. From CPU registers to application memory.

With framing support, all the data goes through the memory bus only once, because the data in each frame is directly placed in the application memory using the RDMA direct data placement technique. As mentioned earlier in Section 5.1.3, the receiving hardware can use the RID information in the application frame header to place the payload data directly into application memory, without going through the CPU. (Please refer to [110, 106] for details.)

[7] For frame sizes larger than 800 bytes, the performance of the multi-framing mode is the same as that of the regular (single) framing mode.

To evaluate the benefit of the framing support for RDMA applications, we have performed a number of simulations. We use a large array of data to simulate the network in the equilibrium state. The RTT, network bandwidth and the packet loss rate are varied to simulate various network conditions. Without framing support, after a packet loss, the out-of-order packets are copied to temporary memory, then after a RTT, when the lost packet is retransmitted, the RDMA receiver finds the frame boundaries and copies the data to the final memory location. With framing, after a packet loss, out-of-order packets are still directly copied to the correct final memory location, bypassing the frame boundary searching step and the extra data movement.

In the first experiment, we used the following parameters:

```
Frame size = 1KB,
RTT = 0.5ms,
Bandwidth = 2Gbps,
RTT*Bandwidth = 1Mb = 125KB = 125 frames.
```

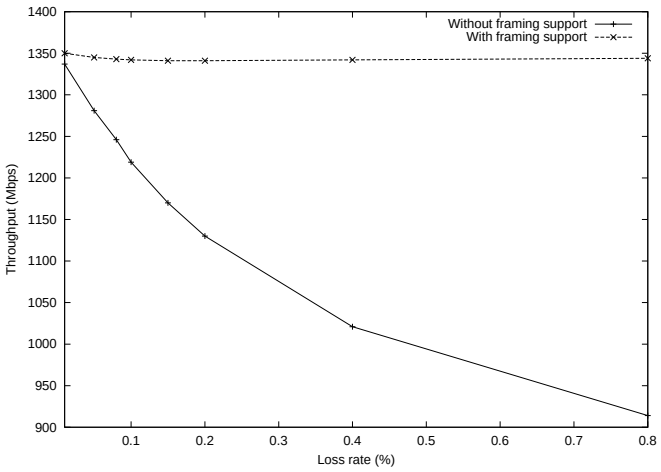


Figure 5.9: *Comparison of RDMA frame recovery throughput with packet loss. RTT*BW = 125 frames.*

Figure 5.9 shows the simulation results. With framing support, because all data (in-order or out-of-order) is directly placed into application memory buffers, throughput barely changes when the loss rate ranges from 0.01% to 0.8%. However, without framing support, the throughput significantly drops when loss rate rises to 0.8%. Figure 5.10 shows that

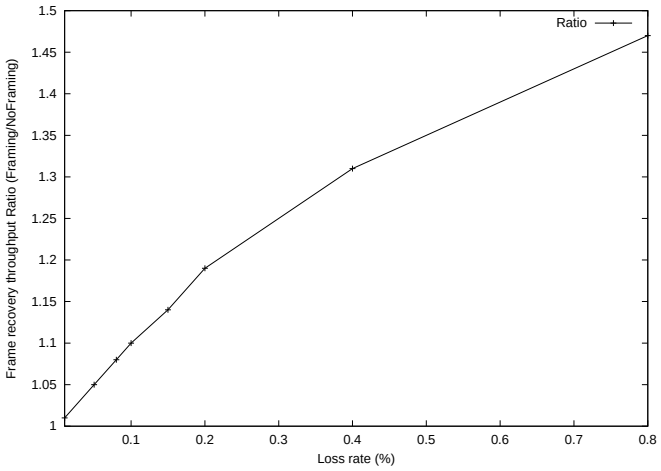


Figure 5.10: *RDMA frame recovery throughput ratio between with framing and without framing support. RTT*BW = 125 frames.*

the higher the loss rate, the bigger the benefit of the framing support. When the loss rate is 0.8%, the throughput with framing support is 47% higher than that without framing support.

In the second experiment, we used the following parameters to simulate a network ten times faster:

```
Frame size = 1KB,
RTT = 1ms,
Bandwidth = 10Gbps,
RTT*Bandwidth = 10Mb = 1250KB = 1250 frames.
```

Figure 5.11 and 5.12 show that when network bandwidth is higher, the advantage of framing support is bigger. When the loss rate is 0.08%, the throughput with framing support is more than twice as high as the throughput without framing support. The reason for this higher ratio is that when the product of RTT and bandwidth is higher, without framing support, the receiver needs to buffer more data in the case of packet loss, and when the lost packet is retransmitted, there is more data to copy to the final memory destination. Because data copying is slow, the overall throughput is hurt more.

Therefore, the larger the `RTT*Bandwidth` product, the bigger the difference the framing support makes.

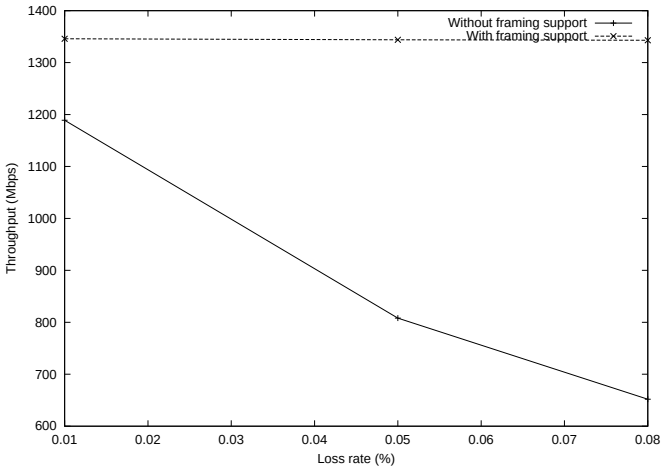


Figure 5.11: *Comparison of RDMA frame recovery throughput with packet loss. RTT*BW = 1250 frames.*

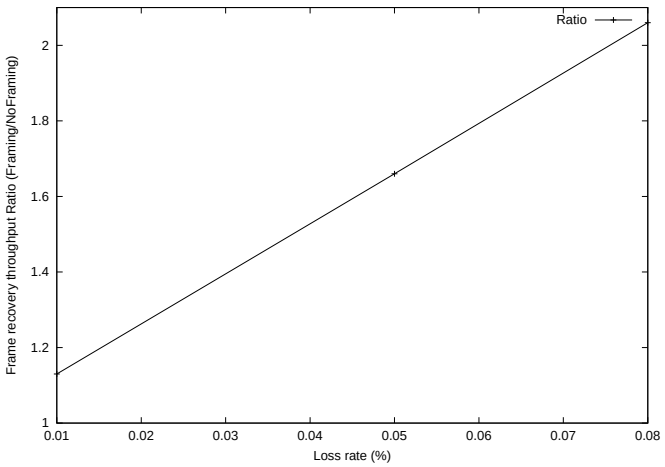


Figure 5.12: *Frame recovery throughput ratio between with framing and without framing support. RTT*BW = 1250 frames.*

5.3.5 RTM Performance Improvement

For a real time application using TCP-RTM, when the receiver skips over a "hole", with framing support, it is trivial to find the header of the next frame (assuming the frame size

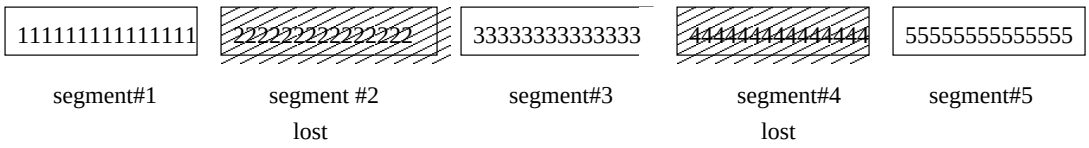


Figure 5.13: *Although segment#2 and segment#4 are lost, the frame headers of segment#3 and segment#5 can be trivially recovered, because they align with the TCP segment boundaries.*

is smaller than the MTU, a common case for voice applications), because the frame header aligns with the TCP segment, and when the application reads the data after the "hole", it receives the complete frame with the header information right at the very beginning. Figure 5.13 shows such an example.

In addition, without framing support, every time the receiver skips over a "hole", it may miss two or more application level frames because each "hole" may contain two or more partial application-level frames. However, with framing support, only one application-level frame is missed. This is illustrated in Figure 5.14, based on the measurement data presented in Chapter 4. It shows that, for example, when the network loss rate is 10%, and when the RTT is 80ms, using RTM with the framing support, the application may miss only 188 frames out of 5000 frames, while without framing support, the application may miss 376 frames. Using the framing support, the application-level loss rate can be reduced by 50%.

5.3.6 Simplifying Applications

In addition to significantly improving performance, framing support simplifies the process to recover frames. Because application-level frames are aligned with transport level segments, it is straightforward to recognize frame boundaries and recover frames, and it significantly reduces application complexity.

Furthermore, this reduction of complexity is more important for hardware design. Higher complexity means more sophisticated circuits, larger die area and hence more expensive hardware. The new framing support can significantly cut the cost of hardware when transport protocols and record-based data transfer applications are built into hardware.

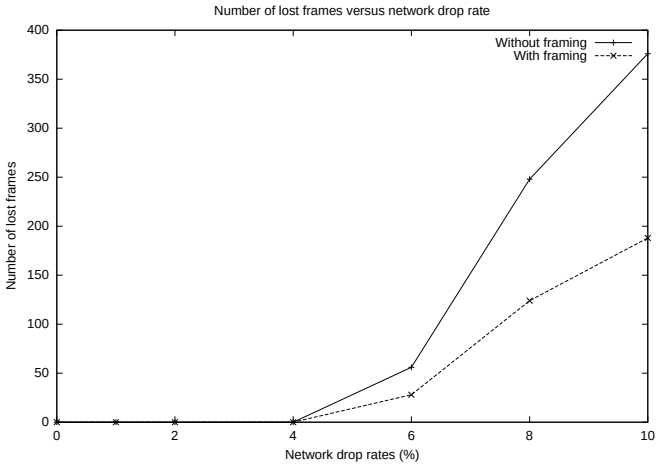


Figure 5.14: *Comparison of the number of lost frames between using the framing support and not using the framing support*

5.3.7 No Performance Penalty for Bulk Data Transfer

The TCP framing option improves the performance for record-based applications. In the mean time, it does not hurt the performance for traditional non-record-based bulk data transfer applications compared to regular byte-stream mode.

In our data transfer experiment over a 100Mbps network, we transmitted a file of 90MB between two computers running Linux operating systems. The throughput of the data transfer was the same (72Mbps) whether the framing option is used or not.

Actually, using the framing mode could be slightly faster. This is because at the receiver, the TCP code does not need to concatenate each segment in the kernel, instead, it just needs to return each segment to the application directly.

5.3.8 No Option Negotiation Needed

With framing support, the transport protocol is backward compatible. There is not any harm if one side uses the framing option when the other side does not. If only one side turns on the framing option, the TCP protocol still operates correctly, the only issue is that there is no framing benefit unless both the sender and the receiver enable the framing option.

Therefore, hosts with the framing option implemented can interoperate with endhosts that do not have framing implemented yet, and this makes the framing option incrementally

deployable.

5.4 Summary

In this chapter, we first identified the performance and complexity problems associated with the lack of framing support in the transport layer. We then described a simple extension to a conventional reliable transport protocol to add framing support. In summary, higher level applications are assumed to use an application-level framing protocol, and break large application frames into sizes that fit the MTU. The framing support we have proposed makes recovery and reassembly of higher-level frames simpler and efficient. With experimental and analytical data, we show that such a simple extension solves the framing problems by significantly improving the performance for many record-based applications, in addition to simplifying their implementations. For semi-reliable applications, the framing support can significantly reduce the application-level loss rate. In other words, the framing extension provides a simple solution to a number of complicated problems. In addition, for applications that have frames smaller than the MTU, the higher-level framing protocol can be optimized out. Furthermore, the new framing option is backward compatible and incrementally deployable.

Chapter 6

Multicast Extension

6.1 Introduction

TCP was designed for one-to-one communication, not one-to-many. To provide multipoint data delivery, it is currently common to use multiple unicast TCP connections from the source server to each receiver. For example, individual TCP streaming is usually done from a media server to a relatively large audience for a distance lecturing session. The obvious problem with this approach is that it can cause high load on the source server, and it can generate heavy duplicate traffic on the network, which wastes network resources and may cause severe network congestion. An often cited example is that when Yahoo attempted to broadcast a Victoria's Secret fashion show on the Internet, the heavy load generated by an audience of hundreds of thousands of viewers quickly crashed their media servers. As another example, right after the September 11 terrorist attack, the hit rate on the CNN web servers skyrocketed in a short time, which tremendously increased the load on the servers and forced their web administrators to significantly simplify their web pages, such as removing graphics and video, and to add new servers to stay operational [96].

A better way to reduce the amount of data to transmit is to take advantage of IP multicast. However, facing the choices of a transport level protocol, the developers for multicast applications are usually forced to use either UDP over IP multicast, in which case the developers have to implement their own mechanisms to achieve in-order data delivery, error recovery, etc.; or they need to choose one of the specialized reliable multicast protocols ([39, 92]), which have special APIs and are not widely deployed. Generally, neither of these approaches, at least in their current forms, is considered satisfactory. Therefore, finding a

simple and deployable transport level protocol that can effectively support in-order data delivery, error recovery and congestion control for multicast applications is still an open research topic.

Also, most of these reliable multicast protocols are designed for very large scale applications that may potentially have millions of receivers. However, most point-to-multipoint applications in practice only involve fewer than a thousand receivers. For example, an online lecturing session or a corporate meeting supported by video/audio streaming typically involves at most hundreds of participants and rarely thousands. It is logistically difficult to get larger number of receivers all ready to receive at the same time except for the few events with truly mass appeal. Some researchers [52] have suggested that the sizes of multicast channels may follow a Zipf [134] distribution, meaning that most multicast channels are small except for a few extremely popular ones, and such distribution may not change significantly in the future. Moreover, current practice is to distribute content in high-demand hierarchically, from a central source to local cache servers (such as an Akamai-like cache server [7]) that can then distribute to end clients. Using this hierarchical structure, a million or more receivers can be serviced while each cache server handles a thousand or fewer clients.

Therefore, we would like to concentrate on this subset of medium scale multicast applications that normally have no more than a thousand receivers, which we believe are the common case for multicast applications.

In this chapter, we describe a simple approach to extend TCP to perform multipoint data delivery. We call this approach Single-source-Multicast-Optimization (SMO), because we view TCP-SMO as a simple optimization over multiple unicasts, which we believe is needed and sufficient for most multicast applications. As shown by the experimental results obtained using our prototype TCP-SMO implementation on Linux, presented later in the paper, this approach achieves good performance for one-to-many data communication, for both reliable and semi-reliable real-time applications, by leveraging IP multicast and the power of TCP, including its guarantee of in-order data delivery, fast retransmission, and congestion control. Furthermore, this approach makes minimal addition to the application programming interface, making it trivial to adapt unicast applications to become multicast-capable.

Using TCP-SMO as an example, we would like to show that a conventional reliable unicast transport protocol can be easily extended to support multicast applications. The

extension described does not depend on any specific design of TCP, so in most places, the reference to TCP in the following sections can be interpreted as a general reliable unicast transport protocol.

The next section describes some general issues regarding multicast and reliability, and it derives a formula to identify and remove a very slow receiver in a multicast channel. Section 6.3 describes the overall design of TCP-SMO and our choices to some design issues such as acknowledgement (ACK) processing, round trip time estimation, retransmission policy, etc. Section 6.4 describes the implementation of TCP-SMO and its socket API. Section 6.5 discusses some issues regarding channel membership management and session relay. Section 6.6 presents the experimental results for performance evaluations.

6.2 Homogeneous Multicast

Fundamentally, multicast is an optimization over multiple unicasts. We only use multicast when it is beneficial, namely when all the receivers are reasonably homogeneous[1] in terms of their capability, link bandwidth and RTT, etc. It is not reasonable to multicast a file to a receiver behind a 56kbps modem and ten other receivers on a 100Mbps LAN at the same time. Instead, it is more efficient to unicast to the slow receiver, while multicasting to the other fast ones. So multicast requires homogeneity. A lot of past work on reliable multicast protocols tried very hard to support widely heterogeneous receivers in one multicast group, which significantly complicated their design.

If there is a "crying baby" in a multicast group, whose bottleneck link is particularly congested and who repeatedly requests retransmissions, it should be removed from the multicast group. We should not slow down all the other fast receivers to accommodate this one excessively slow receiver.

Intuitively, this model is consistent with the common strategy used in the real world. For example, in a school, it is assumed that all the students taking the same course should meet the prerequisite requirements of this course. During a class, the teacher tries to transfer knowledge to all the students in the class at the same time (similar to a multicast session). If one student, who does not satisfy the prerequisites, is particularly slow and keeps asking the teacher to repeat the same material over and over again, which has already been understood

[1]The term "homogeneous" here means similar within a bounded range, not necessarily "identical". It allows small differences, for example, a difference of 10%.

by most other students, it is more efficient for the teacher to help this student in a separate one-to-one tutoring session, and continue the class at a regular pace that is suitable for most students. Alternatively, this student may take another course which has a slower pace.

For real-time multicast applications, such as IPTV, we require all the receivers be able to move at the same speed, which is the speed of the content being multicast. For membership control, we use a combination of receiver-driven approach [83] and sender-enforced approach. Basically, the source server provides several channels for the same content, such as a movie, with different quality. The data-rate of each channel is basically constant based on the quality. A receiver should subscribe to a channel with which it is able to keep up in the first place. If a receiver cannot keep up after it joins a channel, the playback quality is poor, so it should quit this channel voluntarily and switch to a lower-data-rate channel. In the mean time, the source monitors the progress of each subscriber in a channel, and removes the slow receivers in case they do not quit by themselves, which is important to free up resources. The server does not change the data-rate of a channel just because some one cannot keep up.

For reliable multicast bulk data transfer, we use a different strategy. The data rate of the server is not pre-defined, instead, it is adaptive to the speed of the receivers. The speed of the receivers is required to be similar within a bounded range, so that no one is too slow relative to the majority. The server adapts to the slowest receiver in the bounded range in order to be fully reliable, which means it tries to transmit the data reliably to all the receivers. The bounded range needs to be carefully determined, so that the slowest one in the range does not excessively slow down all the other receivers, which prolongs the latency (data transmission time) to transfer the data to everybody. Also, a receiver may fail, we cannot stop everybody else because of the failure. So the server uses a policy to identify a receiver that is too slow (the "crying baby") or has failed, then removes it from the multicast channel, and uses unicast to transfer the data to it or migrates it to a slower channel.

6.2.1 Identifying the Slow Receiver to Remove

To determine this policy, we mainly consider reducing the average latency (transmission time) for all the receivers to get the data. The criterion to identify a slow receiver is to check whether it slows down the average latency by more than a tolerance percentage: T.

First we define some terms and notations:

- N: total number of receivers.
- D: total amount of data to transmit, in bits.
- $Majority$: the middle 90% (configurable).
- B_i: the bottleneck bandwidth of receiver R_i $(i = 1..N)$.
- B_{avg}: the average bandwidth of the majority.
- B_{slow}: the bottleneck bandwidth of the slowest receiver.
- C_{avg}: the average congestion window size of the majority.
- C_{slow}: the congestion window size of the slowest receiver.
- L: the latency (data transmission time), in seconds.
- L_i: the latency to receiver R_i $(i = 1..N)$, or the total transmission time to transmit all the data to R_i.
- L_{avg}: average latency of all the receivers, in seconds. $L_{avg} = \frac{\sum_{i=1}^{N} L_i}{N}$.
- L_{avg_m}: average latency for the multicast case, multicasting to all the receivers.
- L_{avg_mu}: average latency for the case to multicast to the majority at B_{avg}, but unicast to the slowest receiver.
- T: the latency tolerance ratio.
- TR: threshold ratio.

We use a simple example to illustrate the ideas.

As shown in Figure 6.1, suppose server S has $D = 100Mb$ of data to reliably transfer to $N = 4$ receivers: R_1, R_2, R_3 and R_4. The bandwidth of the tail links to R_2, R_3 and R_4 is all 10Mbps. But the bandwidth to R_1 is lower. The bandwidth of the link out of S is 20Mbps, and the bandwidth of the links in the middle of the multicast tree is all 100Mbps. (We assume the bottleneck is at the tail circuits, which is more common.) S can either multicast the data to R_1, R_2, R_3 and R_4 at the speed of R_1, or S can multicast to R_2, R_3 and R_4, and just unicast to R_1. Next we show how to determine which way is more efficient by using L_{avg}. The decision is basically based on how slow R_1 is compared to the other receivers. In

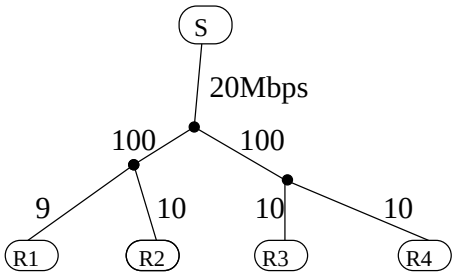


Figure 6.1: *Server S has data to reliably transmit to four receivers: R1, R2, R3 and R4. The number next to each link represents the available bandwidth on that link.*

the first scenario, we assume the bottleneck bandwidth for R_1 is 9Mbps, then in the second scenario, we assume it is 5Mbps.

Scenario 1: the bottleneck bandwidth for R_1 is 9Mbps. $B_{slow} = B_1 = 9Mbps$.

Case 1. S multicasts to all four receivers, adapting the speed to the slowest.

$L_1 = L_2 = L_3 = L_4 = \frac{D}{B_1} = \frac{100}{9} = 11.1s$

$L_{avg_m} = 11.1s$

Case 2. S multicasts to R_2, R_3 and R_4, which are homogeneous, but removes R_1 from the multicast channel, and unicasts to it.

$L_1 = \frac{100}{9} = 11.1s$, $L_2 = L_3 = L_4 = \frac{100}{10} = 10s$

$L\_avg\_mu = \frac{11.1+10\cdot 3}{4} = \frac{41.1}{4} = 10.28s$. The average latency is better than that in case 1.

In Case 1, including the slow receiver in the multicast channel slows down the overall average latency by 8%[2]. If the tolerance ratio T is 1.10, because 8% is less than 10%, we can keep it in the multicast channel and still comply with the policy. It uses less bandwidth, while keeping the latency degradation within the policy range.

Scenario 2: we assume the bottleneck bandwidth for R_1 is 5Mbps. $B_{slow} = B_1 = 5Mbps$.

Case 1. S multicasts to all four receivers, adapting the speed to the slowest.

$L_1 = L_2 = L_3 = L_4 = \frac{D}{B_1} = \frac{100}{5} = 20s$

$L_{avg\_m} = 20s$

Case 2. S multicasts to R_2, R_3 and R_4, which are homogeneous, but removes R_1 from

[2] $(11.1 - 10.28)/10.28 = 0.08$.

the multicast channel, and unicasts to it.

$L_1 = \frac{100}{5} = 20s$, $L_2 = L_3 = L_4 = \frac{100}{10} = 10s$

$L\_avg\_mu = \frac{20+10\cdot 3}{4} = \frac{50}{4} = 12.5s$.

In Case 1, including the slow receiver in the multicast channel slows down the overall average latency by 60%[3], which is much larger than 10%, so the policy requires that it be removed from the multicast channel and unicast to it.

This example shows that when the slow receiver's bottleneck bandwidth (9Mbps in the above example) is close to that of the majority (10Mbps), it is consistent with the policy to keep it in the multicast channel. But when it is too slow (5Mbps), it is compliant to remove it, so that the majority of the receivers are not delayed excessively.

Thus we use the ratio of the average bandwidth of the majority vs. the bandwidth of the slow receiver to determine whether to remove the slow receiver. Basically, if the ratio is greater than a threshold (TR), it means that the slow receiver is too slow, and it should be removed from the multicast channel.

We derive the formula for TR as follows,

$$L_{avg\_m} = \frac{D}{B_{slow}} \tag{6.1}$$

$$L_{avg\_mu} = \frac{\frac{D}{B_{slow}} + \frac{D}{B_{avg}} \cdot (N-1)}{N} \tag{6.2}$$

$$LatencyRatio(LR) = \frac{L_{avg\_m}}{L_{avg\_mu}} > T \tag{6.3}$$

$$BandwidthRatio(BR) = \frac{B_{avg}}{B_{slow}} > TR = \frac{N-1}{\frac{N}{T}-1} \tag{6.4}$$

Equation (6.1) indicates that the average latency for the multicast case is determined by the bottleneck bandwidth of the slowest receiver in the channel. Equation (6.2) shows that the average latency for the case to multicast to the majority but unicast to the slowest receiver is a weighted average of the latency to the slowest receiver and the multicast latency based on the average bandwidth of the majority.[4] Equation (6.3) indicates the

[3] $(20 - 12.5)/12.5 = 0.60$.

[4] The average bandwidth of the majority, B_{avg} is used as an indicative of the bandwidth of the multicast tree. It is only used to compute the average multicast latency for the majority, which is used to calculate L_{avg_mu} to determine whether to remove the slowest receiver. The average bandwidth is never used to actually determine the multicast data transmission rate. After the slowest receiver is removed, if the second slowest is not removed based on these equations, the multicast connection transmits at the rate allowed by

condition when the ratio of the average latency for the multicast case vs. the average latency for the multicast plus unicast case is greater than the configured tolerance level T. Plugging equations (6.1) and (6.2) into (6.3), we can derive (6.4), which shows that when the bandwidth ratio BR exceeds the threshold ratio TR, the latency ratio will be greater than T, which means that the slowest receiver should be removed from the multicast channel.

The threshold TR is a function of the number of receivers N and the tolerance configuration T. The larger N, the smaller the threshold. This is also intuitive, because the more receivers, the less tolerant the server is of the slowest receiver, because the slowest receiver can slow down more fast receivers and increase the average latency even more.

Because S does not know the bottleneck bandwidth to each receiver in advance, it cannot use it to calculate the bandwidth ratio and compare it with the threshold. However, at the steady state, the congestion-window (cwin) size for each receiver is proportional to the respective bottleneck bandwidth, so the congestion-window size ratio (CR) is equal to the bandwidth ratio (BR), and S can use CR and compare it with the threshold ratio.

$$CongestionWindowRatio(CR) = \frac{C_{avg}}{C_{slow}} \tag{6.5}$$

Therefore, we use the following rule:

`When` $(CR > TR)$`, change the slowest receiver to unicast.`

One final problem is that the congestion window size can vary dramatically within a short period if a packet is lost (because cwin is cut by half), so instead of using the instantaneous cwin, we use a smoothed-cwin defined as follows to compute CR.

$smoothed\_cwin = (1 - \alpha) \cdot old\_smoothed\_cwin + \alpha \cdot new\_cwin$

The value of α is set to $\frac{1}{8}$[5].

The case that a receiver is dead is also covered by this rule. Because the dead receiver's smoothed_cwin quickly drops because its cwin becomes 1 after a timeout, and the CR quickly exceeds the TR, so the dead receiver is detected and removed.

After removing the slowest receiver from the multicast channel, the source unicasts to it. But if the source detects that the data-rate of the multicast channel is reduced due to this unicast session, because it actually increases the load on some bottleneck link shared with

the second slowest receiver, not the average rate of the majority.

[5]This value of α is chosen as the same weighting factor used in computing the smoothed RTT in TCP. The optimal value of α is a subject for future work.

the multicast channel, the source suspends this unicast to the slow receiver until after the multicast session finishes. For example, if the bandwidth of the link out of S in Figure 6.1 is 10Mbps, then unicasting to R1 actually slows down the multicast session. In this case, S suspends the unicast session to R1 until after the multicast session.

This subsection illustrates the trade-off between latency and bandwidth. When multicasting to all receivers, bandwidth consumption is low, but the average latency can be high due to slow receivers in the channel. When using multicast plus unicast to the slow receivers, the bandwidth consumption is higher, but the average latency can be reduced. Using the simple mechanism described above, the user can achieve good latency-bandwidth trade-off according to configurable policy.

Conversely, the policy can be made to limit the bandwidth consumption, for example, no more than three times the unicast bandwidth use, and be flexible on latency requirements. But this approach is less useful because bandwidth is usually over-provisioned today and latency is a more important metric to optimize.

These general principles regarding receiver characteristics in a multicast channel and the algorithms to identify and remove a slow receiver are used in the design of multicast extension to TCP described next.

6.3 Design of Multicast Extension to TCP

6.3.1 Overall Design

We focus on single source multicast applications and use the SSM channel model [53, 108]. A channel is defined by a (S, G) pair, where S is the address of the source server and G some designated multicast address.

In our multicast extension to TCP, the source server maintains a separate TCP unicast connection to each receiver, and it maintains a common multicast channel (S,G) that is associated with these unicast connections. The source server sends each packet to the multicast address G, and states are updated as if the packet is sent out on each of the separate TCP connections. For the source server, this setup appears as n TCP connections, one for each receiver, in addition to the multicast connection. Logically, the multicast is an optimized transmission of the data to all the receivers, with the individual TCP connections handling the recovery, etc. The ACKs from each receiver are received on each separate connection, causing unicast or multicast retransmission, as appropriate based on

the number of receivers missing the data. More discussion on retransmission policies is presented in sub-section 6.3.5.

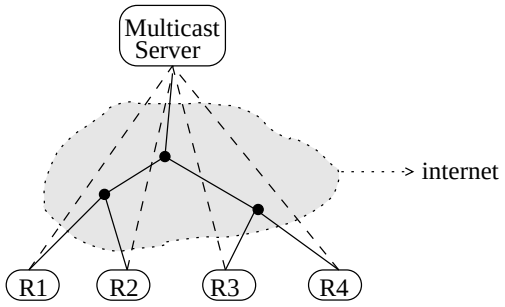


Figure 6.2: *Illustration of TCP-SMO. The solid lines represent the multicast distribution tree. The dashed lines represent the individual TCP connections between the server and the receivers. The black dots represent multicast routers.*

At each receiver, this setup appears as a single TCP connection with the source. A multicast TCP packet received addressed with (S,G) is mapped to the associated TCP connection so that a packet is processed as though received addressed to this connection (after verifying the checksum and re-mapping the destination address).

Similar to the current common service model, such as that of an HTTP server or a FTP server, we support a *receiver-initiated connection model.* At the source, a server daemon waits for a connection from a subscriber [6]. For reliable data transfer, the server can choose to wait until a certain number of connections have been established before it starts sending data. Alternatively, especially for real-time applications, the server can continuously multicast data; after a new subscriber joins the channel and establishes connection to the server, it immediately starts receiving the data being multicast by the server. This model allows asynchronous join and leave of a session, which is particularly important for a multicast session of multimedia streams.

We also support server-initiated connection model, which is more suitable for fully reliable applications. For example, when the server has a new file ready to transmit to multiple receivers, it can initiate the connection to start the transmission process.

Going from unicast to multicast, a number of design issues arise. For example, how

[6] In this chapter we are only concerned with multicast applications, so we use the term receiver, subscriber and client interchangeably, unless otherwise noted.

to efficiently manage the relationship between the multicast connection and the unicast connections for a channel, how the source server processes the acknowledgements from the multiple receivers and advances its send window, how it estimates the round-trip time for the entire audience as a whole, how the source server does retransmission when some receivers lose a packet, and so on. Next, we discuss these issues one by one.

6.3.2 Connection Management

For each multicast channel (S, G), a TCP-SMO source server creates a *master-socket* to manage the multicast connection. In addition, for each individual TCP connection to each subscriber to the channel, the source server creates a *child-socket*. These TCP connections are created to use a common sequence number for the data transmission to each subscriber.

Each *master-socket* contains a TCP Control Block (TCB) [103] that manages TCP state variables, such as sequence numbers, RTT, etc. We call this the *master-TCB* for the channel. Similarly, each *child-socket* contains a *child-TCB*.

The *master-TCB* keeps track of all the *child-TCBs* in the channel, as illustrated in Fig 6.3. When the *master-TCB* transmits a packet to the multicast channel, it informs each *child-TCB* for each receiver of the transmission, so that they can update their control information, such as the last sequence number sent, etc. When a *child-TCB* receives an ACK, after processing the ACK, it forwards the ACK to the corresponding *master-TCB* for further processing.

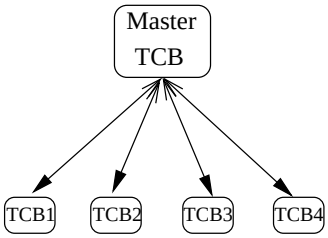


Figure 6.3: *The relationship between the master-TCB and the child-TCBs for a multicast channel.*

6.3.3 Send Window Advancement

In a TCP sender's TCB, a pointer to the send sequence space $SND.UNA$ [103] keeps track of the next sequence number the sender expects to be acknowledged by the receiver. It also indicates the left edge of the send window. Data with smaller sequence number than $SND.UNA$ can be released. In the unicast case, $SND.UNA$ is advanced whenever an ACK acknowledging new data is received. However, in the multicast case, the problem of when to advance the master-TCB's $SND.UNA$ is more involved. There are three basic options to address this problem.

1. Wait for the slowest receiver within certain limit. The *master-TCB* only advances its send window when a packet is acknowledged by all receivers within a reasonable amount of time (configurable by the application). In other words, *master-TCB* only advances its send window after all its *child-TCBs* has advanced their send window. If a receiver does not acknowledge a packet for a prolonged period of time, it would be assumed too congested or dead by the source and removed from the channel. This option is more suitable for fully reliable applications that want to assure that every single receivers receive complete data successfully.

2. Allow the application to specify an ACK-count threshold in terms of a percentage of the session size that can advance the send window in the *master-TCB*. For example, the application can specify that if 90% of the receivers have acknowledged a packet, then that packet can be released and the send-window can be advanced.

3. Buffer-limited. If the *master-TCB's* send-buffer is full, remove the oldest packet and advance the window. One issue with this approach is how to determine the send-buffer size. We use *expected-data-rate* $\times$ *max-RTT* as the buffer size.

Our design allows the application to specify one of these options, with the first one as the default (for fully reliable applications).

Conceptually, these options specify a function F to compute the $SND.UNA$ value for the *Master-TCB* based on the input of the $SND.UNA$ values of the *child-TCBs*, as illustrated in Fig 6.4.

For reliable data transfer, such as software or web cache distribution, the source server can be conservative and keep pace with the slowest receiver, in order to assure that all receivers successfully receive the exact copy of the data. Thus, these applications can

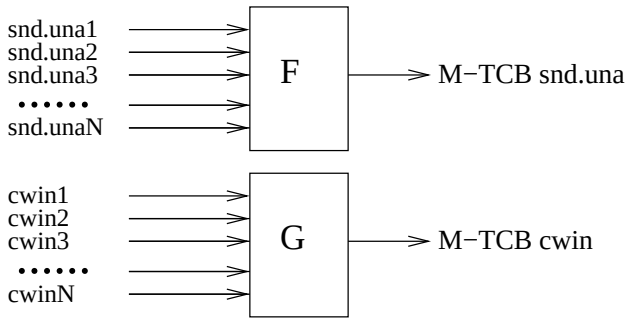


Figure 6.4: *snd.una and cwin control functions. (M-TCB refers to the master-TCB.)*

usually use option one above. However, as described in section 6.2.1, an excessively slow receiver or a failed receiver are removed from the multicast channel.

For real-time applications, each receiver is expected and required to continue at the real-time rate, just delayed by the playback delay, so no receiver can fall excessively far behind. If one does, its connection is removed from the multicast connection, and it can choose to join a lower data rate channel. For example, a web proxy could deliver broadcast video to a large number of clients. For such real-time applications, usually option two or three can be chosen.

6.3.4 ACK Processing

One important issue regards the processing of large number of ACK packets generated by the potentially large number of receivers. This has been termed as the *ACK-implosion* problem [60, 27].

However, ACK processing is not a significant overhead in all but very large collections of receivers. This is because there are fewer ACK packets than data packets (per connection) [7], and ACKs are small [8], easy to process and do not need to be buffered. Moreover, modern processors are capable of handling wire-speed packets on relatively high-speed network connections. Therefore, for the medium-scale applications with a sub-thousand audience size, which TCP-SMO targets, ACK implosion is not a significant problem. Our experimental results presented in Section 6.6 show that a regular PC with a 930MHz CPU can handle

[7] In a TCP session, there is typically one ACK for every two data packets.
[8] The size of a typical TCP ACK packet is 40 bytes.

1200 receivers without overloading the CPU.

In addition, with the wide deployment of Gigabit (or even higher bandwidth) networks in the coming years, the ACKs from even a thousand receivers only take a small percentage of the available bandwidth. For example, if each receiver generates 25 ACKs/sec, corresponding to a data transmission rate of 50 packets/sec by the source server, the bandwidth used by the ACKs from 1000 receivers is only $25 \times 1000 \times 40 \times 8 = 8Mbps$, which is less than 1% of the bandwidth of a Gigabit network. Even for a 100Mbps network, the bandwidth used by the ACKs is less than 10% of the capacity, so Gigabit networks are not required. Moreover, such traffic can be easily handled by the intermediate routers, which can process millions or even billions of packets per second in hardware [9].

Furthermore, the number of multicast servers to service a large audience is relatively small (compared to the potential number of receivers). Such server is reasonably expected to be located in a network with relatively high capacity and connected to relatively advanced routers. We do not expect such a multicast server to service a thousand receivers to be located behind a bandwidth-restricted tail circuit, such as a modem dial-up line. So such scenario is not considered in our design.

To further alleviate the ACK processing burden, we also propose an *Adaptive Acknowledgement* scheme to reduce the ACK frequency when no packet loss occurs. Currently, a TCP receiver typically generates one ACK packet for every two received packets. After the slow start phase, and when no packet loss occurs, a TCP receiver can send only one ACK for every four to eight received packets. As a result, the number of ACKs that the multicast server needs to process can be significantly reduced. To avoid the packet burstiness caused by reduced ACK frequency, TCP-pacing [69] can be used to smooth out the packet transmission.

6.3.5 RTT Estimation and Packet Retransmission

Each individual *child-TCB* maintains its own RTT and may timeout independently. The RTT for the multicast channel, maintained in the *master-TCB*, by default takes the maximum (within a configurable limit) of the RTTs to all the receivers. If the RTT to a receiver is excessively longer than other receivers, four times longer than the average, this receiver

[9] For example, as of September 2002, a Cisco Catalyst 4500 switch (http://www.cisco.com) can process up to 48 million packets per second. The 25,000 ACKs generated by the one thousand receivers in the above scenario uses only about 0.05% of the switching capacity.

is removed from the channel and its RTT ignored. Alternatively the application can choose to specify a percentile number to use the maximum RTT of certain portion of the receivers, for example the maximum of the faster 90%. Note this test on RTT is independent from the test on congestion window size described in subsection 6.2.1.

Each *child-TCB* can perform fast-retransmit independently when certain number (typically three) of duplicate ACKs are received, and it can retransmit a packet when its retransmission timer expires.

Whether to do unicast retransmission or multicast retransmission is determined by using a threshold number. This threshold number indicates the point where the server determines that there is a bigger probability that unicast retransmissions would use more bandwidth in the network than multicast retransmission, and when this threshold is exceeded, the server retransmits the missing packet by multicast. This threshold number (in terms of a portion of the session size) can be specified by the application, while the default value is set at 5%. The optimal value for this threshold number is a subject for further research.

6.3.6 Congestion and Flow Control

Each individual *child-TCB* maintains its own congestion window size (*cwin*) using the standard TCP congestion control algorithms [80], such as slow start, exponential back-off and congestion avoidance. The *master-TCB* derives the master *cwin* from the *cwin* values of the *child-TCBs* according to application configurable options. Such an option specifies a function G as illustrated in Fig 6.4.

By default, the master *cwin* takes the minimum *cwin* value (the C_{slow} defined in subsection 6.2.1) of all the *child-TCBs*. This results in a multicast session that is *absolutely fair* [128] to all competing TCP unicast flows on all overlapping links. In other words, the multicast session is completely TCP friendly with respect to any other TCP unicast flow along any shared path. This option is suitable for fully reliable applications, which need to be assured that the data is reliably transmitted to all receivers and is absolutely fair to all competing TCP flows, so it transmits at the rate that the most congested receiver allows. This must be controlled within a certain limit to avoid indefinite waiting for an excessively congested or a dead receiver. If the *cwin* of a connection is excessively small, that connection is removed from the multicast channel and is serviced using unicast. The formula to identify an excessively congested receiver is given in subsection 6.2.1.

Alternatively, we adopt the "essentially fair" concept proposed by Wang et al. [128] and

provide an option to create an "essentially fair" multicast session. Basically, this scheme allows the bandwidth used by the multicast session to be different from that of a competing TCP flow, but bounded by certain parameters, as in the following equation:

$$a \times \lambda_{TCP} \leq \lambda_{TCP\_SMO} \leq b \times \lambda_{TCP} \tag{6.6}$$

Where λ_{TCP_SMO} is the bandwidth used by the TCP-SMO multicast session; λ_{TCP} is the bandwidth used by a competing TCP flow; N is the multicast session size; a and b are two bounding parameters, and $a \leq b < N$. a and b can be specified for TCP-SMO.

The rationale behind this scheme is that since a multicast session is servicing a large number of receivers, it may be acceptable for it to use slightly more bandwidth on an intermediate link.

Most real-time multimedia applications move on at a relatively constant rate according to the media content being transmitted and the encoding schemes. The cwin for TCP-SMO (C_{SMO}) is determined by the media content data-rate: $C_{SMO} = MediaDataRate{*}RTT$. To support receivers behind variously congested links, they use the "essentially fair" approach. The cwin for TCP-SMO can be slightly larger than the smallest cwin in the channel, as long as it is within the configured bound, as shown in equation (6.6). If the cwin ratio outgrows the bound, namely, $\frac{C_{SMO}}{C_{slow}} > b$, the corresponding slow receiver is removed from the multicast channel.

The receive window size for flow control is similarly handled as the congestion window size.

6.4 Implementation and Socket API

We try to leverage existing TCP semantics and its API, and avoid unnecessary changes.

Basically, the way applications construct the TCP-SMO server and receivers is exactly the same as regular TCP applications, such as an HTTP server and a web-browser. The only new steps are that the server needs to use a new socket option to specify the multicast channel information and control parameters, and that the receiver needs to subscribe to the channel before connecting to the server. If the application developer chooses to use the default TCP-SMO control parameters, he/she needs to add just one line each in the traditional TCP server (step 3 in section 6.4.1) and client programs (step 2 in section 6.4.2), and can immediately take advantage of the benefits of multicast.

Next we delve into some technical details and describe the implementation of TCP-SMO and its socket API with an example.

At the source server, with address S, a server daemon creates a TCP socket, which is to be used for accepting TCP connections from the subscribers. We call this socket the *listening-socket*. Then it uses a new socket option to create a TCP-SMO socket associated with the *listening-socket*, which we call the *master-socket* [10], and to specify (in the same socket-option) a multicast group address G and a destination port number MP for this socket, which define the channel (S, G) with destination port MP. In addition, the application can also specify certain control parameters on the *master-socket* with this same socket option, such as the parameters that choose the retransmission policy, the RTT threshold to remove a receiver, or the congestion window control policy, etc., as discussed in the previous section. The server daemon then waits for TCP connection requests from the receivers on the *listening-socket*, just as any regular TCP server program. After certain connections are set up, the server daemon can start multicasting data to the receivers by writing data to the *master-socket*.

At the receiver, the application creates a regular TCP socket, then subscribes the socket to the channel it is interested in, for example (S, G) using standard socket options, which are being used today by multicast applications based on UDP. Next, it just connects to the server as usual and starts receiving data that is multicast by the server. On the same receiver host, multiple sockets can subscribe to the same channel and receive multicast packets.

The following uses a simple example to further describe the API as well as the detailed protocol stack changes. For illustration, we use a sample topology with one source S and two receivers *R1* and *R2*, as in Fig 6.5.

First we define some abbreviations:

- S: the server's source address.
- SP: the server's source port.
- G: the channel's multicast address.
- MP: the channel's multicast port.[11]

[10] This is the same as the *master-socket* introduced in the previous section.

[11] For simplicity, we recommend, but not require, making SP the same as MP, so that when advertising the multicast channel, only S, G, and MP need to be advertised.

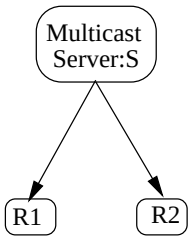


Figure 6.5: *Sample topology to illustrate TCP-SMO's operation.*

- *RA1*: receiver R1's address.
- *RP1*: receiver R1's port.
- *RA2*: receiver R2's address.
- *RP2*: receiver R2's port.

6.4.1 At the Source Server

The source host follows the following steps to set up the server, take connections from the clients and transmit multicast data to the clients.

1. Use system call socket() to create a regular TCP socket, which is the *listening-socket*.
2. Call bind() to bind the socket to *[S, SP]*.
3. Use a new socket-option *TCP_SMO* on the *listening-socket* to create a new socket, called the *master-socket*, which contains a *master-TCB* that controls the multicast TCP operations. This socket-option also specifies the *master-socket* to use *G* and *MP* as its destination. In addition, this socket-option also contains some control parameters to control the operation of the *master-TCB*.
4. Call listen() and accept() on the *listening-socket* to wait for connection requests.
5. When *S* receives a TCP SYN packet from R1, whose 4-tuple [12] is *[S, SP, RA1, RP1]*, *S* creates *TCB1* for *R1*, and replies with a SYN/ACK. *TCB1* contains a pointer to its

[12] A 4-tuple identifies a TCP connection. It contains the local source address, local source port, remote destination address and remote destination port. In this section, for clarity, we always list the server's address and port as the first two items in a 4-tuple.

parent, the *master-TCB*, and *TCB1* is added to a table in the *master-TCB*. This table is used to manage the relationship between the *master-TCB* and the *child-TCBs*, as illustrated in Fig 6.3.

6. When *S* receives a TCP SYN packet from R2, whose 4-tuple is *[S, SP, RA2, RP2]*, it creates *TCB2* for *R2*, and also replies with a SYN/ACK. The relationship between *TCB2* and *master-TCB* is similarly set up as in the previous step.

   The *master-TCB* owns a send-buffer. The children *TCB1* and *TCB2* don't have their own send-buffer, instead they share the *master-TCB's* send-buffer in a read-only manner. Also, *TCB1* and *TCB2* use the same sequence numbers that point to the send-buffer in *master-TCB*.

7. When *S* receives the ACKs from R1 and R2, the 3-way handshakes are completed, and the connections to R1 and R2 are successfully set up. The source server application can send data to the *master-socket*, which is multicast to the receivers. The data packet uses the multicast address *G* as its destination address, and *MP* as its destination port, but its protocol type is TCP instead of UDP.

8. ACKs from R1 are delivered to *TCB1*, and ACKs from R2 are delivered to *TCB2*. As described in sub-section 6.3.4, these ACKs are also forwarded to the *master-TCB* to determine whether to advance the send-window in the *master-TCB*.

9. The connection tear-down is similar to regular TCP. The source server can close the socket connections to R1 or R2 at any time. The only extra step to do is to remove the leaving TCP connection (*TCB1* or *TCB2*) from the *master-TCB*.

6.4.2 At the Receivers

The receiver host follows the following steps to connect to the server and to receive multicast packets sent by the server.

1. Create a regular TCP socket.

2. Subscribe to the channel with *(S, G)* using standard socket options.

   Since TCP-SMO supports multiple sockets on the same receiver host to subscribe to the same channel and make connections to the server, the kernel needs to keep track

of all the sockets that subscribe to *(S, G)* with a hash-table:*multicast-mapping-table* (*MMT*) indexed on *(S, G)*.

3. Send SYN with 4-tuple *[S, SP, RA1, RP1]* to the source server *S*.

4. Get SYN/ACK from *S*, reply with an ACK, set up TCB.

5. Receive a packet with 4-tuple *[S, SP, G, MP]*. Verify its checksum using the multicast address *G*. Look up *(S, G)* in the *MMT*, find all the sockets that subscribe to this channel, make a copy of the packet and deliver it to each of these sockets. The ACK sequence number in the multicast packet is ignored by the receiving socket.

   Each socket replies with an ACK independently.

   Note that such a receiver socket is able to receive both multicast packet and unicast packet that is retransmitted from the source server.

6. After the data transmission is finished, close the socket and un-subscribe from the channel.

In summary, the only API change is a new socket option at the server to specify the group address and the port number to specify the channel, and to specify some control parameters. Normally, the default parameters should be acceptable for most situations.

6.5 Membership Management and Session Relay

6.5.1 Channel Membership Management

A receiver joins the channel it wants to listen to before issuing a connect request to the source server. When the connection is closed, the receiver leaves the channel. We use IGMP [42, 17] for membership management and require no new mechanism.

The server can send a TCP "Reset" message to a receiver to close the connection. When the receiver gets the "Reset" message, it should leave the channel.

Using the existing multicast transport protocols, the source has no knowledge of each receiver, so there is no easy way to get the number of receivers and the connection time of each receiver, and it is difficult to build an effective billing model. Using TCP-SMO, the source server has the knowledge of each receiver, and it can obtain the number of receivers and keep track of the connection duration of each receiver for accounting and

billing purposes. When the source server is charged by the ISP, it can then bill each receiver, possibly based on the receiver's connection duration.

6.5.2 Session Relay

Session relay is used to support multi-source multicast applications, especially most-single-source applications, with the SSM model[13].

TCP-SMO supports session relay in a natural fashion. Each receiver can send any data back to the source server through the individual TCP connection. The source server can monitor the data and determine whether to multicast the data to all receivers.

This allows an end-receiver to actively participate in a multicast session. For example, in a live televised lecture session, a remote student can ask a question through the unicast TCP connection. When the teacher, who is at the source, receives the question, he/she can choose to repeat the question by broadcasting it to the entire class before he/she answers it.

6.6 Multicast Extension Evaluation Results

We created a prototype implementation [14] of TCP-SMO based on Linux kernel 2.2.12. Using this implementation, we conducted a series of experiments to study the performance of TCP-SMO regarding ACK processing, reliable data distribution and real-time multimedia data distribution, as well as its scalability.

6.6.1 ACK Processing

The first thing we would like to find out was whether TCP-SMO could handle the large number of ACKs generated by the potentially large number of receivers. On a LAN with 100Mbps bandwidth, we ran a TCP-SMO server program on a PC with a 930MHz CPU, and we created 1200 TCP-SMO receiver sockets evenly on five similar PCs[15]. These receiver sockets subscribed to a multicast channel rooted at the server machine. After the connection process phase, the server program started to multicast data packets to transmit a file of

[13]For more details, please see section 4 in [53].

[14]The congestion control part has not been implemented in the current prototype.

[15]Note that TCP-SMO supports multiple receiving sockets on the same host for a common channel. Also, such configuration does not affect the rate and the number that ACKs are generated, so it does not affect the result of the ACK processing at the server.

36MB to the channel using TCP-SMO at a rate of about 100 packets/sec. These packets were received and acknowledged by the 1200 sockets, at the aggregate rate of about 50,000 to 60,000 ACKs/sec. These large number of ACKs were handled with ease by the server host; actually the CPU load caused by the TCP-SMO server program on the server host was lower than 0.1. After about 254 seconds, the transmission of the file was completed, and the 1200 receivers reported that they all received the complete file successfully.

When we raised the number of receivers to 1500, we observed that some TCP connections in the TCP-SMO session could not advance in a timely manner due to the loss of ACKs by the server host.

This result demonstrated that a regular PC could handle the large number of ACKs from a thousand or fewer receivers.

6.6.2 Reliable Data Distribution

Applications such as software distribution and web cache distribution to cache servers need to do reliable data transmission to multiple destinations. Because data transmission throughput is a major metric for such applications, we performed a series of experiments to measure the throughput of TCP-SMO.

We used TCP-SMO to transfer a large file to N receivers simultaneously over a *100Mbps* LAN, where N ranges from 100 to 1000. Fig. 6.6 presents the measured throughput results of these transmissions. The figure shows that TCP-SMO can deliver a throughput over 10Mbps to 100 receivers by multicast. It also shows that even for a session size of 800 to 1000, TCP-SMO can still deliver a throughput over *1Mbps*. On the other hand, the theoretical throughput [16] by using multiple unicasts is substantially lower than TCP-SMO. For example, for 100 receivers, TCP-SMO can sustain a data rate of 12Mbps, while multiple unicast can only deliver at most 1Mbps, thus TCP-SMO is more than 12 times better in terms of throughput.

The reason that, for 1000 receivers, the throughput is only 1Mbps is the large number of ACKs generated by the large number of receivers. We observed that the maximum number of ACKs per second the server can handle was about 50,000 to 60,000, and this limited the throughput of TCP-SMO for large sessions. (Though such throughput is still significantly higher than that of the multiple unicast approach.) For bulk data file transfer, each packet

[16] The actually achievable throughput for multiple unicasts will be much lower in reality due to various overhead.

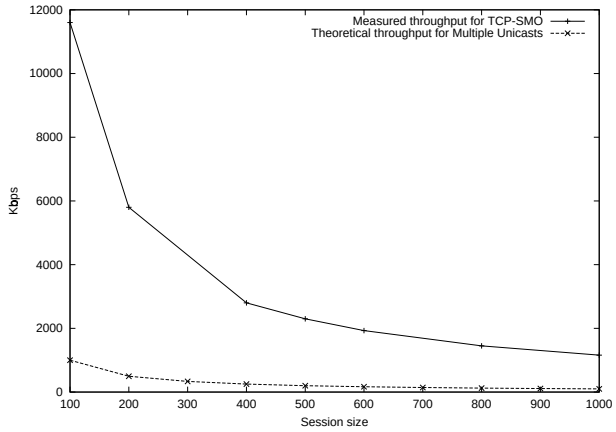


Figure 6.6: *Comparison of the throughput between multiple unicasts and TCP-SMO.*

is filled up to the maximum transmission unit (MTU), and the throughput is proportional to the number of data packets transmitted by the server per second. For a given session size, the number of ACKs generated per second is proportional to the number of data packets per second. When the number of ACKs that can be handled by the server is limited, the throughput is also limited.

The throughput can be improved by reducing the standard TCP ACK frequency. All the above results were obtained by using the standard TCP ACK frequency, i.e. one ACK for every two data packets. When we reduced the ACK frequency to one ACK for every four data packets, we observed that the achievable throughput was basically doubled for each session size.

Furthermore, we hypothesize that the throughput can also be improved when the server's Network Interface Card (NIC) performs "interrupt mitigation", a technique to generate one interrupt for multiple packets instead of one interrupt for each packet. But this needs to be verified when such NIC is available.

We also introduced artificial packet loss to some of the receivers to see if it would affect the TCP-SMO's throughput. In the experiment with 1000 receivers, when 400 of them suffered an independent loss rate of 2%, the multicast transmission using TCP-SMO still maintained a throughput at over *1Mbps*. So TCP-SMO can recover lost packets efficiently with fast-retransmits. The case that packet loss is bursty has not been studied systematically, and is left as part of our future work.

6.6.3 Real-time Multimedia Applications

Real-time multimedia applications have different characteristics from reliable data transmission applications. They are usually delay sensitive but may tolerate small packet loss rate. Traditionally, these applications use UDP as their transport protocol. However, Chapter 4 shows that a simple extension to TCP called TCP-RTM supports real-time applications well and provides good error-recovery. A major feature of TCP-RTM is that under lossy network conditions, it takes advantage of TCP's fast-retransmit feature to recover lost packets. In the case that a lost packet cannot be recovered by the time it is needed to be played back, TCP-RTM allows the receiver to skip over the lost packet and continue the playback using out-of-order packets. It provides the function to enable the receiver to trade full reliability for timely data delivery and low jitter.

Because TCP-RTM is basically the same as regular TCP, only with the option to skip over lost packets, in terms of throughput performance, TCP-RTM performs basically the same as regular TCP. So based on the data presented in section 6.6.2, we can infer that one regular PC server using TCP-SMO together with TCP-RTM can deliver a video stream of 2Mbps, which provides better than DVD video quality [104], to more than 600 receivers at the same time.

To evaluate TCP-SMO's error recovery performance for real-time applications, we performed a series of experiments to use TCP-SMO along with TCP-RTM to multicast a simulated video data stream to 100 receivers simultaneously. This video data stream has a constant packet rate of 50 packets/sec, with the size of each packet being 1448 bytes, giving a data rate at about *580kbps*. (This data rate can support better than VHS video quality [104] and is suitable to support video streaming to the current broadband Internet users using DSL or cable modems.)

For these experiments, we used a topology as depicted in Fig. 6.7. To emulate network latency, we used the Nistnet emulator [19] installed on a router between the server machine and the receiver machines. With the emulator, we conducted experiments that simulated receivers with network latency ranging from *20ms* to *100ms*. We also added packet loss simulation code in the Linux kernel's TCP receiving code and used that to simulate network packet drop rates ranging from 2% to 10% on each receiver [17]. Note that the packet loss on the receivers were not correlated, which actually caused higher load on the server because

[17] We only simulated packet loss from the server to the receivers, but did not drop any ACK packet sent by the receivers to the server. More detailed results with two-way packet loss could be found in Chapter 4.

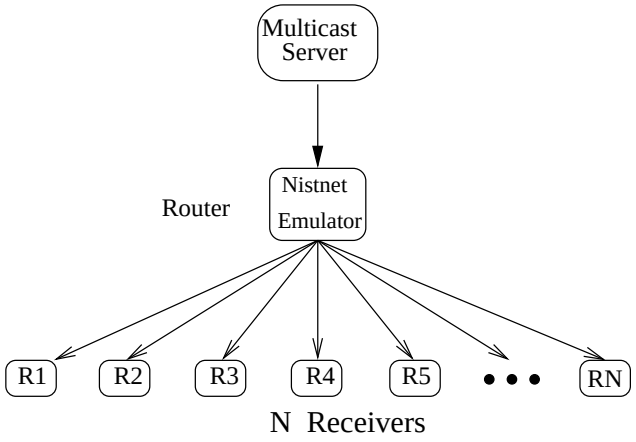


Figure 6.7: *Experiment topology.*

each lost packet needed to be retransmitted separately instead of retransmitted by multicast.

Fig. 6.8 shows the graph of the receivers' average perceived number of lost packets vs. the network drop rates. Out of the 5000 packets transmitted in each experiment, TCP-SMO along with TCP-RTM recovered almost all network dropped packets using TCP fast-retransmit, especially when network packet drop rate was less than 8%, performing substantially better than UDP.

These experimental results show that TCP-SMO along with TCP-RTM performs well for real-time multicast applications in terms of reducing the application-level perceived packet loss rate.

6.6.4 Scalability

We target TCP-SMO for medium-scale applications with fewer than a thousand receivers. But TCP-SMO is also able to support larger scale applications by using a hierarchical structure, as illustrated in Fig. 6.9. At the first level, the root multicast source server uses TCP-SMO to multicast data to the relay-servers (RS), which after receiving the data, use TCP-SMO to multicast it to the end receivers.

We conducted an experiment that used a two-level hierarchy. Three relay servers at the second level were connected to a root server using TCP-SMO. And these three relay servers used application level forwarding to forward data received from the root server to

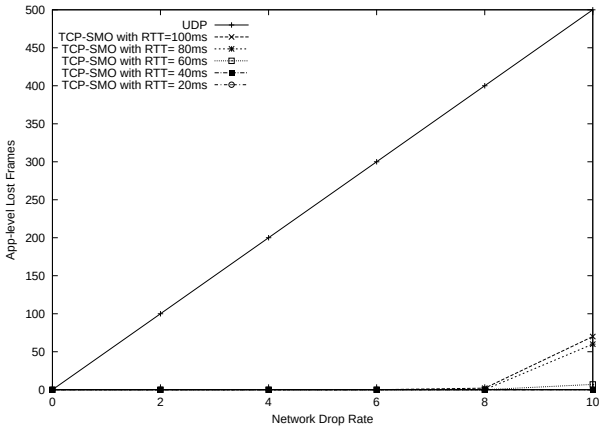


Figure 6.8: *Application level-perceived number of lost frames vs network drop rates. 5000 packets (or frames) were transmitted for each experiment. The curves for various RTTs overlap with the X-axis for network drop rates less than* 8%

a total of 2300 receivers at the bottom of the hierarchy. Experimental results showed that a data rate of 1.1Mbps could be sustained from the root server to the end receivers. This means that using a regular PC source server and three regular PC relay servers, a video stream with better than VHS quality can be delivered to more than two thousand viewers simultaneously, with an aggregate data rate of more than 2Gbps. At the same time, the load on the source server is very low – sending out just one stream. So the ratio of the aggregate data rate vs. server load is very high. If more relay servers had been used at the second level, higher data rate could be achieved.

In addition, measurement showed that the latency introduced by the relay servers was less than 0.1ms, which is basically negligible for almost all applications. As mentioned earlier in Chapter 4, a playback delay of 100ms is usually not noticeable for interactive applications. So an extra delay of 0.1ms, which is only 0.1% of 100ms, has no negative effect even for interactive applications, such as multimedia tele-conferencing.

This experiment showed that TCP-SMO could scale up to applications with thousands of receivers. In theory, assuming one TCP-SMO server can support 1000 receivers at 1Mbps, a two-level hierarchy can support $1000 * 1000 = 1million$ receivers at 1Mbps.

As mentioned in section 3.3.9 in the Related Work Chapter, TCP-SMO can be used together with an application-level multicast technique called *stream-splitting*, and TCP-SMO's hierarchical relay model is analogous to the hierarchical stream splitting scheme

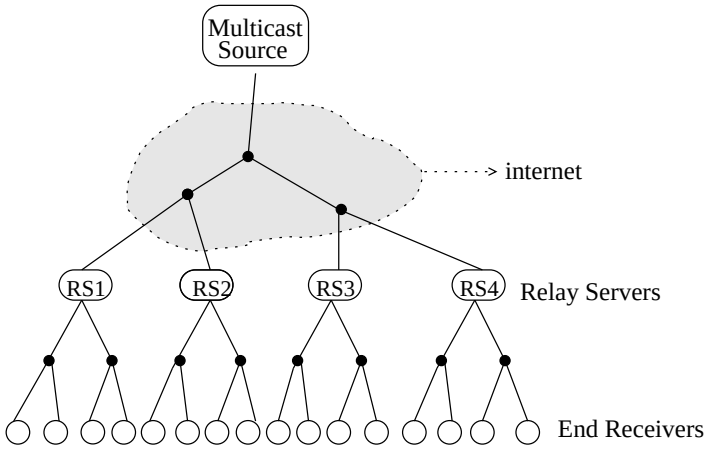


Figure 6.9: *Illustration of a two-level hierarchical structure. At the first level, the Multicast Source uses TCP-SMO to multicast to the relay servers. At the second level, each relay server uses TCP-SMO to multicast to the end receivers.*

to support a larger scale audience. The major difference is that a proxy-server that does stream-splitting needs to duplicate the data received from the source server at the application level and send out multiple copies to the end receivers, while a TCP-SMO relay server only needs to send out one data copy. Because a proxy-server for stream-splitting does application-level data duplication and it needs to do multiple writes for each packet, once for each receiver, the latency is higher.

These experiments were representative and the results did give us much confidence in the good performance of TCP-SMO for a wide range of multicast applications.

6.7 Summary

This chapter, using TCP-SMO as an example, shows that a modest extension (with about 10% of code change) to a reliable unicast transport protocol provides multicast transport functionality in a simple and clean fashion. It makes available all the benefits of a conventional reliable unicast transport protocol, such as TCP, including in-order data delivery, fast-retransmit, congestion control and its extensively fine-tuned implementation, for multicast applications.

Our analysis and experimental results show that this approach provides the following

five major advantages. Because all the techniques used in TCP-SMO that are based on TCP can be applied to a general reliable unicast transport protocol, in the following text, to avoid unnecessary wordiness, TCP is used in the place of a general reliable unicast transport protocol.

- It preserves the TCP semantics and the existing TCP socket API, adding just one new socket option. Such simplicity and full compatibility with TCP enables either easy adaptation of existing unicast applications or familiar development of new applications to take advantage of multicast transmission, achieving higher efficiency in the utilization of network and server resources. Most of the other multicast transport protocols either do not have well-defined API or have special unfamiliar APIs, which complicates multicast application development.

- It offers simple implementation and is more deployable. As a measure of its low implementation complexity, our Linux kernel implementation of TCP-SMO required fewer than a thousand lines of code change, about ten percent of the original TCP implementation.

- It provides good performance for target applications. First, it uses much lower network bandwidth compared to the multiple unicast approach. In the downstream direction, it takes advantage of IP multicast and substantially reduces the consumption of network resources, achieving far better results than N unicasts. In the upstream direction, the ACK overhead is not a serious problem in today's fast networks, especially for a medium scale audience with fewer than a thousand receivers, because the ACK packets take only a small percentage of the available network bandwidth, which is even smaller with Gigabit networks. In addition, with an adaptive-ACK scheme, the ACK overhead is even lower. Second, TCP-SMO provides good performance for reliable applications with hundreds of receivers, such as software or web cache distribution, even under lossy network condition. Third, when used together with TCP-RTM, TCP-SMO offers good performance for real-time multimedia applications in terms of achieving low application-level loss rates and low jitter, for receivers with various latency and network loss rates.

  Furthermore, among all the current multicast transport protocols, this approach is the only one that supports well both fully reliable multicast applications and real-time multimedia applications.

- It is scalable to support thousands of receivers with a hierarchical structure. As shown above, we performed experiments with good results that used application level forwarding at relay servers to support thousands of receivers in a two-level hierarchy.
- It provides a simple and straightforward model for multicast congestion control. Few other multicast transport protocols provide congestion control.

Given the above benefits, we conclude that this TCP extension is a more sensible engineering decision than inventing a brand-new multicast transport protocol, even though it stretches the original domain targeted by the TCP designers. It provides the key benefits of multicast while largely retaining the TCP mechanisms.

In summary, we show that a unicast reliable transport protocol can be extended to accommodate multicast. Such extension leverages the power of the existing unicast protocol and meets the requirements of today's key multipoint data delivery applications. It offers numerous benefits with reasonable trade-offs, therefore it is an sensible approach that deserves further investigation and experimental deployment.

Chapter 7

Conclusion

This dissertation first presents statistical data from various studies and shows that one transport layer protocol, the Transmission Control Protocol (TCP), can support most applications on the Internet. It then identifies three classes of major applications that are not well supported by the original design of TCP: real-time multimedia applications, record-based applications and multicast applications. Next, the dissertation surveys the state of the art and concludes that there is no widely deployed transport level protocol to support these applications. To solve these problems, this dissertation proposes three simple extensions to the most extensively used transport protocol — TCP. These extensions leverage all the functionalities of TCP, including in-order data delivery, fast-retransmit and congestion control, for the aforementioned applications.

The first extension is TCP-RTM, a new mode for TCP that allows it to support a range of real-time applications with performance superior to using a basic datagram service such as UDP. TCP-RTM allows application developers to use one transport protocol, not two or several. It also ensures real-time applications to be responsive to network congestion in a way compatible with established TCP dynamics. All these benefits are obtained with less than one percent of the code change in the TCP implementation.

The second extension is TCP-Framing, which adds a new mode to TCP to change it from a byte-stream-only protocol to also a message-based protocol. This new framing mode not only complies with the end-to-end design principle, but also significantly improves the performance for many record-based applications, in addition to simplifying their implementations.

The third extension is TCP-SMO, which provides multicast transport functionality in a

simple and clean fashion. TCP-SMO provides good performance for both reliable applications with hundreds of receivers, such as software or web cache distribution, and real-time multimedia applications by achieving low application-level loss rate and low jitter.

To evaluate their functionalities and performance, we implemented prototypes of these three extensions in a Linux kernel, and conducted both testbed-based experiments and some public Internet experiments. The resulting performance measurement data have been quantitatively analyzed, and they show that these extensions are both effective and efficient for the targeted applications.

In summary, this dissertation makes the following major original contributions:

- With TCP-RTM, we show that a modest extension to TCP allows it to support real-time applications.
- With TCP-Framing, we show that a modest extension to TCP allows it to support record-based applications.
- With TCP-SMO, we show that a modest extension to TCP allows it to support multicast applications.

The word "modest" is used here to mean "simple" and "small", in terms of both design and implementation complexity. There is no standard way to measure design complexity quantitatively, just as intellectual complexity is hard to quantify. In general, a simple design means it has only a few easy-to-understand and well-defined mechanisms. Implementation-wise, a small extension means the code change is usually less than 10%.

With these extensions, in addition to reliable unicast applications, TCP also supports semi-reliable real-time unicast applications, record-based applications, and multicast applications. All significant applications that currently run on UDP can run on TCP with the extensions, and any application-level protocol that currently runs on UDP can run on top of TCP.

Therefore, all significant applications can be supported, and the transport layer can be unified.

7.1 Protocol Convergence

The conventional wisdom is that to support the vast number of different applications, many different protocols are needed. However, as David Cheriton said, "we may be smart, but

evolution is smarter" [22]. Although numerous protocols have been invented in the network and transport layer, the evolution shows the pattern of convergence to few dominant protocols. [1]

At the network layer, IP has been selected as the winning network protocol, obsoleting all the other network layer protocols, such as IPX, CLNS and AppleTalk. Such convergence significantly simplifies the network layer and the design/implementations of routers and switches. In addition, it maximizes the global interoperability and makes the Internet a universal service. Visually, the convergence structures the network protocols like an hourglass with the IP as the waist [37].

Email WWW Phone ...	Application layer
SMTP HTTP RTP ...	Middleware
TCP UDP XTP SPP SCTP	Transport layer
IP	Network layer
Ethernet, PPP, FDDI, SONET, ATM	Datalink layer
Coax, copper, fiber, radio wireless...	Physical layer

At the transport layer, TCP has already emerged as the dominant protocol supporting most applications, which demonstrates its versatility and extensibility. We predict that the evolution is unifying the transport layer into one transport protocol — TCP, and this unification results in a new and simplified protocol picture [2], as shown in Figure 7.1.

In the research community, there has been a sentiment to invent a new protocol for each type of new applications. For example, for internet telephony, SCTP is invented; for video streaming, RTCP. While it is always beneficial to have an open mind and explore various types of solutions, it is important to recognize the engineering trade-off between extending existing protocols and inventing new ones. When faced with a new application to support, we need to ask some important questions, including the following: can the existing protocol(s) support this application sufficiently? If not, is it feasible to extend the existing protocol(s)? If a new protocol is to be invented, how much benefit can be obtained compared with extending an existing protocol under typical network conditions? In addition, in a colossal system as large as the current Internet, scalability and deployability

[1] There is also a trend that Ethernet is becoming the dominant protocol at the link layer.

[2] Note that TCP, as a transport protocol, operates on end-hosts or firewall gateways, not in the network, e.g on routers. We are not advocating placing TCP functionalities into the network, which is under debate in the research community and is out of the scope of this dissertation.

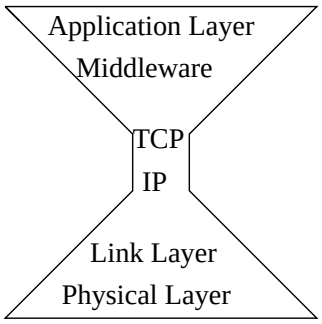


Figure 7.1: *New network protocol picture.*

are very crucial factors to consider, so an additional question is whether the new protocol is scalable and deployable.

This dissertation has shown the feasibility to extend TCP to support applications for which it was not originally designed. The complexity and cost of the extensions are low, and the performance is good. So the targeted applications can be well supported, and no new protocols need to be invented.

This convergence of the transport protocols shows that the heterogeneity at the application level can be supported by homogeneity at the transport level, or in other words, one protocol fits all.

7.2 Benefits of the Unification

Similar to the convergence at the network layer, this unification of the transport layer brings numerous benefits, including the following:

- Simplicity

  Too many transport protocols cause higher complexity and higher cost of implementation and maintenance. The unification of the transport layer significantly simplifies the protocol picture, reduces the cost of implementation and maintenance, and makes Internet applications simpler to develop and deploy because only one protocol is used.

- Interoperability and Deployability

All the proposed extensions combined change about only ten percent of a common TCP implementation. Also, no change is needed in the protocol header format, which enables the extensions to be backward compatible with the existing huge install base. In addition, because TCP is available in all the end-host operating systems, it is much easier to deploy the simple extensions described in this dissertation than numerous brand new protocols, which requires long standardization process, complicated implementation and tuning in all operating systems, and interoperability testing.

- Reliability and Security

  The complexity and large implementation size of numerous transport protocols make reliability and verification hard to obtain, and make security difficult to get [24]. Conversely, the unification makes reliability and security much easier to achieve, because there is only one protocol, not two or more, to be secured. For example, firewalls are much simpler to configure and manage.

- Hardware Optimization Feasibility

  This makes hardware optimization of the transport layer (such as TCP offloading) much more feasible. Only one protocol needs to be supported in the hardware, instead of two or more protocols, which would incur much higher cost.

7.3 Applicability

These extensions are not specific to TCP, and they do not use any specific properties of TCP. First, the TCP-RTM techniques can be applied to any reliable transport protocols that use sequence numbers, positive acknowledgements and fast-retransmit mechanisms. Second, the framing extension only relies on the packet-switching characteristic of a packet-switched internetwork. Third, the TCP-SMO techniques can be applied on any connection-oriented reliable transport protocols. In summary, these extensions are applicable to any general purpose transport protocol in a packet-switched internetwork that possesses the following basic properties:

- Connection-oriented.

- Reliable, with acknowledgement, and providing in order data delivery with sequence number.

- Congestion-adaptive.

The same techniques can be applied to other transport protocols in other packet switched internetwork, such as TP4 in OSI.

7.4 Future Work

There are a number of issues regarding the thesis of transport layer unification for future work. We present a few next.

- Video streaming quality assessment over RTM. More experiments need to be performed to assess the video streaming quality using TCP-RTM. These assessments should be performed by using various RTT and packet loss rate. Furthermore, in addition to packet-level evaluation of loss-rate and jitter, user-level evaluations should also be performed to evaluate the user-level experience.

- Multicast congestion control. TCP-SMO presents a new framework to conduct congestion control for multicast communication. However, we have not implemented the congestion control portion of the design. Experimental data need to be collected and analyzed to study how to optimally configure the congestion control parameters for multiple receivers.

- Connection set-up and state overhead. For transaction-oriented applications, such as DNS queries, to avoid the connection set-up overhead, they can either use persistent connections (for example, between DNS servers), or use T/TCP (TCP extension for transactions) [14, 15]. Because the price of memory keeps declining, the state overhead to maintain the connections is inconsequential.

- Delay sensitivity. Some researchers are concerned that using retransmission to recover lost packets may cause too high a delay for some delay-sensitive applications. As shown in Chapter 4, in most cases, lost packets can be retransmitted within the tolerable playback delay and the interactivity is not affected for real-time multimedia applications. On the other hand, on the wide area network, even without packet loss, the delay of packets may vary by an order of magnitude, and the application itself has to be designed to handle the delay variance. In addition, unless a real-time operating

system is used, the process scheduling delay may also vary by milliseconds. More work needs to be done to further evaluate this issue.

- Congestion control scheme for real-time applications. TCP's congestion control mechanism cuts the congestion-window size by half as soon as one packet is lost, and if timeout happens, TCP sets the congestion-window (cwin) size to one or two and does slow start. Some researchers think that this is too harsh to real-time applications and may hurt their quality severely. To avoid congestion collapse, every application on the Internet should be subject to congestion control, and real-time applications are no exceptions. Can we make the rate-reduction for real-time applications less abrupt? This is an important research question. If there is such a less abrupt mechanism that are still effective to avoid congestion collapse (for example, cutting the cwin by 1/8 instead of by 1/2), then such mechanism should be built into TCP and adopted for all applications. It is a subject for future research to find a more effective congestion control scheme.

- Retransmission schemes. TCP-RTM uses a send-buffer and controls the retransmission process for lost packet, and this process is transparent to the application. TCP-RTM always retransmits the first un-acknowledged packet. When UDP is used, the application protocol uses its own send-buffer, and has flexibility in terms of choosing which packet to re-transmit when packet loss happens. Some researchers would like to tag each packet with certain priority, for example, tagging an MPEG I frame with higher priority than a B frame. When packet loss happens, they would like to be able to choose a packet with higher priority to retransmit first, instead of retransmitting the first un-acknowledged packet. They think that this way the content quality at the receiver would be better. More experiments need to be performed to further study the effect of various retransmission schemes under typical network conditions.

  We think such prioritized retransmission might be useful when the network loss rate is rather high, but its benefit is probably negligible when the network loss rate is low, such as below 1%, because almost all lost packets can be successfully retransmitted and there is no need to make the distinction. If the network loss rate is too high, the overall quality of the media will be bad no matter which packet is retransmitted, and the real problem is with the network itself and that is what really needs to be fixed.

7.5 Conclusion

Finally, we conclude that we can and we should unify the transport layer of a packet-switched network with a single transport protocol.

Bibliography

[1] http://research.ivv.nasa.gov/RMP.

[2] Nortel Networks Alteon Content Cache.
http://www.nortelnetworks.com/products/library/collateral/94001.13-10-01.pdf.

[3] Realnetworks & Cisco.
http://www.realnetworks.com/solutions/enterprise/partners/cisco.html.

[4] Robust Audio Tool. `http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/`.

[5] Rock Band Broadcasts Live on the Internet from Hong Kong to Japan.
http://www.netapp.com/case_studies/japanrock.html.

[6] AX.25 Amateur Packet-Radio Link-Layer Protocol Version 2.0, Octobor 1984.

[7] Akamai Technologies Incorporated. Akamai Technology Overview, May 30 2001.
http://www.akamai.com/html/en/tc/tech_edgeadvantage.html.

[8] Mark Allman and Aaron Falk. On the Effective Evaluation of TCP. *ACM Computer Communications Review*, October 1999.

[9] A.Markoulou and F. Tobagi and M. Karam. Assessment of VoIP quality over Internet backbones. In *InfoCom*, 2002.

[10] J. W. Atwood, O. Catrina, J. Fenton, and W. Timothy Strayer. Reliable Multicasting in the Xpress Transport Protocol. In *Proceedings of the 21st Local Computer Networks Conference*, October 1996.

[11] H. Balakrishnan, H. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proceedings of ACM SIGCOMM '99*, 1999.

[12] Hari Balakrishnan. *Challenges to Reliable Data Transport over Heterogeneous Wireless Networks.* Ph.D. Dissertation, University of California at Berkeley, Department of Computer Science, Berkeley, California, 1998.

[13] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, Mark Stemm, Elan Amir, and Randy H. Katz. TCP Improvements for Heterogeneous Networks: The Daedalus Approach. In *Proceedings of the 35th Annual Allerton Conference on Communication, Control and Computing*, Urbana, Illinois, October 1997.

[14] R.T. Braden. Extending TCP for Transactions – Concepts. In *Internet Requests for Comments (RFC 1379)*, November 1992.

[15] R.T. Braden. Extending TCP for Transactions – Functional Specification. In *IETF Internet drafts (work in progress)*, December 1992.

[16] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data. In *Proceedings of ACM SIGCOMM '98*, Vancouver, British Columbia, August 1998.

[17] B. Cain, S. Deering, B. Fenner, I. Kouvelas, and A. Thyagarajan. Internet Group Management Protocol, Version 3. In *IETF Internet drafts (work in progress)*, March 2001. draft-ietf-idmr-igmp-v3-07.txt.

[18] Georg Carle and Ernst W. Biersack. Survey of Error Recovery Techniques for IP-based Audio-Visual Multicast Applications. *IEEE Network*, 11(6):24–36, December 1997.

[19] Mark Carson. Nistnet Network Emulator. http://snad.ncsl.nist.gov/itg/nistnet/.

[20] S. Cen, C. Pu, and J. Walpole. Flow and Congestion Control for Internet Media Streaming Applications. In *Proceedings of Multimedia Computing and Networking*, 1998.

[21] Jo-Mei Chang and N. F. Maxemchuk. Reliable Broadcast Protocols. *ACM Transactions on Computer Systems*, 2(3):251–273, August 1984.

[22] David R. Cheriton. RE: Another random thought: innovation, firewalls and protocol design. Personal Email. January 15, 2003.

[23] David R. Cheriton. UIO: A Uniform I/O System Interface for Distributed Systems. *ACM Transactions on Computer Systems*, 5(1):12–46, 1987.

[24] David R. Cheriton. Exploiting Recursion to Simplify RPC Communication Architecture. In *Proceedings of ACM SIGCOMM '88*, pages 76–87, Stanford, California, August 1988.

[25] Stuart Cheshire and Mary Baker. Consistent Overhead Byte Stuffing. *IEEE/ACM Transactions on Networking*, 7(2), April 1999.

[26] D-M. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer networks and ISDN systems*, 17:1–14, 1989.

[27] George Varghese Christos Papadopoulos, Guru Parulkar. An Error Control Scheme for Large-Scale Multicast Applications. In *InfoCom*, 1998.

[28] K. Claffy, Greg Miller, and Kevin Thompson. The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone. In *Proceedings of INET*, 1998.

[29] D. Clark and D. Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *Proceedings of ACM SIGCOMM '90*, pages 201–208, September 1990.

[30] Danny Cohen. Specifications for the Network Voice Protocol (NVP). Research Report ISI/RR-75-39, USC/ISI, January 1976.

[31] Douglas E. Comer. *Internetworking with TCP/IP, Volume 1, Principles, Protocols and Architecture.* Prentice Hall, 1995.

[32] Walid S. Dabbous and Christophe Diot. High Performance Protocol Architecture. *Computer Networks and ISDN Systems*, 29(7):735–744, 1997.

[33] S. Deering and D. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–111, May 1990.

[34] S. E. Deering and D.R. Cheriton. Host groups: A Multicast Extension to the Internet Protocol. In *Internet Requests for Comments (RFC 966)*, December 1985. Obsoleted by RFC 988.

[35] S.E. Deering. Host Extensions for IP Multicasting. In *Internet Requests for Comments (RFC 1112)*, August 1988.

[36] Stephen Edward Deering. *Multicast Routing in a Datagram Internetwork*. Ph.D. Dissertation, Stanford University, Department of Computer Science, Stanford, California, December 1991.

[37] Steve Deering. IETF 51, August 2001.

[38] Dante DeLucia and Katia Obraczka. Multicast Feedback Suppression Using Representatives. In *Proceedings of IEEE INFOCOM '97*, Kobe, Japan, April 1997.

[39] Christophe Diot, Walid Dabbous, and Jon Crowcroft. Multipoint Communication: A Survey of Protocols, Functions and Mechanisms. *IEEE Journal on Selected Areas in Communications*, 15(3), April 1997.

[40] Dane Dwyer, Sungwon Ha, Jia-Ru Li, and Vaduvur Bharghavan. An Adaptive Transport Protocol for Multimedia Communication. In *IEEE International Conference on Multimedia Computing Systems*, June 1998.

[41] Fusun Ertemalp, David Cheriton, and Andreas Bechtolsheim. Using Dynamic Buffer Limiting to Protect Against Belligerent Flows in High-speed Networks. In *Proceedings of the International Conference on Network Protocols (ICNP-2001)*, 2001.

[42] W. Fenner. Internet Group Management Protocol, Version 2. In *Internet Requests for Comments (RFC 2236)*, November 1997.

[43] S. Floyd. References on RED Queue Management.

[44] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[45] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, 1997.

[46] Chuck Fraleigh and et. al. Packet-Level Traffic Measurements from a Tier-1 IP Backbone. ATL Technical Report TR01-ATL-110101, Sprint, Nov. 2001.

[47] M. Gaynor. Proactive Packet Dropping Methods for IP Gateways. `http://www.eecs.harvard.edu/~gaynor/final.ps`, November 1996.

[48] Jim Gemmell. Scalable Reliable Mutlicast Using Erasure-Correcting Re-sends. MSR Technical Report MSR-TR-97-20, Microsoft Research, 1997.

[49] Jim Gemmell, Eve Schooler, and Jim Gray. Fcast Multicast File Distribution: "Tune in, download, and drop out". In *Proceedings of the IASTED International Multimedia Systems and Applications (IMSA '99)*, Nassau, Bahamas, October 1999.

[50] M. Handley. An Examination of Mbone Performance. Research Report ISI/RR-97-450, USC/ISI, April 1997.

[51] H. Holbrook, S. Singhal, and D. Cheriton. Log-Based Receiver-reliable Multicast for Distributed Interactive Simulation. In *Proceedings of ACM SIGCOMM '95*, pages 328–341, Cambridge, Massachusetts, August 1995.

[52] Hugh W. Holbrook. *A Channel Model for Multicast.* Ph.D. Dissertation, Stanford University, Department of Computer Science, Stanford, California, August 2001.

[53] Hugh W. Holbrook and David R. Cheriton. IP Multicast Channels: EXPRESS support for Large-scale Single-source Applications. In *Proceedings of ACM SIGCOMM '99*, pages 65–78, Cambridge, Massachusetts, September 1999.

[54] Hugh W. Holbrook, Sandeep K. Singhal, and David R. Cheriton. Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *SIGCOMM*, pages 328–341, 1995.

[55] Internet Software Consortium. Internet Domain Survey, 2002. http://www.isc.org/ds.

[56] S. Jacobs and A. Eleftheriadis. Adaptive Video Applications for Non-Qos Networks. In *Proc. 5-th International Workshop on Quality of Service (IWQOS'97)*, 1997.

[57] S. Jacobs and A. Eleftheriadis. Streaming Video Using Dynamic Rate Shaping and TCP Congestion Control. *Journal of Visual Communication and Image Representation*, 9(3):211–222, 1998.

[58] Stephen Jacobs and Alexandros Eleftheriadis. Streaming Video using Dynamic Rate Shaping and TCP Congestion Control. In *Workshop on Multimedia Signal Processing*, Princeton, New Jersey, June 1997.

[59] Van Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM '88*, pages 314–329, Stanford, California, August 1988.

[60] J.Crowcroft and K. Paliwoda. A Multicast Transport Protocol. In *Proceedings of ACM SIGCOMM '88*, pages 247 – 256, 1988.

[61] Jonathan B. Postel. RFC 767: A Structured Format For Transmission of Multi-media Documents.

[62] J.Romkey. A Nonstandard For Transmission of IP Datagrams Over Serial Lines: SLIP. In *Internet Requests for Comments (RFC 1055)*, June 1988.

[63] K. Miller, K, Robertson, A. Tweedly, M. White. Internet Draft: StarBurst Multicast Transfer Protocol Specification, draft-miller-mftp-spec-03.txt, April 1999.

[64] C.A. Kent and J.C. Mogul. Fragmentation Considered Harmful. In *Proceedings of ACM SIGCOMM '87*, pages 390–401, August 1987.

[65] David Kotz. Characterizing the Usage of a Campus-wide Wireless Network. In *Mobicom 2002*, 2002.

[66] Charles Krasic, Jonathan Walpole, Kang Li, and Ashvin Goel. The Case for Streaming Multimedia with TCP. In *Eighth International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, pages 213–218, 2001.

[67] Mark Kraynak. Making the Most of Streaming Media. http://www.xchangemag.com/articles/181solutions2.html.

[68] M. Krueger, R. Haagens, C. Sapuntzakis, and M. Bakke. Small Computer Systems Interface Protocol over the Internet (iSCSI) Requirements and Design Considerations. In *Internet Requests for Comments (RFC 3347)*, July 2002.

[69] Joanna Kulik, Robert Coulter, Dennis Rockwell, and Craig Partridge. A Simulation Study of Paced TCP. *BBN Technical Memorandum 1218, Bolt, Beranek, and Newman*, August 1999.

[70] Li-wei Lehman, Stephen J. Garland, and David L. Tennenhouse. Active Reliable Multicast. In *Proceedings of IEEE INFOCOM '98*, San Francisco, California, 1998.

[71] B. Levine and J.J. Garcia-Luna-Aceves. Improving Internet Multicast with Routing Labels. In *Proceedings of the International Conference on Network Protocols (ICNP-97)*, October 1997.

[72] Dan Li and David Cheriton. Evaluating the Utility of FEC with Reliable Multicast. In *The 7th IEEE International Conference on Network Protocols (ICNP'99)*, Toronto, Canada, October 1999.

[73] Dan Li and David R. Cheriton. OTERS (On-Tree Efficient Recovery using Subcasting): A reliable multicast protocol. In *ICNP*, pages 237–245, 1998.

[74] Dan Li and David R. Cheriton. Scalable Web Caching of Frequently Updated Objects Using Reliable Multicast. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems*, pages 92–103, October 1999.

[75] Jia-Ru Li, Sungwon Ha, and Vaduvur Bharghavan. HPF: A Transport Protocol for Supporting Heterogeneous Packet Flows in the Internet. In *Proceedings of IEEE INFOCOM '99*, 1999.

[76] Dong Lin and Robert Morris. Dynamics of Random Early Detection. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[77] J.C. Lin and S. Paul. RMTP: A Reliable Multicast Transport Protocol. In *Proceedings of IEEE INFOCOM '96*, pages 1414–1424, San Francisco, California, March 1996.

[78] S. Lin and D.J.Costello. *Error Correcting Coding: Fundamentals and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1983.

[79] J. Craig Lowery. Using Dell PowerApp.cache for Caching and Splitting Media Streams. http://www.dell.com/us/en/esg/topics/power_ps3q01-lowery.htm.

[80] M. Allman, V. Paxson, W. Richard Stevens. RFC 2581: TCP Congestion Control, April 1999.

[81] M. Lim and D. Kim. Internet Draft: IP Externsion for Reliable Multicast, 1997. draft-lim-ip-reliable-multicast-00.txt.

[82] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. In *Internet Requests for Comments (RFC 2018)*, October 1996.

[83] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. In *Proceedings of ACM SIGCOMM '96*, pages 117–130, Stanford, California, August 1996.

[84] Steven McCanne and Van Jacobson. Receiver-driven Layered Multicast. In *Proceedings of ACM SIGCOMM '96*, 1996.

[85] Sean McCreary and K. Claffy. Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange. In *Proceedings of the 13th ITC Specialist Seminar on Measurement and Modeling of IP Traffic*, Monterey, CA, Jan. 2000.

[86] Ahmed Mehaoua and Raouf Boutaba. The Impacts of Errors and Delays on the Performance of MPEG2 Video Communications. In *IEEE International Conference On Acoustics, Speech, and Signal Processing (ICASSP)*, Phoenix, Arizona, March 1999.

[87] Microsoft. How to Enable Live Stream Splitting in ISA Server. http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q271270&.

[88] J. Mogul and S. Deering. Path MTU Discovery. In *Internet Requests for Comments (RFC 1191)*, November 1990.

[89] Biswaroop Mukherjee and Tim Brecht. Time-lined TCP for the TCP-friendly Delivery of Streaming Media. In *Proceedings of the International Conference on Network Protocols (ICNP-2000)*, 2000.

[90] Jorg Nonnenmacher, Martin Lacher, Ernst W. Biersack Matthias Jung, and Georg Carle. How Bad is Reliable Multicast Without Local Recovery? In *Proceedings of IEEE INFOCOM '98*, 1998.

[91] UJ. Nonnenmacher, E.W.Biersack, and Don Towsley. Parity-based Loss Recovery for Reliable Multicast Transmission. *IEEE/ACM Transactions on Networking*, 6(4):349–361, August 1998.

[92] Katia Obraczka. Multicast Transport Protocols: A Survey and Taxonomy. *IEEE Communications Magazine*, Jan. 1998.

[93] R.O. Onvural. Asynchronous Transfer Mode Networks: Performance Issues. In *Artech House*, pages 44–82, Boston, 1994.

[94] ITU-T P.800. Methods for Subjective Determination of Transmission Quality.

[95] White Paper. Streaming Media Optimization with CacheFlow Internet Caching Appliances, http://www.cacheflow.com/technology/whitepapers/streaming.cfm.

[96] Pamela Parker and Christopher Saunders. Tragedy Results in Web News Gridlock. http://www.internetnews.com/IAR/article.php/12_88206/.

[97] C. Partridge. Implementing the Reliable Data Protocol (RDP). In *Proceedings of the 1987 Summer USENIX Conference*, pages 367–379, Phoenix, Arizona, 1987.

[98] C. Partridge and R. Hinden. Version 2 of the Reliable Data Protocol (RDP). In *Internet Requests for Comments (RFC 1151)*, April 1990.

[99] Craig Partridge. *Gigabit Networking*, chapter 9, pages 195–207. Addison-Wesley, 1993.

[100] V. Paxson. End-to-End Internet Packet Dynamics. *IEEE/ACM Transactions on Networking*, 7(3):227–292, 1997.

[101] C. Perkins. Options for Repair of Streaming Media. In *Internet Requests for Comments (RFC 2354)*, June 1998.

[102] C. Perkins, O. Hodson, and V. Hardman. A Survey of Packet-Loss Recovery Techniques for Streaming Audio. *IEEE Networks Magazine*, Sept./Oct. 1998.

[103] J. Postel. RFC 793: Transmission Control Protocol, September 1981. Status: STANDARD.

[104] RealVideo. RealVideo9 Data Sheet. http://www.realvideo.com.

[105] R. Rejaie, M. Handley, and D. Estrin. Quality Adaptation for Congestion Controlled Video Playback over the Internet. In *Proceedings of ACM SIGCOMM '99*, Cambridge, Massachusetts, August 1999.

[106] A. Romanow and et al. Internet Draft: RDMA over IP Problem Statement, draft-romanow-rdma-over-ip-problem-statement-00.txt, 2000.

[107] R.Stewart, Q.Xie, and et al. Stream Control Transmission Protocol. In *Internet Requests for Comments (RFC 2960)*, December 2000.

[108] S. Bhattacharyya et al. Internet Draft: An Overview of Source-Specific Multicast (SSM) Deployment, 2001.

[109] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[110] C. Sapuntzakis, A. Romanow, and J. Chase. Internet Draft: The Case for RDMA, draft-csapuntz-caserdma-00.txt, 2000.

[111] H. Schulzrinne. RTP Profile for Audio and Video Conferences with Minimal Control. In *Internet Requests for Comments (RFC 1890)*, January 1996.

[112] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. In *Internet Requests for Comments (RFC 1889)*, January 1996.

[113] H. Schulzrinne, J.F. Kurose, and D. Towsley. Congestion Control for Real-Time Traffic in High-Speed Networks. Technical Report TR 89-92, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, March 1991.

[114] M. Schwartz and D. Beaumount. Quality of Service Requirements for Audio-Visual Multimedia Services. In *ATM Forum, af-saa-94-0640*, July 1994.

[115] N. Shacham. Multipoint communication by Hierarchically Encoded Data. In *Proceedings of IEEE INFOCOM '92.*

[116] William Simpson. PPP in HDLC-like Framing. In *Internet Requests for Comments (RFC 1662)*, July 1994.

[117] D. Sisalem and H. Schulzrinne. Loss-Delay Adjustment Algorithm: A TCP-friendly Adaptation Scheme. In *Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Cambridge, UK, July 1998.

[118] Cisco Systems. http://www.cisco.com.

[119] Cisco Systems. http://www.cisco.com. See models like 7000, 7500, 12000, GSR, etc. for the deployment of RED.

[120] Telco Systems. Layer 4 Switching White Paper. http://www.telcom.com.

[121] R. Talpade and M. Ammar. Single Connection Emulation (SCE): An Architecture for Providing a Reliable Multicast Transport Service. In *15th IEEE International Conference on Distributed Computing Systems*, 1995.

[122] D. Taubman and A. Zakhor. Mutli-rate 3-D Subband Coding of Video. *IEEE Transactions on Image Processing*, 3(5):572–588, Sept 1994.

[123] K. Thompson, G. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11(6):10–23, Nov./Dec. 1997.

[124] Tony Speakerman, et al. Internet Draft: PGM Reliable Transport Protocol, draft-speakerman-pgm-spec-06.txt, Feb. 2001.

[125] UCB/LBNL/VINT Network Simulator - ns (version 2). http://www-mash.cs.berkeley.edu/ns/.

[126] David Velten, Robert Hinden, and Jack Sax. Reliable Data Protocol; RFC 908. In *Internet Requests for Comments*, number 908, July 1984.

[127] Lorenzo Vicisano, Luigi Rizzo (Pisa), and Jon Crowcroft. TCP-like Congestion Control for Layered Multicast Data Transfer. In *Proceedings of IEEE INFOCOM '98*, San Francisco, California, April 1998.

[128] Huayan Amy Wang and Mischa Schwartz. Achieving Bounded Fairness for Multicast and TCP Traffic in the Internet. In *SIGCOMM*, pages 81–92, 1998.

[129] Brian Whetten, Simon Kaplan, and Todd Montgomery. A High Performance Totally Ordered Multicast Protocol. August 1994.

[130] J. Williams, J. Pinkerton, C. Sapuntzakis, J. Wendt, and J.Chase. ULP Framing for TCP. In *IETF Internet drafts (work in progress)*, March 2001. draft-williams-tcpulpframe-01.txt.

[131] Maya Yajnik, Jim Kurose, and Don Towsley. Packet Loss Correlation in the MBone Multicast Network. In *GLOBECOM'96*, 1996.

[132] R. Yavatkar, J. Griffioen, and M. Sudan. A Reliable Dissemination Protocol for Interactive Collaborative Applications. In *ACM Multimedia 1995*, November 1995.

[133] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new Resource Reservation Protocol. *IEEE Networks Magazine*, September 1993.

[134] G.K. Zipf. *Human behavior and the principle of Least-Effort.* Addison-Wesley, Cambridge, Massachusetts, 1949.