ADAPTIVE SCALABLE INTERNET STREAMING

by

DMITRI LOGUINOV

A dissertation submitted to the Graduate School Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2002

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy

	(required signature)
Date	Chair of Examining Committee
	(required signature)
Date	Executive Officer
	(typed name)
	(typed name)
	(typed name)
	Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

ADAPTIVE SCALABLE INTERNET STREAMING
by
Dmitri Loguinov

Advisor: Professor K. Ravindran

This thesis presents a comprehensive investigation of the performance of real-time streaming in the best-effort Internet and studies several novel congestion control and retransmission methods of real-time content delivery over the public Internet. As our experiments show, these new methods provide a significantly better quality-of-service (QoS) to the end-user than the existing methods.

The first half of this work focuses on constant-bitrate streaming over the existing Internet in a large-scale performance study and examines both the behavior of network parameters and the quality of video experienced by the end users. We model select network parameters and gain an insight into what conditions future streaming applications should expect in the best-effort Internet. Furthermore, we extensively study the performance of real-time retransmission (in which the lost packets must be recovered before their decoding deadlines) based on our traces and present new methods of reducing the amount of duplicate packets generated by the server in response to lost packets.

The second half of this thesis studies congestion-adaptive streaming in the best-effort Internet. Using scalable MPEG-4 layered coding as the target application, we develop new congestion control methods that are used to rescale the enhancement layer to match the available bandwidth in the network. We find that traditional ACK-based congestion control used in TCP is not flexible enough to be applied to real-time streaming. On the other hand, as our work shows, rate-based (or NACK-based) congestion control typically does not scale to a large number of flows. To overcome this difficulty, we present a novel rate-based congestion control scheme that scales well while satisfying all requirements of a real-time application. To support this scalable congestion control, we find that the flows must possess the knowledge of the bottleneck bandwidth of an end-to-end path. Consequently, as part of this work, we present an extensive performance study of bandwidth estimation methods that can be used in real-time by the client to supplement its rate-based congestion control with the value of the bottleneck capacity.

Dedication

To my parents

Acknowledgments

I am sincerely grateful to my advisor Hayder Radha for giving me the privilege and honor to work with him over the last 4 years. Without Hayder's constant support, insightful advice, excellent judgment, and, more importantly, his demand for top-quality research, this thesis would not be possible. Even after knowing him for many years, I am continuously amazed and humbled by his infinite knowledge and unmatched wisdom.

I would also like to thank Kaliappa Ravindran for introducing me to the subject of computer networks, giving a necessary direction to my research, providing his continuous encouragement throughout my PhD, and spending countless hours in fruitful discussions.

This work would not be possible without a long-lasting support and infinite patience of Mahesh Balakrishnan and Barry Singer of Philips Research USA. Since a large part of my work was experimental, I am further indebted to Philips Research for their extreme generosity in providing the abundant resources needed for completing this PhD. I also greatly benefited from the interaction with the members of the Internet Video project at Philips Research. Many thanks to Mihaela van der Schaar, Richard Chen, Qiong Li, and Bob Cohen.

Furthermore, I would like to thank my friends and fellow students at the City University of New York for participating in numerous discussions and providing valuable feedback on various pieces of this work. I am especially grateful to Avaré Stewart for being an absolutely awesome friend and colleague, Antonio Lopez-Chavez for maintaining his firm belief in me and treating me to frequent philosophical discussions over dinner, and Andrzej Malachowicz for keeping me sane (and reasonably fit) during the early years of my studies. I also would like to thank Ted Brown for his support over the years and kindly agreeing to serve on my examination committee.

I am thankful to anonymous *IEEE INFOCOM*, *ACM Computer Communication Review*, and *ACM SIGCOMM* reviewers for providing their helpful comments on earlier versions of this work.

Last, but not least, I would like to thank my parents for teaching me the fundamentals of Computer Science at an early age. Without their initial guidance and continuous support this work would be simply impossible.

Contents

List of Tables

List of Figures

Chapter One

1 Introduction

Real-time streaming over the best-effort Internet is a very interesting and rather challenging problem. As end-user access to the Internet is becoming faster, the interest toward real-time streaming and video-on-demand services is steadily rising. Faster end-to-end streaming rates place a higher burden on congestion control to support a wide range of bitrates for each individual receiver (which is usually called *video-bitrate scalability*) and the increase in the number of users subscribing to real-time delivery of multimedia over the Internet creates a demand for congestion control than can scale to a large number of flows (which is called *flow scalability*).

The current situation with Internet streaming is far from ideal. Website visitors are usually given a menu of several available bitrates, which correspond to typical speeds of end-user access to the Internet. The user must select the bitrate that most closely matches his or her connection speed and the subsequent streaming is usually performed at

a constant bitrate (CBR), which is based on the user selection. During streaming, true congestion control is rarely used, and if the user's available bandwidth is less than the selection he or she made, video streaming becomes a sequence of short playouts followed by interruptions, which are used by the receiver to refill its decoder buffer. It is desirable, however, to scale down (and sometimes scale up) the user-selected bitstream to match the user's capabilities in cases when the speed of the access link or the currently available bandwidth are below (or above) the initially-selected bitrate. Hence, both congestion control methods that automatically find the bandwidth available to the end user and video coding schemes that support real-time rescaling of the coded video to match the rates suggested by congestion control are very important to the future of real-time streaming in the Internet.

Traditionally, video (or audio) scalability was thought of in terms of separating the coded video stream into a set of fixed-bitrate video layers. Each layer i was coded at a certain bitrate r_i and relied on the presence of layer i–1 for proper decoding. We call such schemes *coarse-granular scalable*. Hence, simple congestion control for real-time streaming that is used in the current Internet involves the subtraction of a single layer upon congestion and the addition of a single layer when probing for new bandwidth. Typical scalability includes up to 6 layers [174], [232], but in practice, streams with only one or two layers are often used. The subtraction and addition of fixed-bitrate layers represents a type of congestion control known as AIAD (Additive Increase, Additive Decrease). It has been shown [49] that such congestion control does not converge to fairness, where fairness is defined as the ability of a protocol to reach a fair allocation of a

shared resource within a certain finite time frame. Convergence to fairness is an important and desired property of congestion control in the best-effort Internet since the network itself cannot be relied upon to provide any type of fairness quality of service (QoS) to the end flows. The existing CBR and AIAD congestion control schemes perform well in the current Internet, because the fraction of Internet traffic that carries real-time material is miniscule (less than 5% [52]) and the negative effects of not utilizing proper congestion control are minor. However, if the number of CBR and AIAD flows on the Internet backbone increases to the point where real-time flows create a noticeable competition between each other, the end users and network service providers will be more likely to observe the consequences (such as rapidly increasing packet loss) of having a large number of non-responsive (i.e., CBR) or AIAD flows over shared links. Such increase in the amount of UDP traffic in the public Internet is expected since faster end-user access and higher quality of multimedia content are likely to attract a much larger population of Internet users than today.

Recent advancements in video coding created several methods of compressing video material into layers that can be easily scaled to almost any desired bitrate [224], [225], [226], [263]. Typically, the video in such schemes consists of a low-bitrate base layer coded at a fixed bitrate r_0 and a *single* enhancement layer coded up to some maximum bitrate R_{max}. During transmission, the enhancement layer can be resized to match any desired rate between 0 and R_{max} using a simple operation of discarding a certain percentage of each enhancement picture. Consequently, using a low-overhead procedure during transmission, the sender can scale the video stream to any bitrate between r_0 and

r_0+R_{max}. We refer to this kind of scalable coding as *fine-granular scalability* (FGS) and use MPEG-4's streaming profile as the model of our streaming application. Even though MPEG-4 FGS [181], [224], [225], [226] is based on embedded DCT, other methods based on wavelets provide similar functionality [263]. Since a video-scalable compression methodology already exists, the remaining problem is to find flow-scalable congestion control methods that are suitable for delay-sensitive, rate-based applications (i.e., streaming).

The majority of the existing congestion control methods have been developed for ACK-based (or window-based) flow control. In such protocols, the receiver acknowledges each received packet and the sender relies on the arrival of ACKs to decide when and how many new packets to transmit. Consequently, the instantaneous and even average sending rates of a window-based protocol arbitrarily fluctuate based on the network conditions and entirely depend on the arrival pattern of ACKs. Since the ACKs can be delayed, compressed, or lost along the path from the receiver to the sender, the sending rate of an ACK-based protocol is not known in advance (even on short time scales) and is heavily dependent on the conditions of the network. Therefore, this kind of flow control presents a certain level of difficulty when used by an inherently rate-based application such as video streaming. In real-time streaming, assuming a fixed-bitrate video layer, the sending rate for that layer must be maintained at a certain pre-defined rate (which usually equals the rate at which the video layer has been coded or *is* being coded in real-time). This rate is called the *target* streaming rate of the layer. Any deviation from the target streaming rate potentially delays video frames and makes them arrive after their decoding

deadlines. Late frames result in *underflow events* in the decoder buffer and force the video display to freeze for a certain amount of time. In real-time streaming, underflow events are one of the most important network-related pathologies that affect the quality-of-service that the end user receives from the video application and should be avoided at all costs.

Given a fixed-bitrate layer and an ACK-based application, it is possible to overcome a certain level of rate fluctuation due to delayed ACKs by introducing a larger *startup buffering delay* at the beginning of a streaming session. However, in FGS streaming, there exists an additional difficulty in using window-based flow control. In FGS, the decisions about how each enhancement layer picture is rescaled (by discarding a certain percentage of the coded frame) are made based on the *future* streaming rate r and the maximum bitrate R_{max} at which the enhancement layer has been pre-coded. If the application knows that it will sustain an average streaming rate r for a certain period of time (where r is typically given by congestion control and changes over time), the protocol will take fraction $(r–b_0)/R_{max}$ of each enhancement-layer frame and send it to the receiver (recall that b_0 is the rate of the base layer). However, if during this period of time the sending rate drops below r due to delay variation between the arrival of positive ACKs, the receiver will run into underflow events in the enhancement layer, which are undesirable as well.

Both of these difficulties (underflow events in the base layer and problems arising from inaccurate rescaling of the FGS layer) possibly can be overcome with larger startup delays at the receiver and a pessimistic choice of future sending rates r (i.e., by rescaling

the FGS layer to lower rates than the expected average throughput). However, these work-arounds eventually result in reduced quality of the video stream and unpleasant startup delays. The exact penalty of using ACK-based flow control in video streaming has not been quantified or studied in the past since the majority of video experts automatically assume some form of a rate-based transport protocol that carries their coded bitstream over the network. This also explains why current Internet streaming applications [174], [232] implement rate-based flow control.

The suitability of rate-based (or NACK-based[1]) *end-to-end* congestion control for the best-effort Internet has been questioned from the early days of the Internet[2]. Typically, end-to-end rate-based congestion control is labeled as being simply "difficult" or "unstable," and the network community has not paid much attention to NACK-based protocols since NETBLT [56]. However, the exact amount of "difficulty" in scaling rate-based congestion control has not been thoroughly studied in previous work. In NACK-based congestion control, the sender relies on the receiver to compute the next sending rate *r* and feed it back to the sender in special control messages. The absence of feedback in NACK-based protocols indicates that no change in the streaming rate is needed at this time, resulting in periods of congestion-indifferent CBR streaming between the times of receiving the feedback. Consequently, during periods of heavy congestion or persistent

---

[1] In this work, we use terms "rate-based" and "NACK-based" interchangeably when referring to congestion control. Even though the former term usually refers to the type of flow control and the latter one refers to the type of retransmission, in real-time streaming, these terms often mean the same thing.

packet loss along the path from the receiver to the sender, NACK-based congestion control becomes an "open-loop" control system, which is generally very unstable. Hence, when the network is driven into congestion by aggregating a large number of NACK-based flows, this "open-loop" operation of NACK-based congestion control contributes to a substantial packet loss increase along the links shared by these flows. The second reason for the increase in packet loss (studied in this work) is more aggressive probing for new bandwidth and increased overshoot of the bottleneck bandwidth when many rate-based flows are multiplexed over a single bottleneck.

Despite the lack of QoS support in the Internet, real-time streams should be able to achieve fairness when sharing bottleneck links with TCP connections. Currently, there is evidence that this goal can be achieved with the emerging "TCP-friendly" congestion control protocols (e.g., [13], [14], [86], [196], [198], [239]). However, there is other evidence [157] that suggests that these congestion control methods scale poorly when employed in rate-based protocols and that only ACK-based congestion control can be fully TCP-friendly. Consequently, there is a need for a thorough analysis of the existing and emerging congestion control methods in *rate-based* applications, as well as a need for new methods that can scale to a large number of users involved in real-time streaming in the future Internet.

---

[2] Rate-based congestion control on the data-link layer has been successfully adopted in ATM ER (Explicit Rate). However, these solutions do not work on the transport layer where the network cannot feed back explicit rates computed in the routers.

In this work, we shed new light on the performance of NACK-based congestion control and study its suitability for a large-scale deployment in the best-effort Internet. Our work develops a suite of new protocols for real-time streaming, which include both scalable congestion control and efficient real-time retransmission specifically tailored for rate-based applications. Since we argue that the Internet will remain best-effort for quite some time, the focus of this thesis is strictly on non-QoS IP networks (i.e., the current Internet).

1.1 Research Problem

The goal of this thesis is to "*design congestion-controlled, bandwidth scalable, real-time multimedia streaming protocols for the best-effort Internet*." In order to make this goal manageable, we identified a number of subproblems that naturally lead to the solution of the main problem:

- carry out an extensive performance study of constant-bitrate video streaming in the existing Internet and apply the learned lessons to the design of our streaming protocols;
- investigate error control methods based on retransmission and analyze their performance in real-time streaming applications over the current Internet;
- analyze the scalability of rate-based congestion control in real-time applications and design new methods that can scale to a large number of users;

- study the performance of real-time end-to-end bandwidth estimation methods and their applicability to multimedia streaming and congestion control.

1.2 Solution

First, we study the performance of real-time streaming in the Internet from the angle of an average home user in a large-scale Internet experiment and obtain a number of novel results about video and network performance perceived by current Internet users of video streaming applications.

Second, we extensively study the performance of real-time retransmission in the current Internet, define a new performance measure for assessing the quality of retransmission timeout (RTO) estimators, and propose a novel RTO estimator that achieves better performance than the existing methods when used in rate-based applications over a wide range of Internet paths.

Third, we develop a class of non-linear increase-decrease congestion control methods, which scale much better than the existing methods when used in rate-based protocols. We also find that these methods require the knowledge of the bottleneck capacity of an end-to-end link and subsequently, develop new bandwidth measurement and estimation methods that can be used in real-time to supply our congestion control with the value of the bottleneck capacity.

1.3 Contributions

This work makes the following contributions:

- *A better understanding of NACK-based congestion control and its scaling behavior.* We thoroughly study the existing methods and their scalability properties when used in a NACK-based congestion control protocol. First we define scalability measure s_n, which is the rate of packet-loss increase as a function of the number of flows n. Second, using a steady-state analysis and a continuous fluid approximation, we derive the shape of function s_n and its scaling properties. Third, we show that among the existing TCP-friendly methods, AIMD (Additive Increase, Multiplicative Decrease) scales best.
- *A class of novel NACK-based congestion control methods that can scale to a large number of flows.* These methods are a novel extension of the existing non-linear increase-decrease methods previously proposed in the literature [13]. Our theoretical results show that the new class of congestion control methods can scale to any number of flows, and our experimental results show that they do in fact possess much better scaling properties than any of the existing NACK-based protocols when used in a best-effort Internet environment.
- *New real-time bandwidth sampling and estimation algorithms*. We propose a novel concept of using bandwidth sampling methods in *real-time* (i.e., while the application is running). In addition, we develop new sampling methods that do not require any *out-of-band* traffic and only use application packets in the band-

width sampling phase. Furthermore, we design and study the performance of several novel bandwidth *estimation* methods (which extract estimates from the collected samples in real time) and make the analysis results available as part of this work. We further propose a novel concept of using bandwidth estimation in congestion control and show how these real-time bandwidth estimates can be used as part of our new congestion control methods and how they increase the scalability and robustness of a streaming protocol. Once combined with existing fine-granular scalable video compression [225], bandwidth estimation and congestion control become the core of our streaming architecture, which is another contribution of this thesis.

- *A new performance measure of the quality of retransmission timeout (RTO) estimators and a new method of estimating the RTO based on delay jitter.* First, we define a new performance measure of the quality of RTO estimators based on traces of data packets. This measure captures the inherent tradeoff of any RTO estimator between the number of duplicate packets and the amount of unnecessary waiting. Second, we conduct a large performance study of the existing RTO estimation methods in the current Internet. Third, we develop a new estimation method that performs significantly better than the existing methods over a wide variety of paths.
- *An extensive collection of real-time data traces and empirical distributions documenting network parameters and observations made during a large number of real-time Internet experiments*. These experiments were conducted over a variety

of Internet paths that span numerous autonomous systems (AS) and access ISPs. The experimental data also include Internet performance sampled by multiple video streams over paths with various end-user access technologies (i.e., analog modem, ISDN, DSL, and T1). This collection captures the quality of real-time streaming that an average end-user experiences in the current Internet, as well as the behavior of numerous network parameters along diverse end-to-end paths.

- *A real-time streaming architecture as well as a real-life Internet streaming application based on MPEG-4 FGS (Fine-Granular Scalable) coding.* Our design of the real-time streaming architecture includes both the server and the client and utilizes all findings of this thesis. Our application is capable of robust and congestion-controlled streaming of video over the existing Internet, supporting a wide variety of bitrates through scaling of the enhancement (i.e., FGS) layer to the available bandwidth. Our client and server software has been thoroughly tested over the Internet and has been found to be both reliable and scalable.

1.4 Dissertation Overview

The structure of the rest of the dissertation is shown in Figure 1 (chapters are marked with numbers and inter-chapter relationships are shown as arrows).

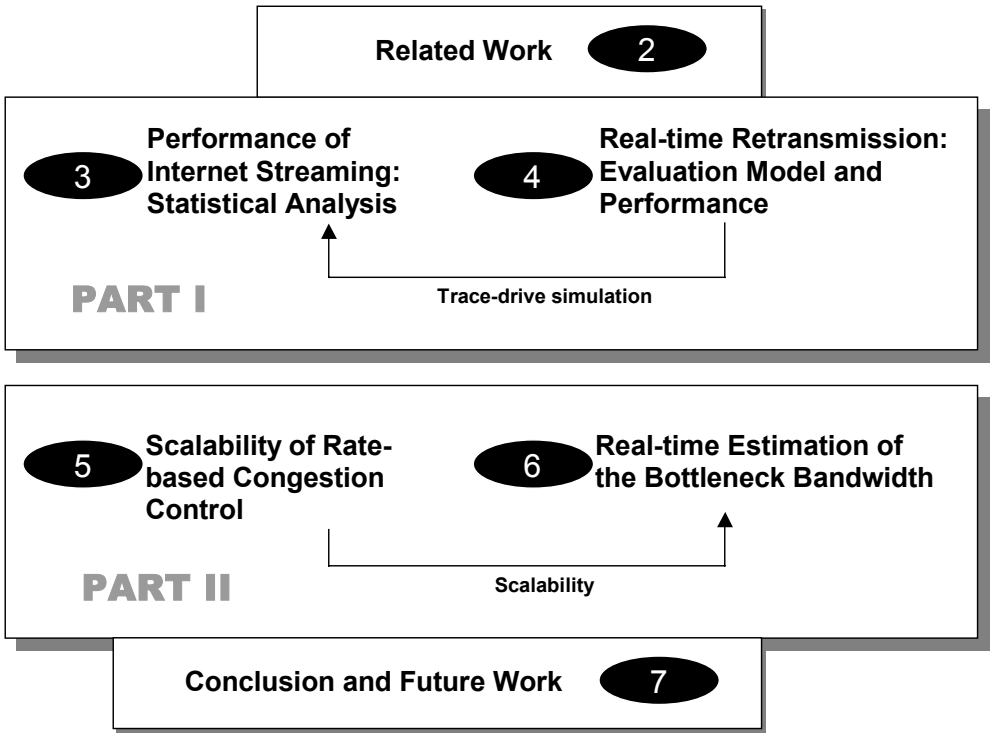

Figure 1. Structure of the dissertation.

Chapter 2 presents an overview of the background material and some of the related work. In Chapter 3, we analyze the results of a seven-month real-time streaming experiment, which was conducted between a number of unicast dialup clients, connecting to the Internet through access points in more than 600 major U.S. cities, and a backbone video server. During the experiment, the clients streamed low-bitrate MPEG-4 video sequences from the server over paths with more than 5,000 distinct Internet routers. We describe the methodology of the experiment, the architecture of our NACK-based streaming application, study end-to-end dynamics of 16 thousand ten-minute sessions (85 million packets), and analyze the behavior of the following network parameters: packet loss,

round-trip delay, one-way delay jitter, packet reordering, and path asymmetry. We also study the impact of these parameters on the quality of real-time streaming.

Chapter 4 presents a trace-driven simulation study of two classes of retransmission timeout (RTO) estimators in the context of real-time streaming over the Internet. We explore the viability of employing retransmission timeouts in NACK-based (i.e., rate-based) streaming applications to support multiple retransmission attempts per lost packet. The first part of our simulation is based on trace data collected during a number of real-time streaming tests between dialup clients in all 50 states in the U.S. and a backbone video server. The second part of the study is based on streaming tests over DSL and ISDN access links. First, we define a generic performance measure for assessing the accuracy of hypothetical RTO estimators based on the samples of the round-trip delay (RTT) recorded in the trace data. Second, using this performance measure, we evaluate the class of TCP-like estimators and find the optimal estimator given our performance measure. Third, we introduce a new class of estimators based on delay jitter and show that they significantly outperform TCP-like estimators in NACK-based applications with *low-frequency* RTT sampling. Finally, we show that high-frequency sampling of the RTT completely changes the situation and makes the class of TCP-like estimators as accurate as the class of delay-jitter estimators.

Chapter 5 studies non-linear increase-decrease congestion control methods and their performance in real-time streaming. In this chapter, we first develop a new *scalability measure* of congestion control and use it to compare the performance of the existing methods. We show that among the existing methods, AIMD (Additive Increase, Multipli-

cative Decrease) is the best solution for the currently best-effort Internet and that in order to provide better scalability than that of AIMD, the application must know the bottleneck capacity of the end-to-end path. We also define "ideal scalability" of congestion control and show that it is possible not only in theory, but also in practice; however, we find that ideal scalability also requires the knowledge of the bottleneck capacity of an end-to-end path. Finally, we develop a new method of using estimates of the bottleneck bandwidth in congestion control to achieve ideal scalability and conclude the chapter by studying the performance of the existing and proposed schemes both in simulation and over an experimental network of Cisco routers.

Chapter 6 defines a new problem of estimating the bottleneck bandwidth in real-time using end-to-end measurements applied to *application* traffic and proposes a novel use of such estimates in congestion control. We start with two basic packet-pair bandwidth *sampling* methods and show how they can be applied in real-time to the application traffic. We show how these two methods fail over multi-channel links and develop a new sampling method that remains robust over such links. We then examine the performance of the three methods in a number of tests conducted using an MPEG-4 congestion-controlled streaming application over a Cisco network under a variety of conditions (including high link utilization scenarios). In addition, we study three *estimation* techniques, which can be applied in real-time to the collected samples, and show their performance in the same Cisco network with each of the sampling methods. We find that two of the sampling methods combined with the best estimator maintain very accurate estimates for the

majority of the corresponding session in a variety of scenarios and could in fact be used by the application for congestion control or other purposes.

Chapter 7 concludes the dissertation and discusses some of the future work.

Chapter Two

2 Related Work

In this chapter, we review the related work in four main categories – measurement studies of Internet performance (section 2.1), error control in Internet transport protocols (section 2.2), Internet congestion control (section 2.3), and end-to-end bandwidth estimation (section 2.4). The background material in each of these four sections is related to our studies in the subsequent four chapters of this thesis.

2.1 Internet Measurement Studies

In this section we overview a small fraction of the vast amount of research that characterizes the performance of the Internet using end-to-end measurements. The majority of this research falls into three categories. The first category consists of TCP-based end-to-end studies of both the Internet and various flow parameters. Most of the work in this area is based on either sending bulk (i.e., FTP-like) transfers across the Internet or

monitoring end-to-end user TCP traffic at various backbone gateways in order to derive the characteristics of the paths along which the user traffic has traveled. The second category consists of ICMP-based `ping` or `traceroute` characterization studies of the Internet. The work in this area analyzed packet loss and round-trip delays as perceived by ICMP packets sent to certain destinations in the Internet. The third category deals with streaming video or audio over the Internet, but usually provides a minimal coverage of Internet paths. The primary goal of these experiments was to show that a particular streaming feature worked and had a certain performance when used over the Internet, rather than to study the Internet itself under real-time streaming conditions. Such features as congestion control, bandwidth adaptivity, retransmission, error resilience, etc., are often the main focus of the research in this category. At the end of this chapter, we also discuss work that models Internet traffic based on traces of real connections.

2.1.1 TCP measurements

Paxson's Ph.D. work [209] and a series of papers [203], [204], [205], [208], is probably the most well-known wide-scale measurement of the Internet. Paxson studied the Internet for two purposes – to investigate the routing behavior and routing pathologies in the Internet using `traceroute` packets and to study the end-to-end behavior of the Internet sampled by TCP traffic. We mention the TCP-based part of his thesis in this section and defer the discussion of the ICMP-based study until the next section.

Paxson's end-to-end TCP experiment included 25 Internet sites in December 1994 and 31 sites in November-December 1995. The geographic spread-out of these sites

was impressive and, besides the U.S., included such countries as Australia, South Korea, The Netherlands, U.K., Germany, Canada, and Norway. The experiment consisted of randomly pairing any two sites and sending a 100-KByte file from one site to the other using TCP. A packet trace tool (Unix `tcpdump`) was used to capture packet headers at both the receiver and the sender. The first part of Paxson's end-to-end analysis focused on the behavior of the participating TCP implementations and their reaction to packet loss, high RTT, and out-of-order packet delivery. The second part of the work analyzed the network conditions present during the transfers, where Paxson studied packet reordering, duplication, loss, delay, the bottleneck bandwidth, and the available bandwidth along various Internet paths. The study analyzed 21,295 end-to-end TCP transfers (over 2 GBytes of data), where each transfer consisted of 100-400 packets (over 5 million packets).

In 1992, Mogul [178] studied a large number of TCP connections passing through a busy Internet gateway. A special Unix kernel daemon was installed in the gateway to record into a disk file the headers of arriving TCP packets. The main focus of the work was the detection of compressed ACKs in TCP connections. In addition, the study measured the frequency of packet loss and out-of-order delivery of packets. During several hours of its operation, the kernel daemon collected roughly 4.1 million packets, which mainly carried email, FTP, and news (NNTP) traffic.

In 1991, Caceres *et al.* [40] performed a similar trace-based analysis of TCP conversations passing through three gateways to two campus networks. UDP, TCP, and IP headers were collected at the gateways and later analyzed with a number of off-line pro-

grams to characterize the individual connections. The study presented distributions of the number of transmitted bytes per connection, durations of connections, the number of transferred packets, packet sizes, and packet inter-arrival times. It was observed that 80 percent of all wide-area traffic belonged to TCP traffic. Of all UDP packets, 63 percent belonged to DNS (Domain Name System), 15 percent to Route, and 10 percent to NTP (Network Time Protocol). The study was based on approximately 13 million packets collected during the period of three days.

2.1.2 ICMP measurements

Paxson in [209] and [206] conducted thorough end-to-end `traceroute` measurements of over 900 Internet paths between 37 geographically distributed sites. The study consisted of periodic `traceroute` executions between random pairs of sites. Each `traceroute` execution involved a synchronized examination of the path between the two sites in both directions in order to capture possible path asymmetries. The study found that `traceroute` had traversed 751 distinct routers in 1994 and 1095 distinct routers in 1995 (for a total of 1,531 distinct routers). In terms of Autonomous Systems (AS), the study sampled 85 unique ASes. The main goal of this routing study was to examine routing pathologies, end-to-end routing instability, and routing asymmetry. The issues addressed in the work included the presence of unresponsive and rate-limiting routers, routing loops, erroneous routing, various unreachability pathologies, temporary outages, routing prevalence and persistence, frequency of route changes, and size and prevalence of routing asymmetry.

In 1993, Bolot [22] analyzed end-to-end packet loss and delay as perceived by a flow of fixed-size echo packets sent by the sender at regular intervals. The study included one site in Europe and two sites in the United States. Bolot analyzed round-trip delays, packet compression effects, and packet loss. The packet loss analysis included the study of loss gaps (i.e., lengths of loss bursts) and both unconditional and conditional probabilities of packet loss. Using Lindley's recurrence equations and a simple queuing model of Internet routers, the paper was able to estimate the amount of Internet cross-traffic from the observed inter-packet delays.

In 1993, Claffy *et al.* [53] analyzed end-to-end delays for a series of ICMP echo request packets sent from the San Diego Supercomputing Center to four sites (one in Europe, one in Japan, and two in the U.S.). The hosts maintained clock synchronization via NTP in order to measure one-way delays. The main focus of the work was to show that the one-way delay often did not equal half of the round-trip delay along the studied Internet paths. The work concluded that both routing asymmetry and different queuing delays in each direction were responsible for the difference in the one-way delays.

In 1994, Mukherjee [182] studied several Internet paths using ICMP echo requests. He found that the round-trip delay was a shifted and scaled gamma distribution and developed a method of estimating the parameters of the distribution based on the sampled RTT. The work also established a correlation between packet loss and the round-trip delay along some of the studied Internet paths.

In 1996, Acharya *et al.* [1] measured the Internet round-trip delay by sending ICMP echo packets from four hosts to 44 popular commercial and educational web serv-

ers. Each path was studied over a period of 48 hours by sending `ping` messages once per second to the corresponding web site. The work computed numerous statistical parameters (such as the mean, variance, standard deviation, mode, etc.) of the RTT and studied their behavior over time on different timescales. The distribution of the RTT was observed to be asymmetric around the mode and resembled that of a gamma distribution. The paper observed long tails in the RTT distribution, but did not provide an analytical model of the RTT. In addition, the study arrived at the conclusion that the RTT exhibited large spatial (i.e., from path to path) and temporal (i.e., over time) variation, the mode dominated the RTT distribution (i.e., most RTT samples lied in the close proximity to the mode), RTT distributions changed slowly over time (the average period before a substantial change in the RTT was computed to be 50 minutes), spikes in RTT samples were isolated, jitter in RTT samples was small, and the minimum RTT occurred quite frequently during the majority of connections.

In 1999, Savage *et al.* [245] used `traceroute` to examine various Internet paths for the purpose of finding paths with potentially better end-to-end conditions than the default paths computed by the routing protocol. New hypothetical paths were constructed for each pair of hosts A and B in the given set of hosts by traversing a path from A to all possible hosts C and then traversing the path from C to B, as long as both paths existed and were different from the default path from A to B. Some of Paxson's routing data from 1995 was used in addition to the new data collected in 1999. The results showed that certain hypothetical paths would have had better end-to-end characteristics, if they had been employed by the routing protocols.

2.1.3 UDP measurements

Yajnik *et al.* [282], [283] studied the Internet by sending equally-spaced unicast and MBone (multicast backbone) UDP audio packets. The experiment covered 128 hours in November-December 1997 and June 1998. One sender and five Internet receivers participated in the experiments, in which each receiver measured packet loss and recorded the results for further analysis. Using collected traces of lost packets, the work modeled packet loss as a random variable, analyzed its stationarity and autocorrelation properties, and established a cross-correlation between the lengths of *good runs* (i.e., the bursts of successfully received packets) and *loss runs* (i.e., the bursts of lost packets).

Borella *at el.* [29] measured packet loss rates between three Internet sites over a one-month period in April 1997. The data was collected by constructing all possible pairs out of the available sites and employing a UDP *echo* server at one site and an audio transmitted at the other site. During each three-minute connection executed once per hour, the sender transmitted low-bitrate audio traffic to the UDP echo server. Both the server and the client recorded packet loss events so that loss patterns in both directions along the path could be later analyzed. The client sent equally-spaced UDP packets at approximately 21 kb/s to the server, and the server echoed each received packet to the client. The study found that the average packet loss varied between 0.36% and 3.54% depending on the path, and the mean loss burst length was 6.9 packets. The paper modeled the upper tails of the distribution of the loss burst lengths as a heavy-tailed (i.e., Pareto) distribution and found that shape parameter α varied from path to path. In addition, the paper examined conditional and unconditional packet loss probabilities, as well

as the asymmetry of packet loss along each Internet path. The conditional loss probability was found to be much higher than the unconditional probability, and the actual probabilities were consistent with those reported in Paxson's study [205], [209]. The observed asymmetry of packet loss along each path was attributed to path asymmetries rather than to varying network load in each direction.

In a 1998 paper, Bolot *et al.* [26] conducted several streaming tests over the Internet to measure the performance of the proposed AIMD (Additive Increase, Multiplicative Decrease) rate control scheme. The scheme was based on RTP (Real-time Transport Protocol) receiver reports. The main focus of the work was to record the changes in the sending rate and show that the sender matched its transmission rate to the available bandwidth. The Internet experiment was a video conferencing session over the MBone, in which the sender responded to packet loss by adjusting its sending rate according to the proposed AIMD scheme.

In 1998, Li *et al.* [147] collected numerous samples of the RTT from packets exchanged by a public NTP (Network Time Protocol) server and modeled the round-trip delay as a long-range dependent (LRD) process. The work showed that exponential models did not fit the data well and that their distribution tails decayed too quickly. The work examined the M/M/1 model of network queues and concluded that hyperexponential distribution of the RTT describing the M/M/1 queues also provided a poor fit to the real data. Instead, the work used LRD (self-similar) modeling and estimated the Hurst parameter to vary from path to path between 0.78 and 0.86. In later work, Li *et al.* [145], [146], [148] examined the same NTP data and new round-trip delay samples derived

from experiments based on ICMP `ping`. The work built upon the fractional Brownian motion process and wavelet-based analysis to conclude that certain data traces could be characterized as incremental time series of a self-similar process. Furthermore, while some data traces were better modeled as multifractal, traces with weak long-range dependence were found to be fractional Gaussian noise.

In 1999, Tan *et al.* [263] conducted several experiments by streaming real-time video over the Internet for the purpose of testing the proposed 3D subband video coding scheme. The work tested two properties of the proposed video coding – scalability of video compression and packet loss resilience. The third motivation for the experiment was to verify that the TCP-friendly, equation-based congestion control implemented by the authors was in fact friendly to other TCP flows. The experiment involved one site in Hong Kong and two sites in North America. First, the work studied packet loss experienced by the video flows in order to show that a video stream equipped with congestion control suffered less packet loss than the same stream without congestion control. Second, the work showed that the TCP-friendly congestion control achieved approximately an equal share of bandwidth with a concurrently running TCP flow. Finally, the work showed that in the presence of packet loss, the error-resilient features of the proposed compression method performed well.

Other video streaming tests, such as [45], have been conducted over the Internet. However, they were usually confined to a single Internet path and measured the performance of the proposed protocol or scheme, rather than the performance of the Internet.

2.1.4 Internet Traffic Modeling

In 1993, Leland *et al.* [143] examined several traces of Ethernet LAN packets and discovered strong fractal behavior in the packet arrival and byte arrival processes. The work found that Ethernet LAN traffic was statistically self-similar and the Hurst parameter of the traces was a good indicator of the degree of burstiness (variance) of traffic (higher Hurst parameters corresponded to burstier traffic). The paper discussed inapplicability of the Poisson model to LAN traffic, pointing out that a finite-variance Poisson process with a fast (i.e., exponentially) decaying autocorrelation function was unable to capture the degree of burstiness and long-term correlation present in real LAN traffic. Finally, the work observed that the degree of burstiness (the degree of self-similarity) intensified with the increase in the number of sources, which also contradicted the Poisson model that predicted smoother traffic with a larger number of sources.

In 1994, Paxson [204] modeled several parameters of TELNET, FTP, NNTP, and SMTP connections based on Internet trace data. While the majority of random variables were found to be best modeled by a log-normal distribution, some exhibited heavy tails and were better modeled by a Pareto distribution. In a related work [214], Paxson studied packet arrival and connection arrival processes for TELNET and FTP sessions. The work found that the Poisson model was adequate only for describing the arrival of user sessions, but individual packet inter-arrivals were better modeled by a self-similar process. Using Whittle's estimator [143] and Beran's goodness-of-fit test [18], Paxson discovered that some traffic traces were consistent with the fractional Gaussian noise, while others, still exhibiting strong self-similarity, failed the test.

In 1995, Willinger *et al.* [279] introduced a method of generating self-similar traffic by using an aggregation of renewal reward processes, each of which modeled an ON/OFF traffic source. The duration between the ON and OFF periods for each source was modeled by an independent identically distributed random variable with a hyperbolic tail distribution (i.e., a heavy-tailed, Pareto-like distribution). The aggregate traffic was shown to be asymptotically fractional Gaussian noise and thus self-similar. Following the derivation of the model, the authors examined two Ethernet traces and presented a physical explanation of self-similarity discovered in their previous work [143]. They partitioned the data according to source-destination pairs and modeled ON/OFF periods of each source as a heavy-tailed distribution. Using Q-Q plots and a Hill estimator, the study showed that many sources exhibited the *Noah Effect* (heavy tailed distribution) with shape parameter α between 1.0 and 2.0, which the paper argued was the source of self-similarity in the aggregate traffic.

In 1996, Crovella *et al.* [60] found the presence of self-similarity in the traces of WWW traffic and established that Hurst parameter H of the traffic was between 0.76 and 0.83; however, the ON/OFF periods of user requests failed to clearly demonstrate a heavy-tailed distribution. Among other related work, Grossglauser *et al.* [93] introduced a modulated fluid traffic model in which the correlation function was asymptotically self-similar with a given Hurst parameter H and modeled the behavior of a single finite server queue fed with such fluid input process. In [207], Paxson presented a fast, approximate method of generating fractional Gaussian noise based on fast Fourier transforms and provided a method of dramatically improving the speed of Whittle's estimator of the Hurst

parameter. Recently, Ribeiro *et al.* [242] used multiplicative superstructures on top of the Haar wavelet transform to generate long-range dependent (LRD) traffic and showed that the results matched real self-similar data very well. Another recent work, Feldman *et al.* [71], showed that actual WAN traffic exhibited more complex behavior than that of self-similar (monofractal) processes and suggested a wavelet-based time-scale analysis technique to model the underlying multifractal nature of WAN traffic.

Self-similarity has been also noticed in VBR video traces. Garrett *et al.* [89] found that the tail behavior of the marginal bandwidth distribution of a VBR source could be accurately described using heavy-tailed distributions and that the autocorrelation function of a VBR video sequence decayed hyperexponentially (i.e., indicating an LRD process). The paper combined the findings into a new model of VBR traffic and used self-similar synthesis to generate LRD video traffic. Huang *et al.* [105] synthesized VBR video traffic by using an asymptotically self-similar process (*fractional autoregressive integrated moving average* (F-ARIMA)) that modeled the marginal distribution of empirical data and long-range, as well as short-range, dependencies in the original autocorrelation function.

2.2 Error Control in Transport Protocols

We start this section by discussing the work related to retransmission-based lost packet recovery and finish with FEC (forward-error correction)-based error recovery and error-concealment methods.

2.2.1 Retransmission schemes for real-time media

In 1996, Dempsey *et al.* [63] developed a retransmission-based buffer model for packet voice streaming. The model was based on the paradigm that the voice data consisted of silence and talk-spurt segments and involved a receiver buffering delay to support retransmission. The proposed protocol relied in NACK-based flow control; however, the paper did not deal with the loss of NACKs or the estimation of retransmission timeouts.

In 1996, Pejhan *et al.* [216] studied retransmission in multicast trees. The study focused on two NACK-based retransmission models. In the first model, a unicast packet was retransmitted directly to the receiver upon the receipt of the corresponding NACK. In the second model, each lost packet was multicast to all receivers. Furthermore, the second model allowed the server to wait a certain amount of time before retransmitting requested packets to accumulate several NACKs from multiple receivers. Specifically, the paper examined the cases of immediate multicast (zero waiting time), multicast upon the receipt of all NACKs (which required the knowledge of the maximum one-way delay in the tree), and a more general case, in which the waiting time was between the minimum and the maximum one-way delay. The authors built a retransmission cost model for multicast trees under certain simplifying assumptions and presented a quantitative analysis of the various retransmission schemes taking into account the probability of generating late retransmissions, the average packet recovery time, the number of times each lost packet is retransmitted, and the cost to the network (in terms of duplicate packets) of each retransmission scheme. After analyzing and simulating the different schemes, the paper

reached a conclusion that immediate (preferably multicast) retransmission worked best and, in the most general case, was optimal in the context of all four criteria.

In 1996, Papadopoulos *et al.* [199] proposed a real-time retransmission scheme for streaming. The scheme utilized a video playout buffer and applied a special buffering delay to the video data before the first frame was passed to the decoder. The extra buffering delay was used to allow more time for the retransmission of the lost packets. The proposed scheme operated using NACKs and relied on the gaps in packet sequence numbers to infer packet loss (i.e., the scheme did not attempt to detect out-of-order packets). The sender inserted each transmitted video packet in a temporary FIFO queue of the same size as the receiver's playout buffer. If a retransmission request came after the corresponding packet had left the sender's temporary buffer, the request was ignored under the assumption that the discarded packet would not have been able to arrive to the receiver before its deadline. Among other features, the proposed scheme included conditional retransmission, which prevented the receiver from requesting packets that were likely to be late for their decoding deadline. For that purpose, the receiver maintained a weighted exponential average of the RTT samples and used this value to decide which packets were likely to be recovered before their deadlines.

In 1998, Rhee [241] proposed a retransmission-based receiver buffer model, in which lost packets were recovered until the deadline of the group of pictures (GOP) instead of the deadline of each individual lost packet. Traditionally, if a packet arrives after its decoding deadline, the entire frame containing such late packet is discarded. Rhee proposed to hold each incomplete frame until all lost packets from that frame were re-

covered (even if the full recovery happened after the frame's decoding deadline) and then use this late frame to reconstruct subsequent frames within the same GOP. However, in order to perform this "catch-up" decoding, the decoder had to work faster than the target fps, and some extra buffering was needed at the client. In the second part of the paper, Rhee applied an FEC coding scheme to the video and tested the entire video streaming scheme in simulation and over the Internet. Furthermore, the paper used a hybrid NACK-ACK retransmission scheme, in which each frame was ACKed by the receiver, and each ACK carried the information about missing packets from the corresponding frame. The sender retransmitted lost packets (as requested by each ACK), waited for three frame durations, and retransmitted the same packets again if it received another ACK showing that the same packets were still missing.

2.2.2 Retransmission in TCP and other protocols

In 1999, Allman *et. al.* [4] studied the performance of TCP's RTO estimator based on the traces of TCP connections between a large number of geographically distributed Internet sites (i.e., Paxson's 1995 dataset [209]). The study compared the performance of RTO estimators for several values of exponential weights (α, β) and concluded that no estimator performed significantly better than TCP's default RTO. Among other TCP-related retransmission schemes, Keshav *et al*. [124] described a retransmission strategy (called SMART) for a reliable transport protocol that used sender-based retransmission timeouts equal to twice the *SRTT* (i.e., identical to those in RFC 793).

Paul *et al*. [202] presented a reliable multicast transport protocol (RMTP) in which the receivers sent NACKs for the same packet at least RTT time units apart, where the RTT was "computed using a TCP-like scheme." Among other Internet protocols, Gupta *et al*. [96] proposed a web transport protocol (WebTP) designed to provide better performance than TCP. Although not directly a real-time streaming protocol, WebTP had many similarities to real-time applications, which included a rate-based flow control and a receiver-initiated (i.e., NACK-based) retransmission scheme. The retransmission time-out (RTO) used in WebTP was based on the current sending rate r (which was estimated at the receiver and conveyed to the sender in special control packets) and a TCP-like (i.e., EWMA) variance σ of the inter-packet arrival delay with undisclosed exponential weights α and β. A timeout occurred at the receiver if no packets arrived within $[1/r + M\sigma]$ time units after the arrival of the last packet, where M was an undisclosed factor that allowed for one-way delay variation. In addition, in order to be able to detect out-of-order packets, WebTP used TCP's triple-ACK scheme, in which three arriving out-of-order packets triggered a retransmission (if the retransmission had not happened earlier due to a timeout).

2.2.3 ITD Video Buffer Model and Efficient Packet Loss Recovery

Some of the previous work mentioned in this section is applicable to voice data and certain video applications that can tolerate playback jitter. However, these solutions may not be acceptable in video-on-demand services (e.g., entertainment-oriented applications). In addition, it is crucial to take into consideration the characteristics of com-

pressed video, which usually has a variable number of bytes per video frame. Due to the wide range of compressed pictures' sizes, a video buffer model has to be employed to ensure the continuous decoding and presentation of video at the receiver. It is important to note that a video buffer model is needed even in the absence of any network jitter or packet loss events. Examples of video buffer models are the ones supported by the ISO MPEG-2 and ITU-T H.263 standards.

To overcome the deficiencies of the protocols only suitable for audio, Radha *et al.* [225] introduced a joint transport-layer/video-decoder buffer model suitable for retransmission-based Internet video applications, which could tolerate short delays (e.g., on the order of a few seconds). We refer to this model as the *Integrated Transport Decoder* (ITD) buffer model. In the ITD, the decoder buffer is partitioned into two regions – the jitter detection region and the retransmission region. Missing packets in the jitter detection region are suspected to be reordered or delayed by the network due to delay jitter. Hence, their retransmission is not initiated until they leave the jitter detection region and make it into the retransmission region. The retransmission region is used to recover lost packets through multiple retransmission requests to the server. The size of each region and the delay between subsequent requests of the same lost packet are determined in real-time based on the measurements of the current network conditions.

2.2.4 Forward Error Correction

Forward error correction codes (such as the ones used in [26], [218], [233]) provide another dimension to error recovery, which is orthogonal to retransmission-based

approaches. The issue of whether FEC is better for real-time traffic than retransmission (or vice versa) is still under consideration in the research community. Clearly, in some cases FEC performs better (especially in multicast sessions and under large end-to-end delays) and in others it may perform worse (e.g., in non-interactive unicast applications with bursty packet loss). This thesis does not explicitly study or apply FEC-based methods to video, but instead studies the performance and limitations of retransmission-based packet loss recovery.

2.2.5 Error Concealment

Receiver-based error concealment (such as in [269]) hides the negative effects of damaged (i.e., lost) packets on video or audio frames by approximating the values of the lost data using the data from the adjacent frames. Error concealment has been successfully used in both audio and video streaming. However, several impediments do exist in concealing packet loss in motion-compensated video. For example, spatial interpolation in video is often hindered by the use of variable-length codes (VLC) within a coded frame. Upon packet loss, errors in the lost macroblocks of a video frame propagate into adjacent blocks and adversely affect the rest of the frame data (since the decoder loses synchronization within the VLC codes). As the result, interpolation of the lost data from adjacent macroblocks of the same frame is sometimes difficult (even with the use of reversible VLCs and resynchronization markers). In addition, temporal interpolation is often not possible either, because the use of motion compensation and strong temporal dependence between frames force errors to propagate into adjacent frames. MPEG-4 and

other video coding standards do not specify or standardize error concealment techniques that can be used to conceal the loss of entire packets or frames. Consequently, in the current Internet, either FEC or retransmission (or both) are often necessary in addition to any error concealment implemented at the video level.

2.3 Congestion Control

In this section, we present a short overview of the work related to congestion control, congestion avoidance, and adaptive video streaming over the Internet. We partitioned the work into several categories and mention a few results in each area. In the first area, we discuss several early congestion control schemes. The second area deals with TCP-related congestion control. In the third area, we overview router-based congestion avoidance schemes and recent advancements in that area. The fourth area of congestion control discusses RTP (Real-time Transport Protocol)-based methods. The fifth area consists of recently-proposed TCP-friendly (equation-based) congestion control. There is mounting evidence that formula-based congestion control schemes can be effectively used for real-time video streaming, while achieving fair utilization of the bandwidth with TCP flows. The final area consists of an overview of several video streaming solutions proposed in the past (including rate adaptation and quality adaptation mechanisms).

2.3.1 Early approaches to congestion control

In one of the first documents on congestion control following the standardization of TCP/IP, Nagle [184] discussed various possibilities of serious congestion in the

ARPANET. The work described one of the first (dating back to 1983) *congestion collapses* in a wide-area TCP/IP network and coined the term "congestion collapse," which refers to a state of the network in which the utilization stays close to 100% and the throughput stays close to zero. In addition, Nagle [184] examined the problem of high overhead associated with small TCP packets (the solution to which was later incorporated into TCP as "Nagle's algorithm") and identified ICMP Source Quench messages as the solution to congestion. Nagle suggested that a TCP sender react to ICMP Source Quenches by closing the sender's transmission window for a certain duration. On a related topic, the paper proposed that ICMP Source Quench messages be sent when the router's queues were half full, instead of when an actual packet loss occurred. Finally, Nagle proposed that, upon congestion, routers penalize flows with the highest sending rate. The latter two ideas are currently used in Random Early Detection (RED) [84].

In 1985 (RFC 969) and later in 1987 [56], [57] Clark *et al.* proposed one of the first *rate-based* flow control protocols called NETBLT (Network Block Transfer). NETBLT was an end-to-end transport-layer protocol designed for reliable high-throughput bulk data transfer. In general, congestion control in rate-based applications is more difficult than that in window-based applications. Due to this difficulty, NETBLT was never supplemented with congestion control and was left in an experimental stage with no provision for rate adaptation in response to packet loss.

In 1989, Jain [116] suggested a delay-based congestion avoidance algorithm, which adapted the sender's transmission window in response to the change in the round-trip delay. The proposed scheme assumed a deterministic network, in which the end-to-

end delay was a linear function of the sender's transmission window size. Packet loss was assumed to be non-existent and acknowledgements traveled instantaneously from the receiver to the sender. The paper developed an AIMD (Additive Increase, Multiplicative Decrease) algorithm to control the sender's window in response to the variation in the RTT. Under the assumed idealistic conditions, the paper showed that the proposed AIMD algorithm performed well and converged to an optimal lossless state. The author acknowledged that the proposed congestion avoidance mechanism was unsuitable for real networks and that future work was required to solve protocol's reliance on idealistic assumptions.

2.3.2 TCP congestion control

The basic operation of TCP is specified in RFC 793 [221]. TCP is a window-based transmission protocol, in which congestion control was not implemented until the late 1980s. The absence of congestion control in TCP led to numerous congestion collapses in the ARPANET and the early NSFNET.

RFC 793 TCP relied on timeouts to detect packet loss and recover lost packets. The computation of the retransmission timeout (RTO) was based on the current estimate of the round-trip delay, whereas current TCPs take into account the variation of the RTT as well as the RTT itself. In all versions of TCP, the smoothed estimate of the round-trip delay, which is called *SRTT*, is computed as an exponentially weighted moving average (EWMA) of the past RTT samples:

$$SRTT_i = (1-\alpha)\cdot SRTT_{i-1} + \alpha\cdot RTT_i, \quad (1)$$

where i is the sample number, RTT_i is the latest observation of the round-trip delay, and α is a smoothing factor (1/8 in the current TCP). Given the current smoothed estimate *SRTT* of the round-trip delay, the RTO in the original TCP was computed as:

$$t_{RTO} = \beta \cdot SRTT, \tag{2}$$

where β (delay variance factor) was a fixed constant between 1.3 and 2.0. In the early 1980s, it seemed reasonable to multiply the latest *SRTT* by a factor of two and expect that all round-trip delays in the immediate future would be well within the RTO. In reality, however, the RTO specified in RFC 793 was frequently insufficient when used over the Internet and led to numerous *unnecessary* retransmissions. The problem was especially noticeable, because many Internet hosts connected to the relatively slow Internet through high-speed Ethernet LANs and were able to send substantial amounts of retransmitted traffic into congested routers before realizing that their RTO was too low.

Another problem with the original TCP was the absence of any graceful startup, i.e., the original TCP started blasting packets at the maximum speed (up to the size of its initial window) before it even got its first estimate of the RTT. In cases when packets from the original burst were lost, it was too soon for the RTO to reflect the actual RTT. Consequently, to a large degree, many retransmission decisions at the beginning of a session were based on inaccurate RTOs.

In 1988, Jacobson [110] proposed the first effective version of congestion control for TCP. Jacobson, following a 1986 congestion collapse in the ARPANET, suggested several modifications to TCP that increased its stability during periods of heavy loss and

reduced the number of spurious retransmissions. The proposed modifications included *slow start*, *exponential timer backoff*, *dynamic windows sizing* (currently called *congestion avoidance*), *fast retransmit*, and *RTT variance estimator*.

Under proposed *slow start*, a TCP sender started its transmission by slowly increasing the sending rate until a packet loss occurred, instead of blasting entire windows of packets into the network. A new congestion window, *cwnd*, was used to keep track of the current number of packets that the sender could send in one RTT. Under slow start, *cwnd* started at one packet and was increased exponentially during each RTT until a packet loss occurred.

In [110], in order to increase network stability during congestion, the *exponential timer backoff* algorithm instructed a TCP sender to double its retransmission timeout each time the same packet was retransmitted. The exponential backoff slowed down the sender in case of heavy congestion (i.e., when the sender had no feedback) and allowed the network to stay in a stable state.

In addition, each packet loss triggered the *dynamic window sizing* algorithm. The algorithm recorded half of the effective transmission window at the time of a packet loss as *ssthresh* (slow start threshold), set the congestion window to one, and immediately initiated *slow start*. Slow start was executed until either another packet loss occurred or un-

til *cwnd* reached the value of *ssthresh*. When the latter condition occurred, slow start was over and *cwnd* was increased linearly to probe for new bandwidth.[3]

Fast retransmit referred to the method of retransmitting certain lost packets without waiting for a timeout. If a packet was lost or reordered, the subsequent packets arriving to the receiver triggered duplicate ACKs (i.e., ACKs that acknowledged the same data). The sender, from the receipt of duplicate ACKs, could infer that a particular packet had been lost. In order to avoid retransmitting reordered packets, the sender responded only to the third consecutive duplicate ACK (hence, the fast retransmit method is sometimes called "triple-ACK").

Finally, Jacobson [110] proposed adding an *RTT variance estimator* in the computation of retransmission timeouts (RTO). According to Jacobson's observations, under high network load, the original TCP [221] was unable to keep up with highly varying values of the RTT and caused frequent premature retransmissions. Recall that RFC 793 TCP [221] suggested multiplying the value of *SRTT* by constant β (between 1.3 and 2.0) and taking the result as the value of the RTO. Jacobson estimated that $\beta = 2$ could adapt to link utilization loads of only 30%, which was too low for frequently-congested Internet of the 1980s. As a result, [110] suggested computing the value of the RTO by adding four smoothed RTT variances to the value of the smoothed RTT (i.e.,

[3] In current literature, the behavior of TCP when *cwnd* < *ssthresh* is called "slow start" and when *cwnd* > *ssthresh* it is called "congestion avoidance." When *cwnd* = *ssthresh*, either algorithm can be executed [5], [258].

$t_{RTO} = SRTT + 4 \cdot RTTVAR$). The paper showed that the resulting RTO was able to adapt to much higher network loads.

TCP, with Jacobson's modifications, was implemented in 4.3 BSD Unix in 1988 and became known as *Tahoe TCP*. In 1990, the *fast recovery* algorithm was added to TCP's congestion control [5], [258]. In Tahoe TCP, upon *fast retransmit*, the sender was forced to go into slow start. Later, it was realized that a single packet loss (such as the one detected by fast retransmit) did not necessarily indicate a network congestion and did not require slow start. Instead, it was proposed that upon packet loss, the sender reduce congestion window *cwnd* by half and perform a modified version of congestion avoidance that quickly restored the window back to the original value in the absence of additional packet loss. The latter algorithm was termed *fast recovery* and was first implemented in 4.3 BSD Unix in 1990 (the *Reno* release).

In the next few years, Jacobson *et al.* [113], [114], [115] proposed three TCP options that have since been placed on the IETF (Internet Engineering Task Force) standards track. Besides large TCP windows, the suggested options allowed the sender to include transmission timestamps in TCP headers in order to accurately compute the round-trip delay. In addition to computing sender-based RTT, timestamps could be used to compute packet delay jitter at the receiver, even though there is no clear use of delay jitter in TCP. The third and most important option allowed the receiver to use selective ACKs (called SACKs) to request retransmission of specific lost packets (regular ACKs could still be used in conjunction with SACKs). The resulting TCP is usually referred to as SACK TCP and its detailed description can be found in [165].

The addition of SACK to TCP stems from the following observations. Upon *fast retransmit*, Tahoe TCP suffers from potentially retransmitting multiple packets that have already been successfully delivered. Reno TCP, on the other hand, is limited to retransmitting at most one lost packet per RTT. In addition, the performance of Reno TCP is substantially reduced when multiple packets from the same window are dropped (which happened in cases of bursty packet loss). Fall *et al.* [69] compared the performance of Tahoe, Reno, and SACK TCP, and showed that Reno TCP had major performance problems in the presence of bursty packet loss (performance was even worse than that of Tahoe TCP). Reno TCP performed so poorly in the presence of bursty packet loss, because it exited *fast recovery* too soon and was forced to wait for a retransmission timeout in order to recover additional lost packets within the same window. Fall *et. al*. proposed a modification to Reno, called NewReno (currently in IETF as [82]), that did not suffer from the same problem. NewReno was based on Hoe's work in [102] and instructed the sender to stay in the *fast recovery* mode until all packets from the window, which was in effect when the first packet loss was detected, were acknowledged. NewReno's performance could still be hindered by bursty packet loss, although not as severely as that of Reno. The overall conclusion of [69] was that SACK TCP performed well in all cases and that selective acknowledgements should become a part of the TCP standard [165].

Hoe [102], in addition to the changes to the fast recovery algorithm (discussed above under the NewReno modifications), suggested an adaptive way of estimating the initial value of *ssthresh* (recall that the initial *ssthresh* is used to gauge how long the sender performs the initial slow start). Large values of *ssthresh* cause the sender to in-

crease its rate too aggressively and may potentially result in overshooting the available bandwidth by up to 100% during slow start. Hoe [102] proposed that the sender estimate the bandwidth-delay product from the first several acknowledgements it receives and use the estimated bandwidth-delay product to compute *ssthresh* before slow start causes a packet loss (similar ideas were developed by Brakmo *et al.* in [35], see below). Accurate estimates can be especially beneficial to short connections (such as the majority of HTTP flows); however, accurate estimation of the bandwidth is hard in the presence of delayed and compressed ACKs. Allman *et. al*. [4] experimented with the same idea and found that in the majority of TCP connections studied in Paxson's large-scale experiment, real-time estimation of *ssthresh* was not warranted.

In 1996, Mathis *et al.* [164] proposed a *forward ACK* scheme for TCP, which they called *FACK TCP*. The proposed scheme was built on selective acknowledgements proposed earlier [113], [114], [115], [165] and took a better use of SACKs in the presence of bursty packet loss. The work suggested minor modifications to the fast recovery and fast retransmit algorithms that allowed the sender to interpret non-continuous segments of SACKs in a more intelligent way. Mathis *et al.* showed that in simulations FACK performed better than SACK-enhanced Reno due to better management of unacknowledged packets and better ability to recover from episodes of heavy loss.

Independently of the NewReno and SACK modifications to TCP, a new version of TCP was designed and simulated by Brakmo *et. al.* in 1994. Brakmo *et al.* [35] proposed a new TCP congestion control scheme called *TCP Vegas*, which incorporated a novel bandwidth estimation scheme into the TCP sender. The congestion avoidance

phase was changed to include real-time estimation of the *expected throughput* (based on the smallest experienced RTT) and the *actual throughput* (based on the latest RTT). Knowing the amount of packets in flight (i.e., window size), the per-RTT throughput could be computed as the ratio of the window size to the RTT. The paper made an assumption that as the network congestion approached, the *actual throughput* would become smaller (due to larger round-trip delays and backed up queues). Therefore, if the actual throughput dropped below a certain threshold compared to the expected throughput, TCP Vegas initiated a linear (per RTT) decrease of the congestion window. If, however, the actual throughput was close to the expected throughput and stayed above another threshold, TCP Vegas linearly increased the congestion window to probe for new bandwidth. If the actual throughput was between the two thresholds, the network was considered to be stable and the congestion window was not changed. Other changes in TCP Vegas included a modified slow start (also based on the expected and actual throughput, and a *half-rate* exponential increase), fine-granular RTO timers, and the decision not to react to multiple packet losses within the same transmission window (i.e., Vegas reacted to congestion at most once per RTT). In simulations, TCP Vegas achieved between 37 and 71 percent better throughput and incurred between 50 and 80 percent fewer lost packets than Reno TCP.

In addition to TCP Vegas, Brakmo *et al.* [35] examined the possibility of estimating the *available bandwidth* during slow start based on a modified packet-pair mechanism [121]. The new version of TCP was tentatively called Vegas* and used four consecutive ACKs arriving from the receiver during slow start to set the initial value of

ssthresh. In experiments, Vegas* had a more graceful startup behavior and rarely caused heavy packet loss. On the other hand, Vegas (as well as Tahoe and Reno) during slow start often overshot the available bandwidth by as much as 100% and caused significant packet loss. Despite a better startup behavior of Vegas*, the authors concluded that the new mechanism did not have a measurable effect on throughput and only marginally improved the loss rate. For these reasons, Vegas* modifications were not included into the final version of TCP Vegas.

One year later, Ahn *et al.* [3] ported the TCP Vegas implementation from the *x*-kernel simulator to SunOS and tested it over the Internet. They confirmed that Vegas had better congestion avoidance properties and on average retransmitted 2 to 5 times fewer packets than Reno, experienced fewer coarse timeouts, and subsequently incurred less congestion in the network. When Vegas and Reno competed for bandwidth along the same path, Reno was more aggressive and achieved up to 50 percent higher throughput. The difference in throughput was explained by the fact that Vegas was more conservative and backed off upon incipient congestion, while Reno pushed the network to the limit and forced packet loss. The study also discovered that a Reno sender behaved poorly when paired with a Tahoe receiver and that a Vegas sender actually performed better with a Tahoe receiver than with a Reno receiver. The paper concluded that Vegas offered an overall throughput improvement of 4-20 percent over Reno, much lower RTT, and a reduction in lost packets between 50 and 80 percent.

After much controversy in the Internet community, TCP Vegas received recent attention from Hengartner *et al.* in [101]. Under the same simulation conditions used in

[35], Hengartner reported similar performance of TCP Vegas to that described by Brakmo *et. al.* [35]. Besides studying the performance of TCP Vegas, Hengartner *et al.* identified which new features of Vegas were responsible for the improvement in throughput and lower packet loss. The paper found that Vegas' new *fast recovery* and *slow start* mechanisms had the most influence on the throughput and its ability to reduce packet loss, and showed that the other changes in Vegas had little impact on its performance. In addition, the study noticed that in a scenario with multiple competing Vegas flows, sessions that started earlier yielded bandwidth to sessions that started later, and that Vegas flows running along the same path were sometimes unfair to each other.

Other recent papers [97], [177] showed that Vegas' congestion avoidance algorithm was more stable than that of Reno, that Vegas was not biased against connections with large RTTs, and that, at the same time, Vegas suffered from unfairness to other Vegas flows and yielded to Reno TCP along shared paths.

Bolliger *et al.* [20] took Paxson's approach [209] and studied several versions of TCP by sending streams of packets between geographically distributed Internet sites. The study lasted for six months in 1997-1998 and involved eleven Internet sites in North America and Europe. The purpose of the study was to compare Reno and Vegas TCPs with several FACK-based versions [164]. At exponentially distributed intervals, a centralized process chose a random pair of Internet hosts and instructed them to send a 1-MByte file using a randomly chosen TCP protocol. The main result of the study was that FACK-based TCPs outperformed in all aspects Reno and Vegas TCPs. FACK TCP achieved higher throughput under all conditions, maintained fewer unnecessary retrans-

missions, had up to 50% fewer timeouts, and was able to cope with bursty packet loss much better than either Reno or Vegas. The work concluded that, even though the results somewhat contradicted earlier work [11], [153], FACK-based TCP is a considerable improvement over Reno-style TCPs.

2.3.3 Router-based congestion avoidance

One of the first router-based congestion control methods for IP networks was the ICMP Source Quench mechanism [219]. According to [219], a congested router would send a special ICMP message (called *Source Quench*) back to each source whose packet had been dropped by the congested router. In 1987, IETF gateway requirement document [34] confirmed the use of Source Quenches, acknowledging, however, that sometimes Source Quenches could worsen congestion and that routers could become overly burdened by generating such messages. In 1995, [34] was "overruled" by a new IETF router requirement document [10], in which routers were explicitly prohibited to send Source Quenches due to unnecessarily heavy CPU overhead and little effect on congestion.

In 1988, Ramakrishnan *et al.* [229] proposed a congestion avoidance scheme called DECbit. DECbit was a window-based protocol, which relied on routers to set a congestion notification bit in packet headers to signal congestion to the sources. Each router computed the average outgoing queue size for every arriving packet over the last and current busy periods. If the average queue size was larger than one, the congestion bit was set in the packet's header. If at least half of the packets in the last transmission window had their congestion bit set, the sender reduced its transmission window multi-

plicatively. Otherwise, the window was increased additively. This approach relied on instantaneous router queue size and therefore, was somewhat unfair to bursty traffic and appeared to be susceptible to instability and oscillation.

Other similar Explicit Congestion Notification (ECN) schemes have been proposed for ATM networks – Forward ECN (FECN) and Backward ECN (BECN). It is expected that ATM FECN/BECN will be replaced with the IP version of ECN that we discuss below.

Random Early Detection (RED) [84] is a congestion avoidance algorithm that allows routers to drop randomly selected packets when they detect that their average queue size has reached a certain threshold. In addition to avoiding congestion, RED targets to improve the fairness between flows and avoid global synchronization between sources. Furthermore, RED naturally penalizes sources with the highest sending rate and, upon congestion, slows down the sources that contribute to the congestion the most. Recent IETF literature [32] encourages Internet routers to support RED, and many of them do. Even though routing software supports RED, it is still not deployed in the public Internet [156] due to several implementation issues [50], [167].

In 1994, immediately following the proposal of RED in [84], Floyd [79] suggested a modification that allowed RED to *mark* IP packets instead of *dropping* them. The proposed architecture called for the Explicit Congestion Notification (ECN) bit in the IP header that would be set by routers in the packets they decided to mark and would be left untouched in unmarked packets. Upon the receipt of a packet with the ECN bit set, the receiver would communicate congestion status to the sender and expect the sender to

reduce its transmission rate. The paper suggested that TCP react to marked packets the same way it reacted to packet loss (i.e., by reducing congestion window *cwnd* and *ssthresh* by half), however, no more than one rate reduction per round-trip time was allowed. The paper studied an ECN-enabled TCP in a network of ECN routers and compared its performance with that of non-ECN TCPs in RED and Drop Tail networks. Floyd found that ECN TCPs maintained a much lower end-to-end delay and achieved the same or higher throughput than non-ECN TCPs in the same simulated network.

Currently, the IETF proposal for IP ECN is in an experimental stage [228]. In the proposal, the ECN is supported through reserved bits in the Differentiated Services (DS) [189] (formerly known as the ToS) field of the IP header. In addition to reserved bits in the IP header, the scheme utilizes reserved bits in the TCP header to allow a TCP receiver to inform the source about congestion. To avoid setting the IP CE (*Congestion Experienced*) bit in non-ECN flows and inherent unfairness, the proposal calls for another bit in the DS field (called *ECT – ECN-Capable Transport*) that would be set by all ECN sources in every packet they send. Thus, routers would use packet drop to mark packets from non-ECN flows (i.e., flows with ECT = 0) and would set the congestion bit to mark packets of ECN-enabled flows (i.e., flows with ECT = 1).

In another recent work, Floyd *et al.* [80] suggested that routers detect flows that do not respond to packet loss in a TCP-friendly fashion and penalize them in case of congestion. Among possible methods of identifying TCP-unfriendly flows, the paper proposed that routers apply one of the TCP-friendly formulas [166], [194] to each flow and verify that the flow was in fact responding to packet loss in a TCP-compatible way (i.e.,

the flow's sending rate was below or equal to the rate predicted by the model of TCP throughput). Since routers generally do not know the end-to-end packet loss or RTTs of individual flows, the paper developed a method for the routers to estimate the upper limit of each flow's sending rate using information local to the router.

Among router-based congestion control methods, we mention only a few. In 1992, Mishra *et al.* [175] developed a router-based, rate congestion control scheme, in which routers performed congestion management by exchanging control messages between each other, maintained a per-flow state for each network stream, and applied per-link rate limiting to each flow. In 1995, Kanakia *et al.* [119], suggested an adaptive router-based congestion control scheme for real-time video applications. The scheme provided explicit feedback to each source regarding the optimal sending rate. The sources would then adjust their real-time encoding rates according to a control law, which maintained a predefined queue length at the bottleneck router and resulted in a congestion-free network. Many similar methods are currently used in ATM.

2.3.4 RTP-based congestion control

Real-time Transport Protocol (RTP) [246] is an IETF standards-track protocol for real-time, rate-based data transport for unicast and multicast networks. RTP uses a control protocol called RTCP (Real-time Transport Control Protocol) to collect feedback from each receiver about their perceived quality of the network. Receiver feedback messages include packet loss, observed throughput, delay jitter, etc., and, in theory, are used by the source to adapt the sending rate in response to congestion. RTP/RTCP [246], how-

ever, does not specify how the sources should react to RTCP receiver reports, and the issue of proper rate adaptation in response to RTCP reports has become the center of RTP-related research.

In 1994 and later in 1996, Turletti *et al.* [264] and Bolot *et al.* [25] experimented with RTP-based congestion control in the context of the Internet. These studies developed a rate adaptation algorithm based on receiver-reported packet loss in a multicast session. If the packet loss was above a certain tolerable threshold, the streaming rate was reduced by half. However, if the packet loss was below the same threshold, the sender used an *exponential* rate increase to probe for new bandwidth. The resulting MIMD (Multiplicative Increase, Multiplicative Decrease) scheme does not converge to fairness in a general sense [49] and can be very unstable when used in a real network.

The authors redesigned their congestion control scheme in 1998, when Bolot *et al.* [26] proposed a true AIMD congestion control scheme on top of RTP. The conditions of the AIMD control decision have changed as well, allowing periods of neither increase nor decrease. The authors implemented the new schemes and found them to work well over the MBone.

In 1998, Sisalem *et al.* [255] used a packet-pair bandwidth estimation algorithm in conjunction with RTCP receiver reports to solve the problem of congestion control for multicast. Under the proposed scheme, the sender was instructed to send data in n-packet bursts and the receiver used each of the n–1 inter-packet gaps in each burst to estimate the bottleneck bandwidth. Before sending bandwidth samples back to the source, the receiver used Carter's filtering procedure [44] to arrive at a single good estimate. The re-

sulting estimates were sent back to the source through RTCP, and the sender used the information about packet loss and estimated bottleneck bandwidth in an AIMD-like rate adaptation algorithm.

2.3.5 TCP-friendly congestion control

In 1998, Cen *et al.* [45] developed a window-based congestion control that utilized a TCP-like rate adaptation scheme and proposed its use in multimedia streaming applications. The suggested scheme, called Streaming Control Protocol (SCP), borrowed the slow start algorithm, exponential retransmission timer backoff, and exponential weighted averaging of RTT samples from TCP. The sender continuously estimated the current network bandwidth from the receipt of ACKs (according to an undisclosed formula) and used this estimate in its congestion control. TCP's congestion avoidance algorithm was modified and involved keeping the sender's transmission window W equal to the bandwidth-delay product of the end-to-end path at all times. Finally, the computation of the RTO (retransmission timeout) was adopted from TCP; however, since TCP's RTO performed poorly in SCP, the authors multiplied it by a factor of 1.25.

In 1999, Rejaie *et al.* [239] proposed a TCP-friendly congestion control scheme called RAP (Rate Adaptation Protocol). RAP was a rate-based version of TCP for streaming. In RAP, the calculation of the RTT and timeouts was identical to that in TCP, ACKs were modified to include the location of the last gap in sequence numbers as detected by the receiver. The control part of RAP consisted of an AIMD rate-adjustment scheme, in which the decisions to change the rate were made once per RTT. Extensive

simulation in `ns2` showed that RAP was TCP-friendly across a wide range of experiments and achieved even better fairness with TCP in RED-enabled simulations.

Models of TCP throughput as a function of packet loss and round-trip delay [166], [196] have recently prompted several formula-based, rate congestion control schemes. Padhye *et. al*. [198] proposed a scheme called *TCP-friendly rate control protocol* (TFRCP), in which the sender adjusted its rate at every round-trip interval based on the information fed back by the receiver inside positive ACKs. The ACKs contained the sequence number of the packet they acknowledged and the status of the preceding eight packets (whether they had been received or not). Upon packet loss within an RTT interval, the sender recomputed the transmission rate based on the updated values of the RTT and packet loss. In the absence of lost packets with an RTT interval, the sender doubled its rate.

Floyd *et al.* in [86] proposed an improved version of the above protocol that was better suited for Internet-like environments and contained several important oscillation-preventing methods. The work adopted one of the formulas from [196], supplied it with slow start and a stable rate control function, and called the new scheme *TCP-friendly Rate Control* (TFRC) (discussed in more detail in [274]). The basis of TFRC was the *TCP-friendly rate equation* [196]:

$$T = \frac{MTU}{R\sqrt{\frac{2p}{3}} + t_{RTO} \cdot 3 \cdot \sqrt{\frac{3p}{8}} \cdot p \cdot \left(1 + 32p^2\right)} \tag{3}$$

where p was the current *loss event* rate, R was the current *smoothed* estimate of the RTT, MTU was the packet size, and t_{RTO} was the value of a TCP-like retransmission timeout. Furthermore, in TFRC, each sender packet was acknowledged by the receiver. In cases when the sender ceased receiving the ACKs, it slowly reduced its rate and eventually stopped. The latter condition prevented an open-loop streaming in cases when the receiver had died or was unreachable. Robust estimation of packet loss was the essence of the proposed protocol. In TFRC, the receiver with each arriving packet computed the *loss event* rate p based on the *average loss interval* s and fed back the value of p to the sender. The paper used a clever weighted smoothing of the values of loss interval s to achieve smoothness of the sending rate. Another aspect addressed in the paper was the prevention of rate oscillations due to frequently-changing RTTs, while maintaining quick response to congestion. To avoid unnecessary instability arising from instantaneous changes in the RTT, the paper proposed to set the inter-packet spacing for actual transmission of packets to:

$$t = \frac{MTU\sqrt{R_0}}{T \cdot M}, \tag{4}$$

where R_0 was the latest RTT sample, T was the latest TCP-friendly rate in bytes per second, and M was the weighted exponential average of the square-roots of the RTTs.

Finally, the paper supplied TFRC with slow start, similar to the one in TCP. In order to avoid large overshooting of the available bandwidth at the beginning of a session, the sender doubled its rate once per RTT according to a modified version of slow

start. At each RTT interval, the sender chose the minimum of the sending rate during the last RTT interval and the reported rate at which the packets arrived to the receiver during the same interval. The chosen minimum was then doubled to probe for new bandwidth. TFRC was extensively tested over the Internet and in two network simulators, was shown to perform with considerably less rate fluctuation than TCP, and was found to be TCP-friendly.

In 1998, Tan *et al.* [263] developed a 3D subband scalable video coding and used a formula-based rate control to stream video data over the Internet. The resulting scheme, called *TCP-friendly rate-based transport protocol* (TFRP), used the formula of TCP throughput from [166] to control the sending rate of the source and achieved a fair allocation of bandwidth with TCP during several Internet experiments.

2.3.6 Real-time video streaming protocols

In 1998, Bolot *et al.* [26] (and earlier in [25], [264]) used RTP for adapting the streaming rate of a video sender in a multicast session. The congestion control proposed in the paper was based on RTCP receiver reports and implemented an AIMD rate-adaptation scheme (for discussion, see section 2.3.4). However, the proposed video encoding method (H.261) did not envision any scalable video compression, and as a consequence, the method was applicable only to live video sources (i.e., the sources that could adapt the encoding rate in real time). For non-live video sources (i.e., stored video), the method used frame skipping and frame rate variation in response to network congestion.

During the experiments with a live video source shown in [26], the receiver's frame rate fluctuated between 0 and 25 frames per second (fps) in response to packet loss.

Rejaie *et al.* [236] discussed a method for a video application to manipulate coarse-granular video enhancement layers in response to network congestion in order to minimize the fluctuation of the perceived video quality at the receiver. The paper assumed a unicast session, a fixed number of equal-bandwidth enhancement layers, a linear quality-bandwidth dependence within each layer, and solved the problem of finding an optimal layer-manipulation scheme given arbitrarily-fluctuating available bandwidth in the network.

2.4 Bandwidth Measurement

In this section, we briefly describe each of the techniques that can be used to measure the bottleneck bandwidth of an end-to-end path and overview related work that utilizes these bandwidth estimation techniques. We discuss four bandwidth estimation methods, three of which are based on the *packet pair* principle (explained in section 2.4.1). The majority of work in this area uses off-line analysis and generates single (per-connection) bandwidth estimates using packet data traces from the entire connection.

2.4.1 Sender-based packet pair

Sender-based packet pair (SBPP) refers to the mechanism of estimating the bottleneck (or sometimes the available[4]) bandwidth from the spacing delay between the ACKs arriving to the sender. In the simplest form of SBPP, the sender transmits two back-to-back packets (*packet pair*) of size s_1 and s_2. Assuming there is a bottleneck link along the path to the receiver and assuming ideal conditions for both packets (i.e., no cross traffic), the packets will be buffered behind each other at the bottleneck link (Figure 2 adapted from [110]), since the second packet arrives before the link can fully transmit the first packet (if this is not true, then the link is not the bottleneck link by definition). After passing through the bottleneck link, the packets are spread out by the transmission time of the second packet over the bottleneck link, and the same spacing is ideally preserved by the ACKs. From the spacing between the ACKs, the sender is able to derive samples of the bottleneck bandwidth.

[4] In cases of WFQ (Weighted Fair Queueing) routers, for example.

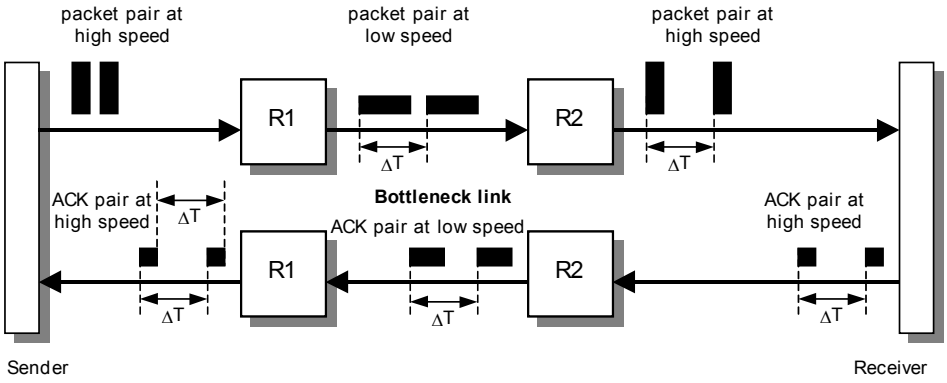

Figure 2. Sender Based Packet Pair.

Figure 2 shows a sender transmitting two packets back-to-back over the first high speed link. The vertical dimension of each packet represents the transmission speed of the link and the horizontal dimension is the transmission time. Once the packets arrive to the bottleneck link, their transmission duration increases (i.e., they become wider and vertically shorter). Assuming there is no other traffic, the bottleneck link spreads out the packets by the transmission time of the *second* packet over the bottleneck link (ΔT in the figure). For all practical purposes, we use the transmission time of the second packet instead of the first one, because the receiver is notified about the arrival of a packet when the packet's *last* bit is received by the NIC (network interface card). In Figure 2, the same spacing ΔT between packets is preserved along other links since they are no slower than the bottleneck link according to our assumption. Therefore, packets ideally arrive to the destination the same way they left the bottleneck link and the receiver generates ACKs at the times when the last bit of each packet is received. Since ACKs generally have the same size, spacing ΔT applies to the first bits of the ACKs as well as to the last bits as

shown in Figure 2. Upon the receipt of the ACKs, the sender is able to estimate ΔT and compute the bottleneck bandwidth as:

$$B_B = \frac{s_2}{\Delta T} \tag{5}$$

SBPP assumes no cross-traffic at the bottleneck link; however, in practice, packet from other flows typically queue between the packets in the packet pair and cause the samples of the bottleneck bandwidth to be inaccurate. Other limitations of SBPP are discussed in section 2.4.2.

In 1988, Jacobson [110] noticed the effect of bottleneck links on the spacing between ACKs and suggested that sender's self-clocking (derived from ACK spacing) allows a TCP connection to indirectly estimate the available network bandwidth. The paper, however, did not go as far as literally computing the bandwidth from ACK spacing.

In 1991, Keshav [121], [122], [123] proposed a rate-based packet-pair flow control mechanism for FQ (Fair Queuing)-enabled networks. Keshav applied stochastic modeling to FQ networks and used packet pair probing mechanism to estimate the current service rate at the bottleneck FQ router. In the proposed scheme, the source transmitted a packet pair at regular intervals, and the receiver acknowledged each packet as soon as it was received. Each pair of ACKs provided the sender with the current estimate of the service rate at the bottleneck router and the current estimate of the RTT. Since the bottleneck bandwidth in a FQ router depends on the number of concurrently running flows through the bottleneck router, the service rate μ at the router was not constant. Due

to arriving and departing flows, service rate at the bottleneck router at any time could either increase or decrease. Keshav modeled the change in the service rate as white Gaussian noise ω (i.e., $\mu(k+1) = \mu(k) + \omega(k)$) and derived control-theoretic laws for the sending rate $\lambda(k)$, where k represented discrete RTT intervals. The work analyzed the stability of the control law and derived an asymptotically stable rate adjustment scheme. Independently of the control law, the scheme needed a robust state estimator. Keshav examined the suitability of the Kalman filter (i.e., the minimum variance state estimator of a linear system) in solving the problem, but found it to be impractical since the Kalman filter required the knowledge of both observation and system noises. As an alternative, the work suggested a heuristic estimator based on fuzzy exponential averaging and showed that it did not require the knowledge of either of the noises. The proposed flow control scheme was found to work well in simulation, but was never implemented in the real Internet (due to the fact that FQ is not deployed in the current Internet routers).

In 1996, Carter *et al.* [44] took another look at the packet pair technique and investigated its applicability to the real Internet. The authors designed two tools for measuring the bottleneck (`bprobe`) and the available (`cprobe`) bandwidths. The bottleneck bandwidth estimation in `bprobe` utilized a 10-packet SBPP technique, which was applied to ICMP traffic. In this scheme, the sender used each of the nine inter-packet gaps to estimate the bottleneck bandwidth. Out of the nine estimates, the highest and lowest were dropped, and the resulting seven participated in the estimation process. Each 10-packet run was repeated with increasingly larger packet sizes (from 124 to 8,000 bytes per packet). After a seven-run measurement (ideally resulting in 49 samples of bandwidth), a

filtering technique was used to eliminate the outliers and derive a final estimate. Carter's filtering method involved looking for the area of the highest clustering of samples and locating the final estimate in that area using one of the two proposed heuristic methods. The second SBPP technique (implemented in `cprobe`) was designed to measure the available bandwidth of an end-to-end path by sending ten back-to-back ICMP echo packets and deriving a single bandwidth estimate from the arrival times of the first and the last ICMP echo replies. Assuming the difference between the arrival time of the first and the last ICMP echo reply messages was ΔT, the final estimate was computed by dividing the size of all ten packets by ΔT. Both `bprobe` and `cprobe` were tested in several LAN and Internet environments and were found to work well.

In 1997, Paxson [205], [209] collected extensive packet traces from a large number of TCP connections and utilized an off-line tool to investigate the effectiveness of SBPP as if it were used in real-time by each recorded TCP connection. Using collected TCP headers, Paxson studied pairs of back-to-back packets that suffered packet expansion along the path to the receiver. Analyzing ACK pairs generated by the receiver in response to expanded packet pairs, Paxson was able to estimate the bottleneck bandwidth as it could have been computed by the sender using SBPP. After applying several heuristics to the SBPP method in order to reject inaccurate samples, Paxson found that SBPP's estimates were within $\pm$ 20% of the capacity of the bottleneck link only for 60% of the connections.

2.4.2 Receiver-based packet pair

Receiver-based packet pair (RBPP) is an improvement over SBPP that does not allow ACKs to interfere with the measurement of bandwidth. In RBPP, the receiver computes the bottleneck bandwidth based on the spacing between packets in the packet pair and sends the estimates of the bottleneck bandwidth to the sender. This idea is illustrated in Figure 3.

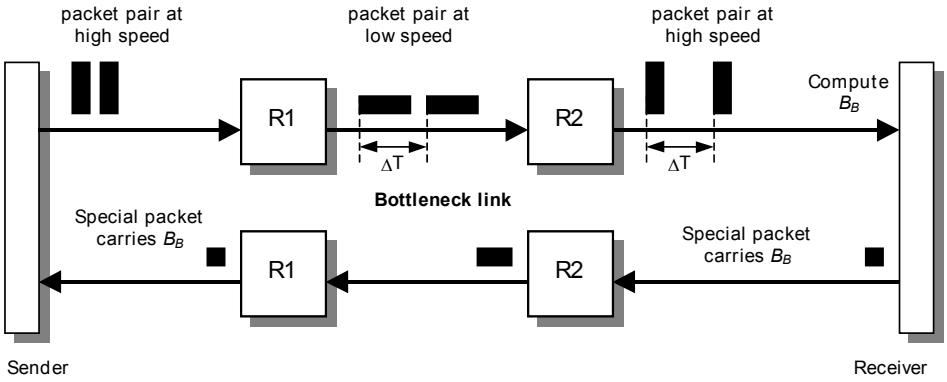

Figure 3. Receiver-based packet pair (RBPP).

In Figure 3, the receiver computes the bottleneck bandwidth B_B at the time it receives the packet pair. It then generates a special packet (which could be a regular ACK) with an embedded estimate B_B. RBPP is typically more accurate than SBPP for several reasons. First, in SBPP, the protocol overhead needed to generate the ACKs may skew their spacing and provide an inaccurate estimate to the sender. Some protocol implementations employ delayed ACKs (such as TCP), in which case instead of two ACKs the receiver sends only one. In addition, ACK spacing may be altered due to OS kernel scheduling delays or other protocol-independent reasons. Second, ACKs in SBPP have a good

chance of getting arbitrarily compressed or expanded on the way back to the sender due to possible heavy cross-traffic in the reverse direction. Therefore, SBPP is more likely to generate an inaccurate sample. Finally, if the bottleneck bandwidth along the reverse path from the receiver to the sender is significantly less than that in the forward direction, ACKs in SBPP can measure the *reverse* bottleneck bandwidth instead of the *forward* bottleneck bandwidth. Next, we mention several papers that studied RBPP.

In addition to experimenting with SBPP, Paxson [209], [205] analyzed the performance of RBPP using the same TCP packet traces. Paxson observed that, while SBPP measured the bandwidth within ± 20% of the correct value only for 60% of the connections, RBPP achieved similar accuracy for 98% of the connections. Paxson identified the main shortcoming of the RBPP method to be the necessity for the sender to include transmission timestamps in each packet of a packet pair. Such timestamps would be needed for the receiver to properly detect packet compression and avoid generating inaccurate estimates upon the receipt of compressed packet pairs. In addition, Paxson noticed that both SBPP and RBPP consistently failed over multi-channel bottleneck links (i.e., ISDN BRI).

In 1999, Lai *et al.* conducted a thorough review of various bandwidth estimation techniques in [132] and suggested a low-overhead version called Receiver-Only Packet Pair (ROPP). ROPP was based on RBPP, except the sender did not use timestamps to detect compressed packet pairs. Lai *et al.* further concluded that SBPP was quite inaccurate under a variety of conditions and focused on improving RBPP. In the proposed modification, called Measured Bandwidth Filtering (MBF), the paper suggested applying a mov-

ing interpolation kernel to the samples in order to average the collected bandwidth samples, suppress the outliers, and smooth the random errors generated by ROPP.

2.4.3 Packet Bunch Mode

In order to overcome the deficiencies of SBPP and RBPP, Paxson [4], [205], [209] proposed a novel approach to reliable estimation of the bottleneck bandwidth. The new scheme, called Packet Bunch Mode (PBM), was based on RBPP, but allowed the sender to employ more than two packets in a packet pair and allowed the packets to vary in size. Paxson applied PBM to the same traces of TCP connections and searched for multiple back-to-back packets that qualified for the packet pair technique. By default, PBM stopped at packet bunches of size four, and proceeded to larger bunches only if its heuristic bandwidth estimator did not converge by that time. PBM performed much better than SBPP or RBPP; however, a major part of PBM's operation was based on a number of heuristic rules that were formulated based on Paxson's data.

Recently, PBM received more attention from Ratnasamy *et al.* in [231]. The paper applied Paxson's PBM to packet pairs (i.e., only packet bunches of size two). The main theme of the paper was to derive a logical topology of a multicast tree by correlating receivers by their packet loss patterns. Once a logical multicast tree was constructed, the paper used PBM to narrow down the location and compute the speed of the bottleneck links in the constructed tree.

2.4.4 Pathchar

`Pathchar` (Path Characterization) is a method of estimating the bottleneck bandwidth (as well as latencies and queuing delays) of individual links along an Internet path. `Pathchar` started as a program written by Van Jacobson in 1997 [111]. `Pathchar` has not been given much thought until recently, when Downey [68] examined the performance of `pathchar` along two Internet paths and derived interesting methods of speeding up the convergence of the algorithm. `Pathchar` works similar to `traceroute` and exploits the TTL field of IP packets. Instead of sending one packet per hop, `pathchar` sends multiple packets of linearly increasing size to probe each router. Each of the *TTL expired* messages returned by a router provides an estimate of the round-trip time to that particular router. Using minimum filtering, `pathchar` keeps only samples of the RTT that (ideally) have not been affected by the cross traffic. In the presence of several simple assumptions about the path, estimates of the bottleneck bandwidth for each link can be derived from the linear increase slope in the minimum RTT as a function of probe size.

Unfortunately, achieving high accuracy using `pathchar` requires sending enormous amounts of traffic along the path under consideration. A thorough investigation of a path with a slow bottleneck link may up to last several hours. Downey in [68] utilized 64 probes per packet size and 45 different packet sizes from 120 to 1528 bytes. Under such testing conditions, `pathchar` required 2.8 Mbytes of data per *each hop*. Even given a relatively exhaustive examination of the path mentioned above, [68] found that `pathchar` measurements contained a substantial amount of inaccurate samples. In the end, Downey was able to reduce the number of messages sent to each hop by establishing a conver-

gence criteria for each estimate; however, he was unable to significantly improve the accuracy of the algorithm.

PART I

Performance Study of Real-time Streaming in the Internet

Chapter Three

3 Performance of Internet Streaming: Statistical Analysis

In this chapter, we analyze the results of a seven-month real-time streaming experiment, which was conducted between a number of unicast dialup clients, connecting to the Internet through access points in more than 600 major U.S. cities, and a backbone video server. During the experiment, the clients streamed low-bitrate MPEG-4 video sequences from the server over paths with more than 5,000 distinct Internet routers. We describe the methodology of the experiment, the architecture of our NACK-based streaming application, study end-to-end dynamics of 16 thousand ten-minute sessions (85 million packets), and analyze the behavior of the following network parameters: packet loss, round-trip delay, one-way delay jitter, packet reordering, and path asymmetry. We also study the impact of these parameters on the quality of real-time streaming.

3.1 Introduction

The Internet has become a complex interconnection of a large number of computer networks. The behavior of the Internet has been the target of numerous studies, but nevertheless, the performance of the Internet from the perspective of an average home user still remains relatively undocumented. At the same time, we believe that since end users are responsible for a large fraction of Internet traffic, the study of network conditions experienced by these users is an important research topic. This is the reason that compelled us to conduct a fundamentally different performance study that looks at Internet dynamics from the angle of an average Internet user.

Even though the Internet has been extensively analyzed in the past, an overwhelming majority of previous studies were based on TCP or ICMP traffic. On the other hand, real-time streaming protocols have not received as much attention in these studies. In fact, the dynamics of UDP NACK-based protocols (not necessarily real-time) are still not understood very well in the Internet community. As an illustration, a recent study [159] found that the widely accepted TCP retransmission timeout (RTO) estimator [4], [110] was not necessarily an optimal choice for low-bitrate NACK-based protocols employed over the Internet.

The novelty of our study is emphasized by the fact that no previous work attempted to characterize the performance of real-time streaming in a large-scale experiment involving low-bitrate Internet paths. The Internet has been studied from the perspective of TCP connections by Paxson [209], Bolliger *et al.* [19], Caceres *et al.* [40],

Mogul [178], and several others (e.g., [11]). Paxson's study included 35 geographically distributed sites in 9 countries; Bolliger *et al*. employed 11 sites in 7 countries and compared the throughput performance of various implementations of TCP during a six-month experiment; whereas the majority of other researchers monitored transit TCP traffic at a single backbone router [11], [178] or inside several campus networks [40] for the duration ranging from several hours to several days.

The methodology used in both large-scale TCP experiments [19], [209] was similar and involved a topology where each participating site was paired with every other participating site for an FTP-like transfer. Although this setup approximates well the current use of TCP in the Internet, future entertainment-oriented streaming services, however, are more likely to involve a small number of backbone video servers and a large number of home users.[5]

We believe that in order to study the current dynamics of real-time streaming in the Internet, we must take the same steps to connect to the Internet as an average end-user[6] (i.e., through dialup ISPs). For example, ISPs often experience congestion in their own backbones, and during busy hours, V.90 modems in certain access points are not available due to high user demand, none of which can be captured by studying the Internet from a small campus network directly connected to the Internet backbone.

---

[5] Our work focuses on non-interactive streaming applications where the user can tolerate short (i.e., in the order of several seconds) startup delays (e.g., TV over the Internet).

[6] Recent market research reports (e.g., [108]) show that in Q2 of 2001, approximately 89% of US households used dialup access to connect to the Internet. Furthermore, it is predicted [107], [217] that even in

In addition to choosing a different topological setup for the experiment, our work is different from the previous studies in several other aspects. First, the sending rate of a TCP connection is driven by its congestion control, which can often cause increased packet loss and higher end-to-end delays in the path along which it operates (e.g., during slow start). In our experiment, we measured *true* end-to-end path dynamics without the bias of congestion control applied to slow modem links.[7] Furthermore, our decision not to use congestion control was influenced by the evidence that the majority of streaming traffic in the current Internet employs constant-bitrate (CBR) video streams [232], where the user explicitly selects the desired streaming rate from the corresponding web page (note that the additional rate adaptation implemented in [232] is very rudimentary and could hardly be considered congestion control).

Second, TCP uses a positive ACK retransmission scheme, whereas current real-time applications (such as [232]) employ NACK-based retransmission to reduce the amount of traffic from the users to the streaming server. As a consequence, end-to-end path dynamics perceived by a NACK-based protocol could differ from those sampled by TCP along the same path: real-time applications acquire samples of the round-trip delay (RTT) at rare intervals, send significantly less data along the path from the receiver to the sender, and bypass certain aspects of TCP's retransmission scheme (such as exponential timer backoff). Previous work [159] suggests that NACK-based retransmission schemes

---

2005, the majority of US households will still be using dialup modems, and it is unclear when broadband penetration in the US will reach 50% of households.

may require a different retransmission timeout (RTO) estimator and leads us to believe that research in this area should be extended.

Finally, TCP relies on window-based flow control, and real-time applications usually utilize rate-based flow control. In many video-coding schemes, a real-time streaming server must maintain a certain target streaming (i.e., sending) rate for the decoder to avoid *underflow events*, which are produced by packets arriving after their decoding deadlines. As a result, a real-time sender may operate at different levels of packet burstiness and instantaneous sending rate than a TCP sender, because the sending rate of a TCP connection is governed by the arrival of positive ACKs from the receiver rather than by the application.

We should further mention that the Internet has been extensively studied by various researchers using ICMP `ping` and `traceroute` packets [1], [53], [54], [55], [182], [209], UDP echo packets [22], [29], [30], and multicast backbone (MBone) audio packets [282], [283]. With the exception of the last one, similar observations apply to these studies – neither the setup, nor the type of probe traffic represented realistic real-time streaming scenarios. In addition, among the studies that specifically sent audio/video traffic over the Internet [25], [26], [63], [62], [247], [263], [264] the majority of experiments involved only a few Internet paths, lasted for a very short period of time, and focused on

---

[7] Without a doubt, future real-time streaming protocols will include some form of scalable congestion control; however, at the time of the experiment, it was not even clear which methods represented such congestion control.

analyzing the features of the proposed scheme rather than the impact of Internet conditions on real-time streaming.

In this chapter, we present the methodology and analyze the results of a seven-month large-scale real-time streaming experiment, which involved three nation-wide dialup ISPs, each with several million active subscribers in the United States. The topology of the experiment consisted of a backbone video server streaming MPEG-4 video sequences to unicast home users located in more than 600 major U.S. cities. The streaming was performed in real-time (i.e., with a real-time decoder), utilized UDP for the transport of all messages, and relied on simple NACK-based retransmission to recover lost packets before their decoding deadlines.

Even though we consider it novel and unique in many aspects, there are two limitations to our study. First, our experiments document Internet path dynamics perceived by low-bitrate (i.e., modem-speed) streaming sessions. Recall that one of the goals of our work was to conduct a performance study of the Internet from the angle of a typical home Internet user, and to this extent, we consider our work to be both thorough and successful. In addition, by focusing on low-bitrate paths, our study shows the performance of real-time protocols under the most difficult network conditions (i.e., large end-to-end delays, relatively high bit-error rates, low available bandwidth, etc.) and provides a “lower bound” on the performance of future Internet streaming applications. In addition, note that some of the emerging wireless data services with packet video capabilities (such as 3G or GPRS) share certain end-to-end characteristics (e.g., large delays and high error

rates) with the current dialup technologies, and the results of this chapter may be relevant to their design.

Second, during the experiment, the server did not adapt the streaming rate to the available bandwidth. Our study explicitly considers rate-based congestion control to be beyond the scope of the chapter (as discussed above) and attempts to sample the network parameters of the Internet with the minimum influence on the congestion already present in the network.

Despite these limitations, we believe that the results of our study conclusively establish the feasibility of video streaming in the currently best-effort Internet, show that retransmission is an effective method of recovering lost packets even for real-time traffic, and provide a valuable insight into dynamics of real-time streaming from the perspective of an average Internet user.

The remainder of the chapter is organized as follows. Section 3.2 describes the methodology of the experiment and section 3.3 discusses end-to-end path dynamics observed by our application.

3.2 Methodology

3.2.1 Setup for the Experiment

We started our work by attaching a Unix video server to the UUNET backbone via a T1 link (Figure 4). To support the client's connectivity to the Internet, we selected

three major nation-wide dialup ISPs (which we call ISP_a, ISP_b, and ISP_c)[8], each with at least five hundred V.90 (i.e., 56 kb/s) dialup numbers in the U.S., and designed an experiment in which hypothetical Internet users dialed a local access number to reach the Internet (through one of our three ISPs) and streamed video sequences from the server. Although the clients were physically placed in our lab in the state of New York, they dialed long-distance phone numbers and connected to the Internet through ISPs' access points located in each of the 50 states. Our database of phone numbers included 1813 different V.90 access points in 1188 major U.S. cities.

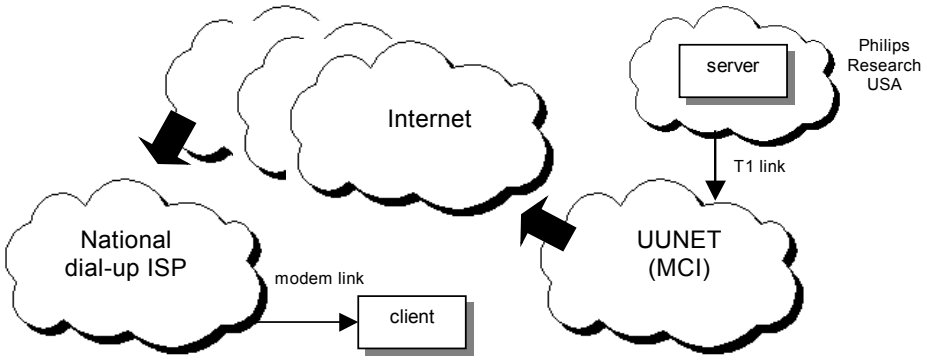

Figure 4. Setup of the experiment.

After the phone database was in place, we designed and implemented special software, which we call the *dialer*, that dialed phone numbers from the database, connected to the ISPs using the point-to-point protocol (PPP), issued a *parallel traceroute* to the server, and upon success, started the video client with the instructions to stream a ten-minute video sequence from the server. Our implementation of traceroute (built into the

[8] The corresponding ISPs were AT&T WorldNet, Earthlink, and IBM Global Network.

dialer) used ICMP probes, sent all probes in parallel instead of sequentially (hence the name "parallel"), and recorded the IP time-to-live (TTL)[9] field of each returned "TTL expired" message. The use of ICMP packets and parallel traceroute facilitated much quicker discovery of routers, and the analysis of the TTL field in the returned packets allowed the dialer to compute the number of hops in the reverse path from each intermediate router to the client machine (using a simple fact that each router reset the TTL field of each generated "TTL expired" packet to the value of the *initial TTL*[10]). Using the information about the number of *forward* and *reverse* hops for each router, the dialer was able to detect asymmetric end-to-end paths, which we study in section 3.3.6.

In our analysis of the data, we attempted to isolate clearly modem-related pathologies (such as packet loss caused by a poor connection over the modem link and large RTTs due to data-link retransmission) from those caused by congested routers of the Internet. Thus, connections that were unable to complete a traceroute to the server, connections with high bit-error rates (BER), and connections during which the modem could not sustain our streaming rates were all considered useless for our study and were excluded from the analysis in this chapter.

In practice, to avoid studying connections with clearly insufficient end-to-end bandwidth and various modem-related problems, we utilized the following methodology.

---

[9] Recall that the TTL field is decremented by 1 every time a packet is forwarded by a level-3 (i.e., IP-level) device.

[10] The majority of routers used the initial TTL equal to 255, while some initialized the field to 30, 64, or 128. Subtracting the received TTL from the initial TTL produced the number of hops along the reverse path.

We defined a streaming attempt through a particular access number to be *successful*, if the ISP's access number was able to sustain the transmission of our video stream for its entire length at the stream's target IP bitrate r. To be specific, the video client terminated connections in which the *aggregate* (i.e., counting from the very beginning of a session) packet loss grew beyond a certain threshold β_p or the *aggregate* incoming bitrate dropped below another threshold β_r. The experiments reported in this chapter used β_p equal to 15% and β_r equal to $0.9r$, both of which were experimentally found to be necessary conditions for efficient filtering out of modem-related failures. The packet-loss threshold was activated after 1 minute of streaming and the bitrate threshold after 2 minutes to make sure that slight fluctuations in packet loss and incoming bitrate at the beginning of a session were not mistaken for poor connection quality. After a session was over, the success or failure of the session was communicated from the video client to the dialer, the latter of which kept track of the time of day and the phone number that either passed or failed the streaming test.

In order to make the experiment reasonably short, we considered all phone numbers from the same state to be equivalent, and consequently, we assumed that a successful streaming attempt through any phone number of a state indicated a *successful coverage* of the state regardless of which phone number was used. Furthermore, we divided each 7-day week into 56 three-hour timeslots (i.e., 8 timeslots per day) and designed the dialer to select phone numbers from the database in such order so that each state would be *successfully covered* within each of the 56 timeslots at least once. In other words, each

ISP needed to sustain exactly 50·56 = 2,800 successful sessions before the experiment was allowed to end.

3.2.2 Real-time Streaming

For the purpose of the experiment, we used an MPEG-4 encoder to create two ten-minute QCIF (176x144) video streams coded at 5 frames per second (fps). The first stream, which we call S_1, was coded at the video bitrate of 14 kb/s, and the second steam, which we call S_2, was coded at 25 kb/s. The experiment with stream S_1 lasted during November – December 1999 and the one with stream S_2 was an immediate follow-up during January – May 2000.

During the transmission of each video stream, the server split it into 576-byte IP packets. Video frames always started on a packet boundary, and consequently, the last packet in each frame was allowed to be smaller than others (in fact, many P (prediction-coded) frames were smaller than the maximum payload size and were carried in a single UDP packet). As a consequence of packetization overhead, the *IP bitrates* (i.e., including IP, UDP, and our special 8-byte headers) for streams S_1 and S_2 were 16.0 and 27.4 kb/s, respectively. The statistics of each stream are summarized in Table I.

Stream	Size, MB	Packets	Video bi-trate, kb/s	Average frame size, bytes
S_1	1.05	4,188	14.0	350
S_2	1.87	5,016	25.0	623

Table I. Summary of streams statistics.

In our streaming experiment, the term *real-time* refers to the fact that the video decoder was running in real-time. Recall that each compressed video frame has a specific *decoding deadline*, which is usually based on the time of the frame's encoding. If a compressed video frame is not fully received by the decoder buffer at the time of its deadline, the video frame is discarded and an underflow event is registered. Moreover, to simplify the analysis of the results, we implemented a *strict* real-time decoder model, in which the playback of the arriving frames continued at the encoder-specified deadlines regardless of the number of underflow events (i.e., the decoding deadlines were not adjusted based on network conditions). Note that in practice, better results can be achieved by allowing the decoder to freeze the display and re-buffer a certain number of frames when underflow events become frequent (e.g., as done in [232]).

In addition, many CBR (constant bitrate) video coding schemes include the notion of the *ideal startup delay* [225], [232] (the delay is called "ideal" because it assumes a network with no packet loss and a constant end-to-end delay). This ideal delay must always be applied to the decoder buffer before the decoding process may begin. The ideal startup delay is independent of the network conditions and solely depends on the decisions made by the encoder during the encoding process.[11] On top of this ideal startup delay, the client in a streaming session usually must apply an *additional* startup delay in order to compensate for delay jitter (i.e., variation in the one-way delay) and permit the recovery of lost packets via retransmission. This additional startup delay is called the *de-*

lay budget (D_{budget}) and reflects the values of the *expected* (at the beginning of a session) delay jitter and round-trip delay during the length of the session. Note that in the context of Internet streaming, it is common to call D_{budget} simply "startup delay" and to completely ignore the *ideal* startup delay (e.g., [63]). From this point on, we will use the same convention. In all our experiments, we used D_{budget} equal to 2,700 ms, which was manually selected based on preliminary testing. Consequently, the total startup delay (observed by an end-user) at the beginning of each session was approximately 4 seconds.

3.2.3 Client-Server Architecture

For the purpose of our experiment, we implemented a client-server architecture for MPEG-4 streaming over the Internet. The server was fully multithreaded to ensure that the transmission of packetized video was performed at the target IP bitrate of each streaming session and to provide quick response to clients' NACK requests. The streaming was implemented in bursts of packets (with the burst duration D_b varying between 340 and 500 ms depending on the bitrate) for the purposes of making the server as low-overhead as possible (for example, RealAudio servers use D_b = 1,800 ms [173]). Although we agree that in many cases the desired way of sending constant bitrate (CBR) traffic is to equally space packets during transmission, there are practical limitations (such as OS scheduling and inter-process switching delays) that often do not allow us to follow this model.

---

[11] We will not elaborate further on the ideal startup delay, except mention that it was approximately 1,300 ms for each stream.

The second and the more involved part of our architecture, the client, was designed to recover lost packets through NACK-based retransmission and collect extensive statistics about each received packet and each decoded frame. Furthermore, as it is often done in NACK-based protocols, the client was in charge of collecting round-trip delay (RTT) samples.[12] The measurement of the RTT involved the following two methods. In the first method, each successfully recovered packet provided a sample of the RTT (i.e., the RTT was the duration between sending a NACK and receiving the corresponding retransmission). In our experiment, in order to avoid the ambiguity of which retransmission of the same packet actually returned to the client, the header of each NACK request and each retransmitted packet contained an extra field specifying the retransmission number of the packet.

The second method of measuring the RTT was used by the client to obtain *additional* samples of the round-trip delay in cases when network packet loss was too low. The method involved periodically sending *simulated* retransmission requests to the server if packet loss was below a certain threshold. In response to these simulated NACKs, the server included the usual overhead[13] of fetching the needed packets from the storage and sending them to the client. In our experiment, the client activated simulated NACKs, spaced 30 seconds apart, if packet loss was below 1%.

---

[12] The resolution of the timestamps was 100 microseconds.

[13] The server overhead was below 10 ms for all retransmitted packets and did not have a major impact on our characterization of the RTT process later in this chapter.

We tested the software and the concept of a wide-scale experiment of this sort for nine months before we felt comfortable with the setup, the reliability of the software, and the exhaustiveness of the collected statistics. In addition to extensive testing of the prototype for nine months, we monitored various statistics reported by the clients in real-time (i.e., on the screen) during the experiments for sanity and consistency with previous tests. Overall, the work reported in this chapter took us 16 months to complete (9 months testing and 7 months collecting the data).

Our traces consist of six datasets, each collected by a different machine. Throughout this chapter, we will use notation D_n^x to refer to the dataset collected by the client assigned to ISP_x ($x = a, b, c$) during the experiment with stream S_n ($n = 1, 2$). Furthermore, we will use notation D_n to refer to the combined set $\{D_n^a \cup D_n^b \cup D_n^c\}$.

3.3 Experiment

3.3.1 Overview

In dataset D_1, the three clients performed 16,783 long-distance connections to the ISPs' remote modems and successfully completed 8,429 streaming sessions.[14] In D_2, the clients performed 17,465 modem connections and sustained 8,423 successful sessions. Analysis of the above numbers suggests that in order to receive real-time streaming material with a minimum quality at 16 to 27.4 kb/s, an average U.S. end-user, equipped with a V.90 modem, needs to make approximately two dialing attempts to the ISPs' phone num-

numbers within the state where the user resides. The success rate of streaming sessions during different times of the day is illustrated in Figure 5. Note the dip by a factor of two between the best and the worst times of the day.

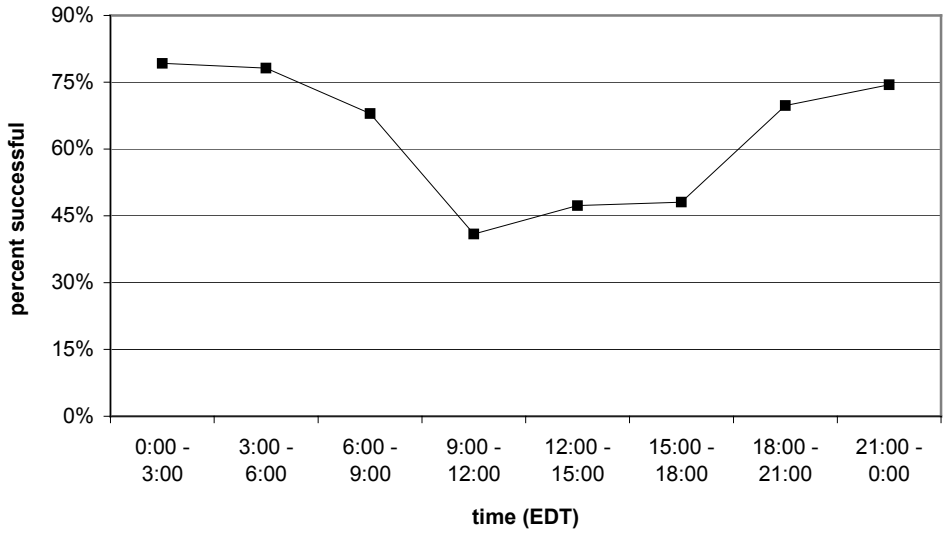

Figure 5. Success of streaming attempts during the day.

Furthermore, in dataset D_1, the clients traced the arrival of 37.7 million packets, and in D_2, the arrival of additional 47.3 million (for a total of 85 million). In terms of bytes, the first experiment transported 9.4 GBytes of video data and the second one transported another 17.7 GBytes (for a total of 27.1 GBytes).

Recall that each experiment lasted as long as it was needed to cover the entire United States. Depending on the success rate within each state, the access points used in the experiment comprised a subset of our database. In D_1, the experiment covered 962

---

[14] Typical reasons for failing a session were PPP-layer connection problems, inability to reach the server (i.e., failed traceroute), high bit-error rates, and low (14.4-19.2 kb/s) connection rates.

dialup points in 637 U.S. cities, and in D_2, it covered 880 dialup points in 575 U.S. cities. Figure 6 shows the combined (i.e., including both datasets D_1 and D_2) number of distinct cities in each state covered by our experiment (1,003 access points in 653 cities).

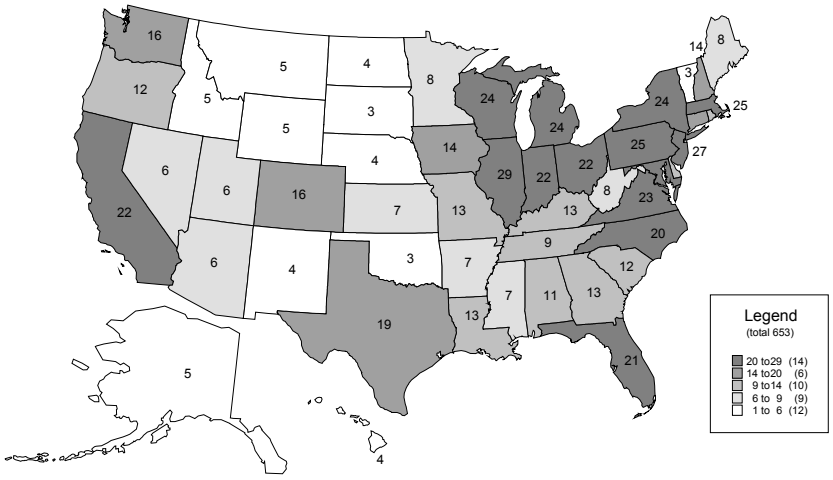

Figure 6. The number of cities per state that participated in either D_1 or D_2.

During the experiment, each session was preceded by a parallel traceroute, which recorded the IP addresses of all discovered routers (DNS and WHOIS[15] lookups were done off-line after the experiments were over). The average time needed to trace an end-to-end path was 1,731 ms, 90% of the paths were traced under 2.5 seconds, and 98% under 5 seconds. Dataset D_1 recorded 3,822 distinct Internet routers, D_2 recorded 4,449 distinct routers, and both experiments combined produced the IP addresses of 5,266 unique routers. The majority of the discovered routers belonged to the ISPs' networks (51%) and UUNET (45%), which confirmed our intuition that all three ISPs had direct peering con-

[15] The WHOIS database was used to discover the Autonomous System (AS) of each router.

nections with UUNET. Moreover, our traces recorded approximately 200 routers that belonged to five additional Autonomous Systems (AS).

The average end-to-end hop count was 11.3 in D_1 (6 minimum and 17 maximum) and 11.9 in D_2 (6 minimum and 22 maximum). Figure 7 shows the distribution of the number of hops in the encountered end-to-end paths in each of D_1 and D_2. As the figure shows, the majority of paths (75% in D_1 and 65% in D_2) contained between 10 and 13 hops.

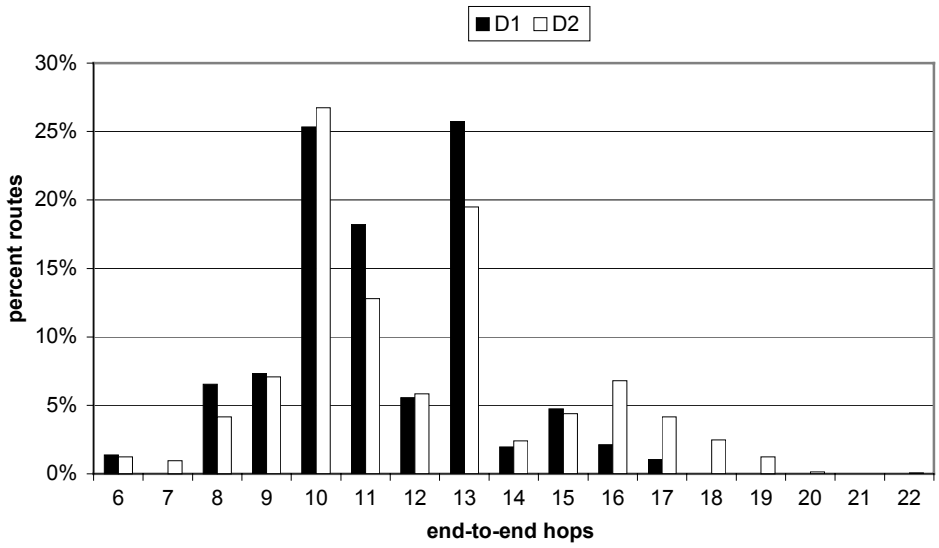

Figure 7. Distribution of the number of end-to-end hops.

Throughout the rest of the chapter, we restrict ourselves to studying only *successful* (as defined in section 3.2.1) sessions in both D_1 and D_2. We call these new purged datasets (with only successful sessions) D_{1p} and D_{2p}, respectively (purged datasets D_{np}^{x} are defined similarly for $n = 1, 2$ and $x = a, b, c$). Recall that $\{D_{1p} \cup D_{2p}\}$ contains 16,852

successful sessions, which are responsible for 90% of the bytes and packets, 73% of the routers, and 74% of the U.S. cities recorded in $\{D_1 \cup D_2\}$.

3.3.2 Packet Loss

3.3.2.1 Overview

Numerous researchers have studied Internet packet loss, and due to the enormous diversity of the Internet, only few studies agree on the average packet loss rate and the average *loss burst length* (i.e., the number of packets lost in a row). Among numerous studies, the average Internet packet loss was reported to vary between 11% and 23% by Bolot [22] depending on the inter-transmission spacing between packets, between 0.36% and 3.54% by Borella *et al*. [29], [30] depending on the studied path, between 1.38% and 11% by Yajnik *et al*. [283] depending on the location of the MBone receiver, and between 2.7% and 5.2% by Paxson [209] depending on the year of the experiment. In addition, 0.49% average packet loss rate was recently reported by Balakrishnan *et al*. [11], who analyzed the dynamics of a large number of TCP web sessions at a busy Internet server.

In dataset D_{1p}, the average recorded packet loss rate was 0.53% and in D_{2p}, it was 0.58%. Even though these rates are much lower[16] than those traditionally reported by Internet researchers during the last decade, they are still somewhat higher than those re-

[16] Note that during the experiment, simply dialing a different access number in most cases fixed the problem of high packet loss. This fact shows that the majority of failed sessions documented pathologies created by the modem (or the access point) rather than the actual packet loss in the Internet. Since an end-user typically would re-dial a bad connection searching for better network conditions, we believe that the bias created by removing failed sessions reflects the actions of an average Internet user.

ported by backbone ISPs [265]. Furthermore, 38% of the sessions in $\{D_{1p} \cup D_{2p}\}$ did not experience any packet loss, 75% experienced loss rates below 0.3%, and 91% experienced loss rates below 2%. On the other hand, 2% of the sessions suffered packet loss rates 6% or higher.

In addition, as we expected, average packet loss rates exhibited a wide variation during the day. Figure 8 shows the evolution of loss rates as a function of the timeslot (i.e., the time of day), where each point represents the average of approximately 1,000 sessions. As the figure shows, the variation in loss rates between the best (3-6 am) and the worst (3-6 pm) times of the day was more than by a factor of three. The apparent discontinuity between timeslots 7 (21:00-0:00) and 0 (0:00-3:00) is due to a coarse timescale in Figure 8. On finer timescales (e.g., minutes), loss rates converge to a common value near midnight. A similar discontinuity in packet loss rates was reported by Paxson [209] for North American sites, where packet loss during timeslot 7 was approximately twice as high as that during timeslot 0.

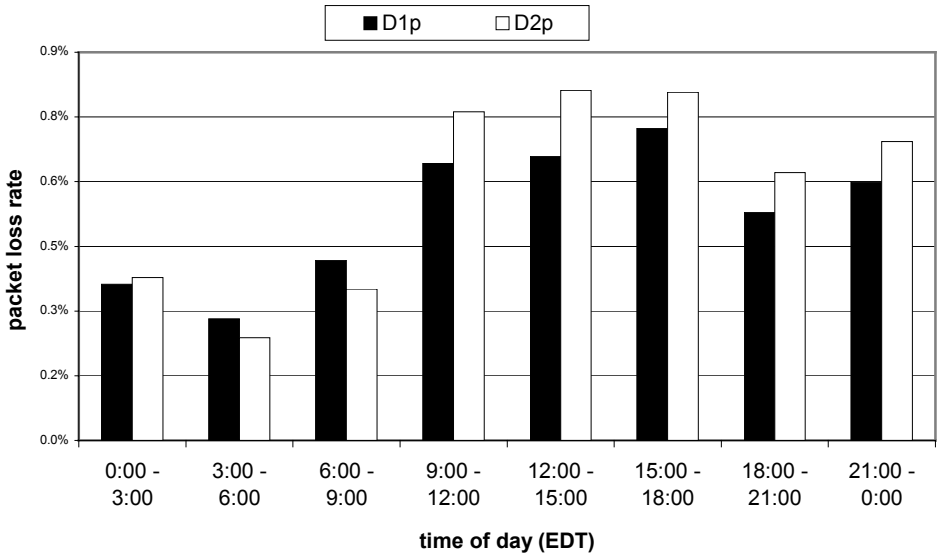

Figure 8. Average packet loss rates during the day.

The variation in the average *per-state* packet loss (as shown in Figure 9) was quite substantial (from 0.2% in Idaho to 1.4% in Oklahoma), but virtually did not depend on the state's average number of end-to-end hops (correlation coefficient ρ was –0.04) or the state's average RTT (correlation –0.16). However, as we will see later, the average per-state RTT and the number of end-to-end hops were in fact positively correlated.

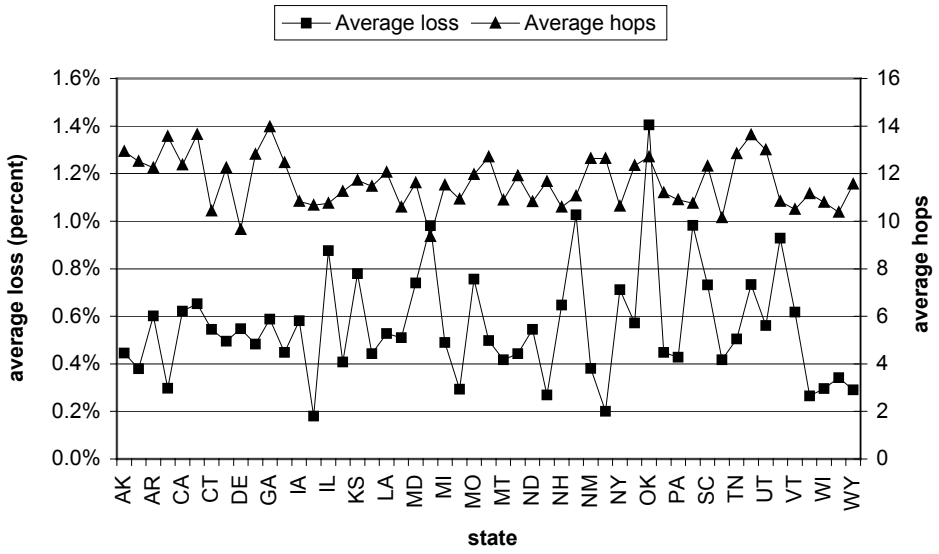

Figure 9. Average per-state packet loss rates.

3.3.2.2 Loss Burst Lengths

We next attempt to answer the question of how bursty Internet packet loss was during the experiment. Figure 10 shows the distribution (both the PDF and the CDF) of loss burst lengths in $\{D_{1p} \cup D_{2p}\}$ (without loss of generality, the figure stops at burst length 20, covering more than 99% of the bursts). Even though the upper tail of the distribution had very few samples, it was fairly long and reached burst lengths of over 100 packets.

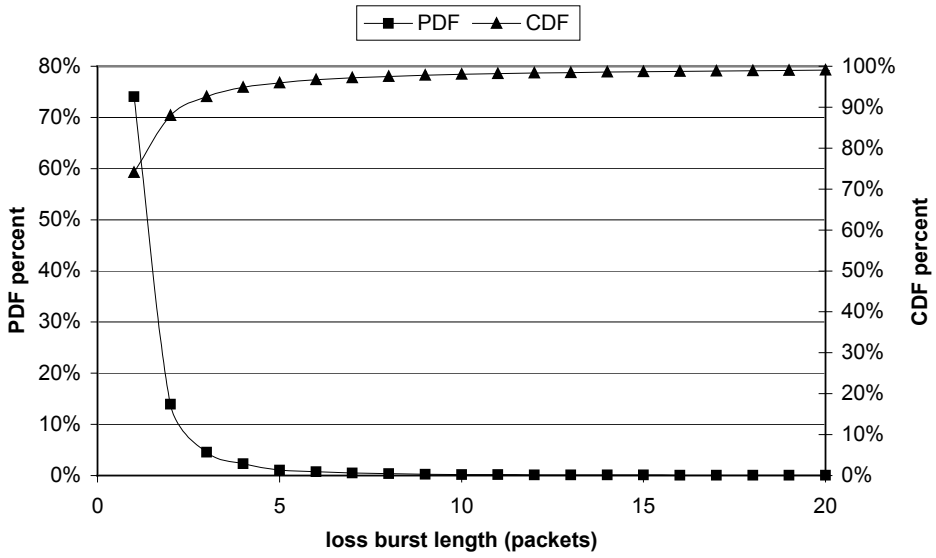

Figure 10. PDF and CDF functions of loss burst lengths in $\{D_{1p} \cup D_{2p}\}$.

Figure 10 is based on 207,384 loss bursts and 431,501 lost packets. The prevalence of single packet losses, given the fact that packets in our experiment were injected into the Internet in bursts at the T1 speed[17], leads us to speculate that either router queues sampled in our experiment overflowed on timescales smaller than the time needed to transmit a single IP packet over a T1 link (i.e., 3 ms for the largest packets and 1.3 ms for the average-size packets), or that backbone routers employed Random Early Detection (RED) for preventing congestion. However, a second look at the situation reveals that server's possible interleaving of packets from different sessions could have expanded the inter-packet transmission distance of each flow by up to a factor of 3. Furthermore, before each lost packet reached the corresponding congested router, packets from other

Internet flows could have queued immediately behind the lost packet, effectively expanding the inter-packet distance even further. Therefore, our initial speculation about the duration of buffer overflows during the experiment may not hold in all cases.

On the other hand, to investigate the presence of RED in the Internet, we contacted several backbone and dialup ISPs whose routers were recorded in our trace data and asked them to comment on the deployment of RED in their backbones. Among the ISPs that responded to our request, the majority had purposely disabled RED and the rest were running RED only for select customers at border routers, but not on the public backbone. Consequently, we conclude that even though the analysis of our datasets points towards transient (i.e., 1-3 ms) buffer overflows in the Internet routers sampled by our experiment, we cannot reliably determine the exact duration of these events.

In addition, we should note that single packet losses were under-represented in our traces, because packets that were lost in bursts longer than one packet could have been dropped by *different* routers along the path from the server to the client. Therefore, using end-to-end measurements, an application cannot distinguish between n ($n \geq 2$) single-packet losses at n different routers from an n-packet bursty loss at a single router. Both types of loss events would appear identical to an end-to-end application, even though the underlying cause is quite different.

---

[17] The server was only involved in low-bitrate streaming for our clients and did not have a problem blasting bursts of packets at the full speed of the adjacent link (i.e., 10 mb/s). The spacing between packets was further expanded by the T1 link to UUNET.

Furthermore, as previously pointed out by many researchers, the upper tail of loss burst lengths usually contains a substantial percentage of all lost packets. In each of D_{1p} and D_{2p}, single-packet bursts contained only 36% of all lost packets, bursts two packets or shorter contained 49%, bursts 10 packets or shorter contained 68%, and bursts 30 packets or shorter contained 82%. At the same time, 13% of all lost packets were dropped in bursts at least 50 packets long.

Traditionally, the burstiness of packet loss is measured by the average loss burst length. In the first dataset (D_{1p}), the average burst length was 2.04 packets. In the second dataset (D_{2p}), the average burst length was slightly higher (2.10), but not high enough to conclude that the higher bitrate of stream S_2 was clearly responsible for burstier packet loss. Furthermore, the *conditional probability* of packet loss, given that the previous packet was also lost, was 51% in D_{1p} and 53% in D_{2p}. These numbers are consistent with those previously reported in the literature. Bolot [22] observed the conditional probability of packet loss to range from 18% to 60% depending on inter-packet spacing during transmission, Borella *et al*. [30] from 10% to 35% depending on the time of day, and Paxson [209] reported 50% conditional probability for *loaded* (i.e., queued behind the previous) TCP packets and 25% for *unloaded* packets. Using Paxson's terminology, the majority of our packets were *loaded* since the server sent packets in bursts at a rate higher than the bottleneck link's capacity.

3.3.2.3 Loss Burst Durations

To a large degree, the average loss burst length depends on how closely the packets are spaced during transmission. Assuming that bursty packet loss comes from buffer overflow events in drop-tail queues rather than from consecutive hits by RED or from bit-level corruption, it is clear that all packets of a flow passing through an overflown router queue will be dropped for the duration of the instantaneous congestion. Hence, the closer together the flow's packets arrive to the router, the more packets will be dropped during each queue overflow. This fact was clearly demonstrated in Bolot's experiments [22], where UDP packets spaced 8 ms apart suffered larger loss burst lengths (mean 2.5 packets) than packets spaced 500 ms apart (mean 1.1 packets). Yajnik *et al*. [283] reported a similar correlation between loss burst lengths and the distance between packets. Consequently, instead of analyzing burst lengths, one might consider analyzing burst durations since the latter does not depend on inter-packet spacing during transmission.

Using our traces, we can only infer an approximate duration of each loss burst, because we do not know the *exact* time when the lost packets were supposed to arrive to the client. Hence, for each loss event, we define the *loss burst duration* as the time elapsed between the receipt of the packet immediately preceding the loss burst and the packet immediately following the loss burst. Figure 11 shows the distribution (CDF) of loss burst durations in seconds. Although the distribution tail is quite long (up to 36 seconds), the majority (more than 98%) of loss burst durations in both datasets D_{1p} and D_{2p} fall under 1 second. Paxson's study [209] also observed large loss burst durations (up to 50 seconds), however, only 60% of the loss bursts studied by Paxson were contained be-

low 1 second. In addition, our traces showed that the average distance between lost packets in the experiment was 172-188 good packets, or 21-27 seconds, depending on the streaming rate.

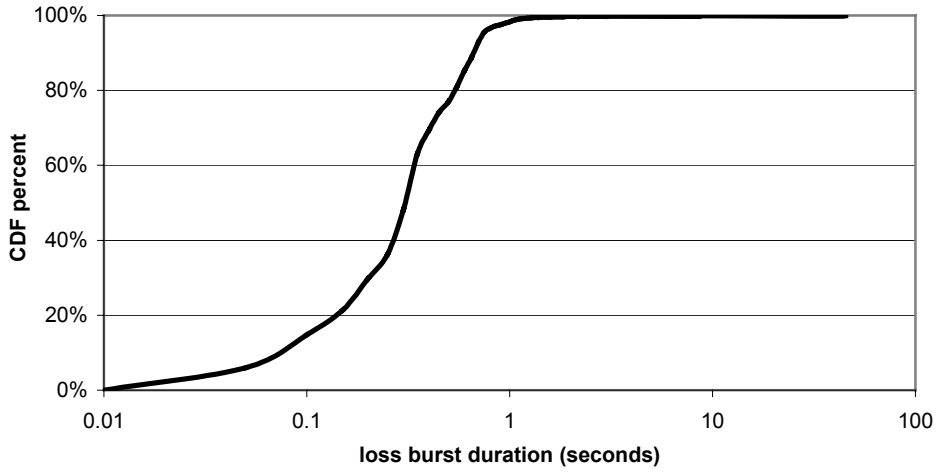

Figure 11. The CDF function of loss burst durations in $\{D_{1p} \cup D_{2p}\}$.

3.3.2.4 Heavy Tails

In conclusion of this section, it is important to note that packet losses sometimes cannot be modeled as independent events due to buffer overflows that last long enough to affect multiple adjacent packets. Consequently, future real-time protocols should expect to deal with bursty packet losses (Figure 10) and possibly heavy-tailed distributions of loss burst lengths (see below).

Several researchers reported a heavy-tailed nature of loss burst lengths, and the shape parameter α of the Pareto distribution fitted to the length (or duration) of loss bursts was recorded to range from 1.06 (Paxson [209]) to 2.75 (Borella *et al*. [30]). On

the other hand, Yajnik *et al*. [283] partitioned the collected data into stationary segments and reported that loss burst lengths could be modeled as exponential (i.e., not heavy-tailed) within each stationary segment. In addition, Zhang *et al*. [292] reported that packet loss along some Internet paths was stationary and could be modeled as exponential, whereas other paths were found to be non-stationary and not easy to model.

Using intuition, it is clear that packet loss and RTT random processes in both D_{1p} and D_{2p} are expected to be non-stationary. For example, the non-stationarity can be attributed to the time of day or the location of the client. In either case, we see three approaches to modeling such non-stationary data. In the first approach, we would have to analyze 16,852 PDF functions (one for each session) for stationarity and heavy tails. Unfortunately, an average session contained only 24 loss bursts, which is insufficient to build a good distribution function for a statistical analysis.

The second approach would be to combine all sessions into groups, which are intuitively perceived to be stationary (e.g., according to the access point or the timeslot), and then perform similar tests for stationarity and heavy tails within each group. We might consider this direction for future work. The third approach is to do what the majority has done in the past – assume that all data samples belong to a stationary process and are drawn from a single distribution. Using this last approach, Figure 12 shows a log-log plot of the complementary CDF function from Figure 10 with a least-squares fit of a straight line representing a hyperbolic (i.e., heavy-tailed) distribution (the dotted curve is the exponential distribution fitted to the data). The fit of a straight line is quite good (with correlation $\rho = 0.99$) and provides a strong indication that the distribution of loss burst

lengths in the combined dataset $\{D_{1p} \cup D_{2p}\}$ is heavy-tailed. Furthermore, as expected, we notice that the exponential distribution in Figure 12 decays too quickly to even remotely fit the data.

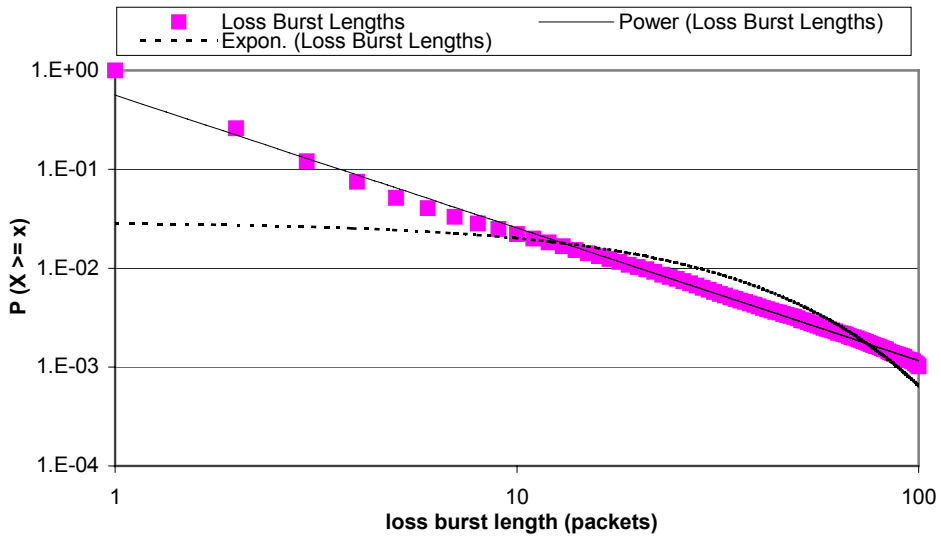

Figure 12. The complimentary CDF of loss burst lengths in $\{D_{1p} \cup D_{2p}\}$ on a log-log scale fitted with hyperbolic (straight line) and exponential (dotted curve) distributions.

Finally, consider a Pareto distribution with CDF $F(x) = 1-(\beta/x)^{\alpha}$ and PDF $f(x) = \alpha\beta^{\alpha}x^{-\alpha-1}$, where α is the shape parameter and β is the location parameter. Using Figure 12, we establish that a Pareto distribution with $\alpha = 1.34$ (finite mean, but infinite variance) and $\beta = 0.65$ fits our data very well.

3.3.3 Underflow Events

The impact of packet losses on real-time applications is understood fairly well. Each lost packet that is not recovered before its deadline causes an underflow event. In addition to packet loss, real-time applications suffer from large end-to-end delays. How-

ever, not all types of delay are equally important to real-time applications. As we will show below, one-way delay jitter was responsible for 90 times more underflow events in our experiment than packet loss combined with large RTTs.

Delays are important for two reasons. First, large round-trip delays make retransmissions late for their decoding deadlines. However, the RTT is important only to the extent of recovering lost packets and, in the worst case, can cause only *lost* packets to be late for decoding. On the other hand, the second kind of delay, delay jitter (i.e., one-way delay variation), can potentially cause each *data* (i.e., non-retransmitted) packet to be late for decoding.

Consider the following. In $\{D_{1p} \cup D_{2p}\}$, packet loss affected 431,501 packets, out of which 159,713 (37%) were discovered to be missing *after* their decoding deadlines had passed, and consequently, NACKs were not sent for these packets. Out of 271,788 remaining lost packets, 257,065 (94.6%) were recovered before their deadlines, 9,013 (3.3%) arrived late, and 5,710 (2.1%) were never recovered. The fact that more than 94% of "recoverable" lost packets were actually received before their deadlines indicates that retransmission is a very efficient method of overcoming packet loss in real-time applications. Clearly, the success rate will be even higher in networks with smaller end-to-end delays.

Before we study underflow events caused by delay jitter, let us introduce two types of *late* retransmissions. The first type consists of packets that arrived after the decoding deadline of the last frame of the corresponding group of pictures (GOP). These packets were *completely* useless and were discarded. The second type of late packets,

which we call *partially late*, consists of those packets that missed their *own* decoding deadline, but arrived before the deadline of the last frame of the same GOP. Since the video decoder in our experiment could decompress frames at a substantially higher rate than the target fps, the client was able to use partially late packets for motion-compensated reconstruction of the remaining frames from the same GOP before *their* corresponding decoding deadlines. Out of 9,013 late retransmissions, 4042 (49%) were partially late. Using each partially late packet, the client was able to save on average 4.98 frames from the same GOP[18] in D_{1p} and 4.89 frames in D_{2p} by employing the above-described *catch-up* decoding technique (for more discussion, see [241]).

The second type of delay, one-way delay jitter, caused 1,167,979 *data* (i.e., non-retransmitted) packets to miss their decoding deadlines. Hence, the total number of *underflow* (i.e., missing at the time of decoding) packets was 159,713 + 9,013 + 5,710 + 1,167,979 = 1,342,415 (1.7% of the number of sent packets), which means that 98.9% of underflow packets were created by large one-way delay jitter, rather than by pure packet loss. Even if the clients had not attempted to recover any lost packets, still 73% of the missing packets at the time of decoding would have been caused by large delay jitter. Furthermore, these 1.3 million underflow packets caused a "freeze-frame" effect for the average duration of 10.5 seconds per ten-minute session in D_{1p} and 8.6 seconds in D_{2p}, which can be considered excellent given the small amount of delay budget (i.e., startup delay) used in the experiments.

---

[18] We used 10-frame GOPs in both sequences.

To further understand the phenomenon of late packets, we plotted in Figure 13 the CDFs of the amount of time by which late packets missed their deadlines (i.e., the amount of time that we need to add to delay budget D_{budget} = 2,700 ms in order to avoid a certain percentage of underflow events) for both late retransmissions and late data packets. As the figure shows, 25% of late retransmissions missed their deadlines by more than 2.6 seconds, 10% by more than 5 seconds, and 1% by more than 10 seconds (the tail of the CDF extends up to 98 seconds). At the same time, one-way delay jitter had a more adverse impact on data packets – 25% of late data packets missed their deadlines by more than 7 seconds, 10% by more than 13 seconds, and 1% by more than 27 seconds (the CDF tail extends up to 56 seconds).

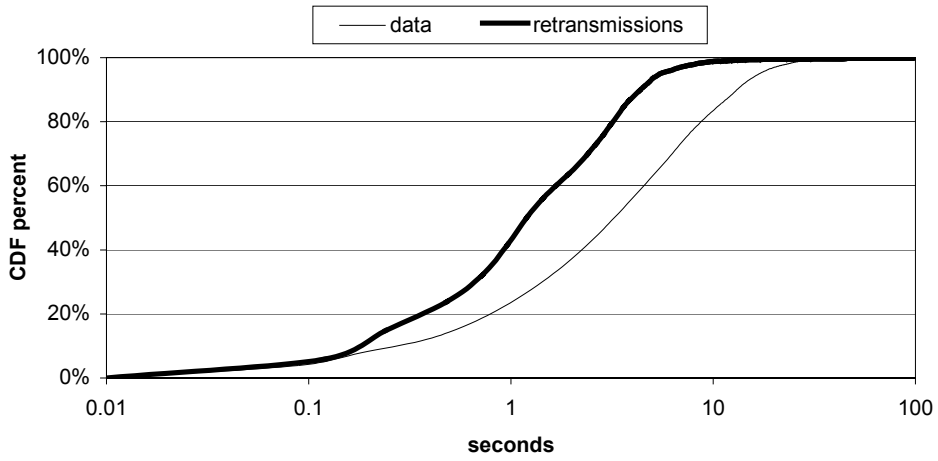

Figure 13. CDF functions of the amount of time by which retransmitted and data packets were late for decoding.

The only way to reduce the number of late packets caused by both large RTTs and delay jitter is to apply a larger startup delay D_{budget} at the beginning of a session (in addition to freezing the display and adding extra startup delays during the session, or running

the server at a faster-than-ideal bitrate to accumulate more frames in the decoder buffer, neither of which was acceptable in our model). Hence, for example, Internet applications utilizing a 13-second delay budget (which corresponds to 10.3 seconds of *additional* delay in Figure 13) would be able to "rescue" 99% of late retransmissions and 84% of late data packets in similar streaming conditions.

3.3.4 Round-trip Delay

3.3.4.1 Overview

We should mention that circuit-switched long-distance links through PSTN between our clients and remote access points did not significantly influence the measured end-to-end delays, because the additional delay on each long-distance link was essentially the propagation delay between New York and the location of the access point (which is determined by the speed of light and the geographical distance, i.e., 16 ms coast to coast). Clearly, this delay is negligible compared to the queuing and transmission delays experienced by each packet along the entire end-to-end path. Figure 14 shows the PDF functions of the round-trip delay in each of D_{1p} and D_{2p} (660,439 RTT samples in both datasets). Although the tail of the combined distribution reached the enormous values of 126 seconds for *simulated* and 102 seconds for *real* retransmissions, the majority (75%) of the samples were below 600 ms, 90% below 1 second, and 99.5% below 10 seconds. The average RTT was 698 ms in D_{1p} and 839 ms in D_{2p}. The minimum RTT was 119 and 172 ms, respectively. Although very rare, extremely high RTTs were found in all six datasets $D_{1p}{}^{a} - D_{2p}{}^{c}$. Furthermore, out of more than 660 thousand RTT samples in

$\{D_{1p} \cup D_{2p}\}$, 437 were at least 30 seconds, 32 at least 50 seconds, and 20 at least 75 seconds.

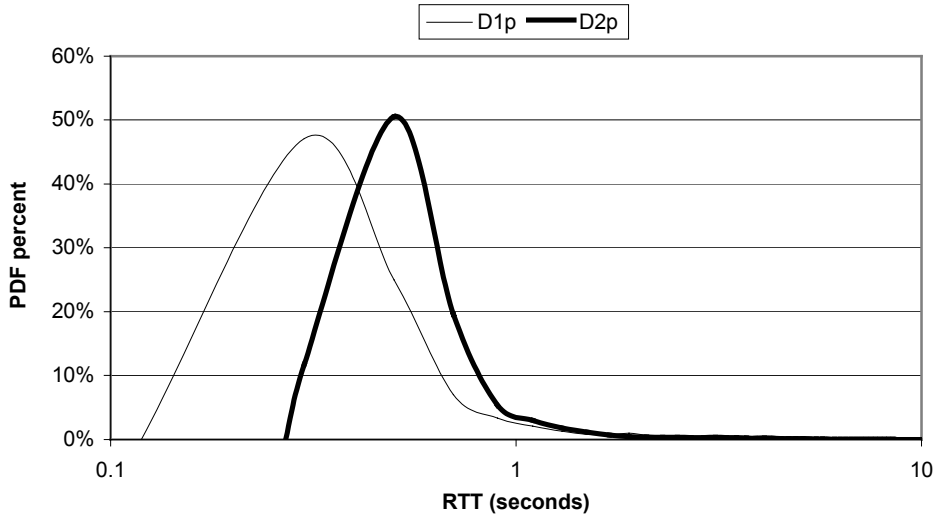

Figure 14. PDF functions of the RTT samples in each of D_{1p} and D_{2p}.

Although pathologically high RTTs may seem puzzling at first, there is a simple explanation. Modem error correction protocols (i.e., the commonly used V.42) implement retransmission for corrupted blocks of data on the physical layer.[19] Error correction is often necessary if modems negotiated data compression (i.e., V.42bis) over the link and is desirable if PPP Compression Control Protocol (CCP) is enabled on the data-link layer. In all our experiments, both types of compression were enabled, imitating the typical setup of a home user. Therefore, if a client established a connection to a remote modem at a low bitrate (which was sometimes accompanied by a significant amount of noise in

the phone line), each retransmission on the physical layer took a large time to complete before the data was delivered to the upper layers. In addition, large IP-level buffers on either side of the modem link further delayed packets arriving to or originating from the client host.

Note that the purpose of classifying sessions into failed and successful in section 3.2.1 was to avoid reporting pathological conditions caused by the modem links. Since only a handful (less than 0.5%) of RTTs in $\{D_{1p} \cup D_{2p}\}$ were seriously effected by modem-level retransmission and bit errors (we consider sessions with RTTs higher than 10 seconds to be caused by modem-related problems[20]), we conclude that our heuristic was successful in filtering out the majority of pathological connections and that future application-layer protocols, running over a modem link, should be prepared to experience RTTs in the order of several seconds.

Furthermore, the removal of sessions with RTTs higher than 10 seconds does not change any of the results below and, at the same time, prohibits us from showing the extent of variation in the network parameters experienced by a home Internet user. Therefore, since all sessions in $\{D_{1p} \cup D_{2p}\}$ were able to successfully complete, we consider the removal of sessions based on their large RTT to be unwarranted.

---

[19] Since the telephone network beyond the local loop in the U.S. is mostly digital, we believe that dialing long-distance (rather than local) numbers had no significant effect on the number of bit errors during the experiment.

[20] For example, one of the authors uses a modem access point at home with IP-level buffering on the ISP side equivalent to 6.7 seconds. Consequently, delays as high as 5-10 seconds may often be caused by non-pathological conditions.

3.3.4.2 Heavy Tails

Mukherjee [182] reported that the distribution of the RTT along certain Internet paths could be modeled as a shifted gamma distribution. Even though the shape of the PDF in Figure 14 resembles that of a gamma function, the distribution tails in the figure decay much slower than those of an exponential distribution (see below). Using our approach from section 3.3.2.4 (i.e., assuming that each studied Internet random process is stationary), we extracted the upper tails of the PDF functions in Figure 14 and plotted the results on a log-log scale in Figure 15. The figure shows that a straight line (without loss of generality fitted to the PDF of D_{2p} in the figure) provides a good fit to the data (correlation 0.96) and allows us to model the upper tails of the PDF functions as a Pareto distribution with PDF $f(x) = \alpha\beta^{\alpha}x^{-\alpha-1}$, where shape parameter α equals 1.16 in dataset D_{1p} and 1.58 in D_{2p} (as before, the distribution has a finite mean, but an infinite variance).

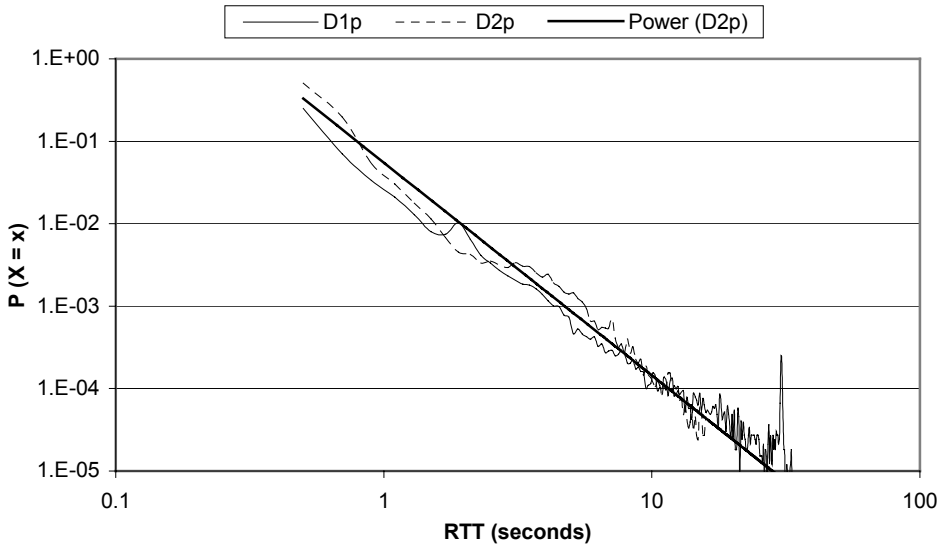

Figure 15. Log-log plot of the upper tails of the distribution of the RTT (PDF). The straight line is fitted to the PDF from D_{2p}.

3.3.4.3 Variation of the RTT

We conclude the discussion of the RTT by showing that the round-trip delay exhibited a variation during the day similar to that of packet loss (previously shown in Figure 8) and that the average RTT was positively correlated with the length of the end-to-end path. Figure 16 shows the average round-trip delay during each of the eight time-slots of the day (as before, each point in the figure represents the average of approximately 1,000 sessions). The figure confirms that the worst time for sending traffic over the Internet is between 9 am and 6 pm EDT and shows that the increase in the delay during the peak hours is relatively small (i.e., by 30-40%).

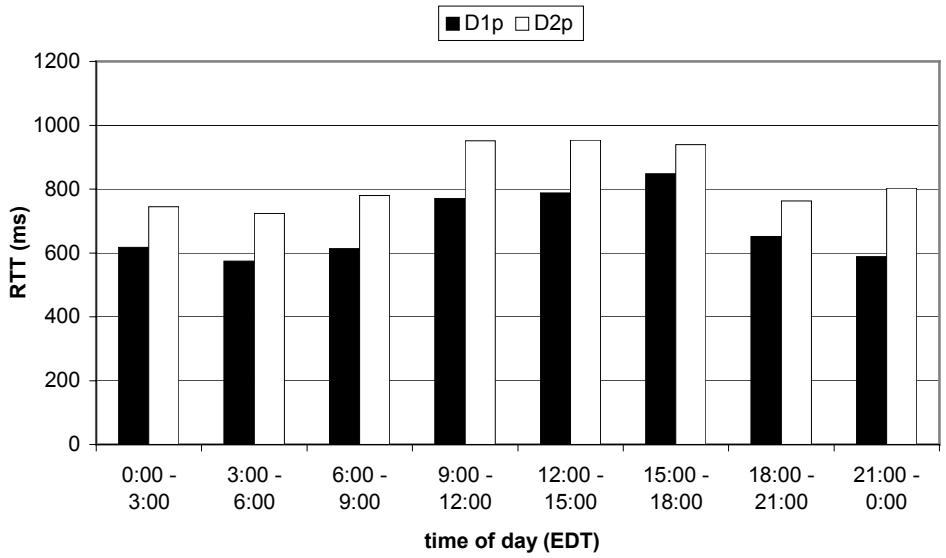

Figure 16. Average RTT as a function of the time of day.

Figure 17 shows the average RTT sampled by the clients in each of the 50 U.S. states. The average round-trip delay was consistently high (above 1 second) for three states – Alaska, New Mexico, and Hawaii. On the other hand, the RTT was consistently

low (below 600 ms) also for three states – Maine, New Hampshire, and Minnesota. These results (except Minnesota) can be directly correlated with the distance from New York; however, in general, we find that the geographical distance of the access point from the East Coast had little correlation with the average RTT. Thus, for example, some states in the Midwest had small (600-800 ms) average round-trip delays and some states on the East Coast had large (800-1000 ms) average RTTs. A more substantial link can be established between the number of end-to-end hops and the average RTT as shown in Figure 17. Even though the average RTT of many states did not exhibit a clear dependency on the average length of the path, the correlation between the RTT and the number of hops in Figure 17 was reasonably high with $\rho = 0.52$. This result was intuitively expected since the RTT is essentially the sum of queuing and transmission delays at intermediate routers.

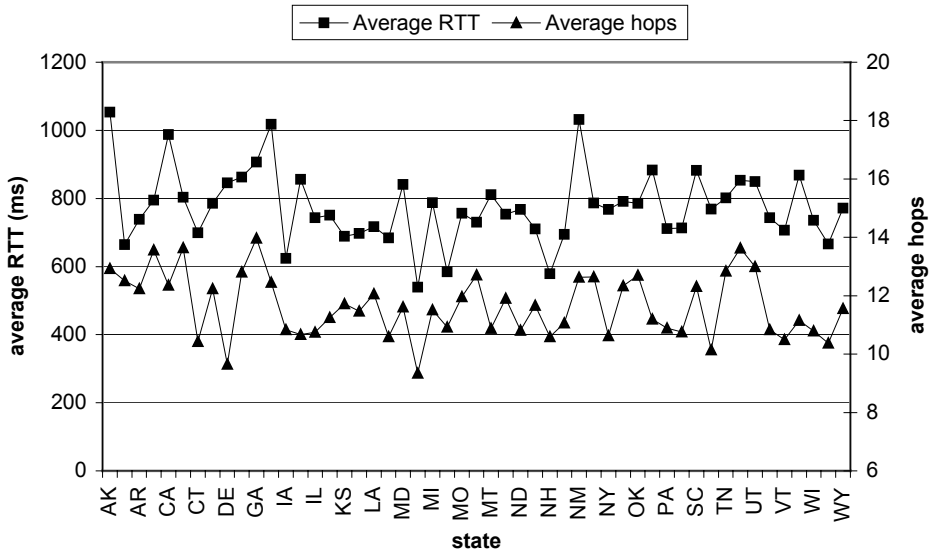

Figure 17. Average RTT and average hop count in each of the states in $\{D_{1p} \cup D_{2p}\}$.

3.3.5 Delay Jitter

As we discussed above, in certain streaming situations, round-trip delays are much less important to real-time applications than one-way delay jitter, because the latter can potentially cause significantly more underflow events. In addition, due to asymmetric path conditions (i.e., uneven congestion in the upstream and downstream directions), large RTTs are not necessarily an indication of bad network conditions for a NACK-based application. For example, in many sessions with high RTTs during the experiment, the outage was caused by the upstream path, while the downstream path did not suffer from extreme one-way delay variation, and data (i.e., non-retransmitted) packets were arriving to the client throughout the entire duration of the outage. Hence, we conclude that the value of the RTT was not necessarily a good indicator of a session's quality during the experiment and that one-way delay jitter should be used instead.

Assuming that delay jitter is defined as the difference between one-way delays of each two consecutively sent packets, an application can sample both positive and negative values of delay jitter. Negative values are produced by two types of packets – those that suffered a *packet compression event* (i.e., the packets' arrival spacing was smaller than their transmission spacing) and those that became reordered. The former case is of great interest in packet-pair bandwidth estimation studies and otherwise remains relatively unimportant. The latter case will be studied in section 3.3.6 under packet reordering. On the other hand, positive values of delay jitter represent *packet expansion events*, which are responsible for late packets. Consequently, we analyzed the distribution of only *positive* delay jitter samples and found that although the highest sample was 45 sec-

onds, 97.5% of the samples were under 140 ms and 99.9% under 1 second. As the above results show, large values of delay jitter were not frequent, but once a packet was significantly delayed by the network, a substantial number of the following packets were delayed as well, creating a "snowball" of late packets. This fact explains the large number of underflow events reported in previous sections, even though the overall delay jitter was relatively low.

3.3.6 Packet Reordering

3.3.6.1 Overview

Real-time protocols often rely on the assumption that packet reordering in the Internet is a rare and an insignificant event for all practical purposes (e.g., [63]). Although this assumptions simplifies the design of a protocol, it also make the protocol poorly suited for the use over the Internet. Certainly, there are Internet paths along which reordering is either non-existent or extremely low. At the same time, there are paths that are dominated by multipath routing effects and often experience reordering (e.g., Paxson [209] reported a session with 36% of packets arriving out of order).

Unfortunately, there is not much data documenting reordering rates experienced by IP traffic over modem links. Using intuition, we expected reordering in our experiments to be extremely rare given the low bitrates of streams S_1 and S_2. However, we were surprised to find out that certain paths experienced consistent reordering with a relatively large number of packets arriving our of order, although the average reordering rates in our experiments were substantially lower than those reported by Paxson [209].

For example, in dataset $D_{1p}{}^{a}$, we observed that out of every three missing[21] packets one was reordered. Hence, if users of ISP_a employed a streaming protocol, which used a gap-based detection of lost packets [63] (i.e., the first out-of-order packet triggers a NACK), 33% of NACKs would be flat-out redundant and a large number of retransmissions would be unnecessary, causing a noticeable fraction of ISP's bandwidth to be wasted.

Since each missing packet is potentially reordered, the true frequency of reordering can be captured by computing the percentage of reordered packets relative to the total number of *missing* packets. The average reordering rate in our experiment was 6.5% of the number of *missing* packets, or 0.04% of the number of *sent* packets. These numbers show that our reordering rates were at least by a factor of 10 lower than those reported by Paxson [209], whose average reordering rates varied between 0.3% and 2% of number of sent packets depending on the dataset. This difference can be explained by the fact that our experiment was conducted at substantially lower end-to-end bitrates, as well as by the fact that Paxson's experiment involved several paths with extremely high reordering rates.

Out of 16,852 sessions in $\{D_{1p} \cup D_{2p}\}$, 1,599 (9.5%) experienced at least one reordering. Interestingly, the average session reordering rates in our datasets were very close to those in Paxson's 1995 data [209] (12% sessions with at least one reordering), despite the fundamental differences in sending rates. The highest reordering rate *per ISP* in our

[21] Missing packets are defined as gaps in sequence numbers.

experiment occurred in D_{1p}^{a}, where 35% of the number of missing packets (0.2% of the number of sent packets) turned out to be reordered. In the same D_{1p}^{a}, almost half of the sessions (47%) experienced at least one reordering event. Furthermore, the maximum number of reordered packets in a single session occurred in D_{1p}^{b} and was 315 packets (7.5% of the number of sent packets).

Interestingly, the reordering probability did not show any dependence on the time of day (i.e., the timeslot), and was virtually the same for all states.

3.3.6.2 Reordering Delay

To further study packet reordering, we define two metrics that allow us to measure the extent of packet reordering. First, let *packet reordering delay* D_r be the delay from the time when a reordered packet was declared as *missing* to the time when the reordered packet arrived to the client. Second, let *packet reordering distance* d_r be the number of packets (including the very first out-of-sequence packet, but not the reordered packet itself) received by the client during reordering delay D_r. These definitions are illustrated in Figure 18, where reordering distance d_r is 2 packets and reordering delay D_r is the delay between receiving packets 3 and 2.

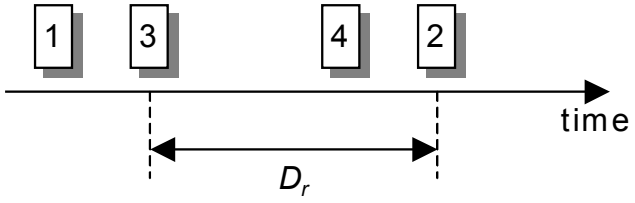

Figure 18. The meaning of reordering delay D_r.

Figure 19 shows the PDF of the reordering delay D_r in $\{D_{1p} \cup D_{2p}\}$. The largest reordering distance d_r in the combined dataset was 10 packets, and the largest reordering delay D_r was 20 seconds (however, in the latter case, d_r was only 1 packet). Although quite large, the maximum value of D_r is consistent with previously reported numbers (e.g., 12 seconds in Paxson's data [209]). The majority (90%) of samples in Figure 19 are below 150 ms, 97% below 300 ms, and 99% below 500 ms.

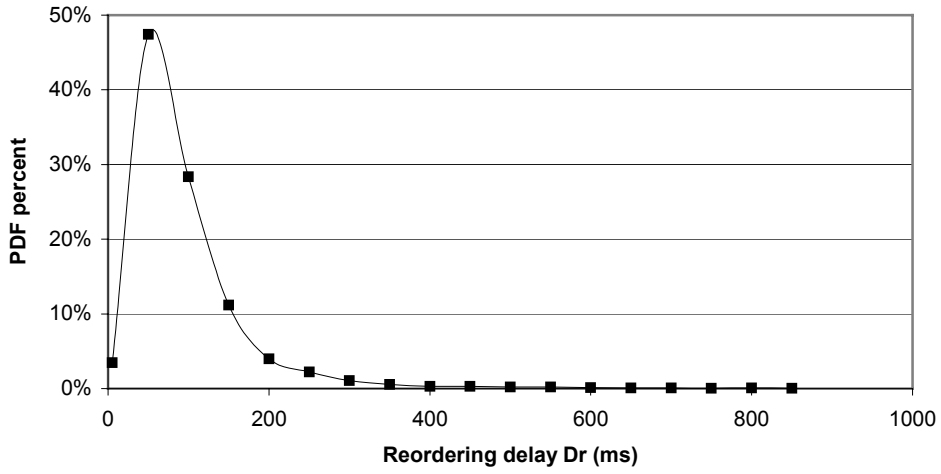

Figure 19. The PDF of reordering delay D_r in $\{D_{1p} \cup D_{2p}\}$.

3.3.6.3 Reordering Distance

We next analyze the suitability of TCP's triple-ACK scheme in helping NACK-based protocols detect reordering. TCP's *fast retransmit* relies on *three* consecutive duplicate ACKs (hence the name "triple-ACK") from the receiver to detect packet loss and avoid unnecessary retransmissions. Therefore, if reordering distance d_r is either 1 or 2, the triple-ACK scheme successfully avoids duplicate packets, and if d_r is greater than or

equal to 3, it generates a duplicate packet. Figure 20 shows the PDF of reordering distance d_r in both datasets. Using the figure, we can infer that TCP's triple-ACK would be successful for 91.1% of the reordering events in our experiment, double-ACK for 84.6%, and quadruple-ACK for 95.7%. Note that Paxson's TCP-based data [209] show similar, but slightly better detection rates, specifically 95.5% for triple-ACK, 86.5% for double-ACK, and 98.2% for quadruple-ACK.

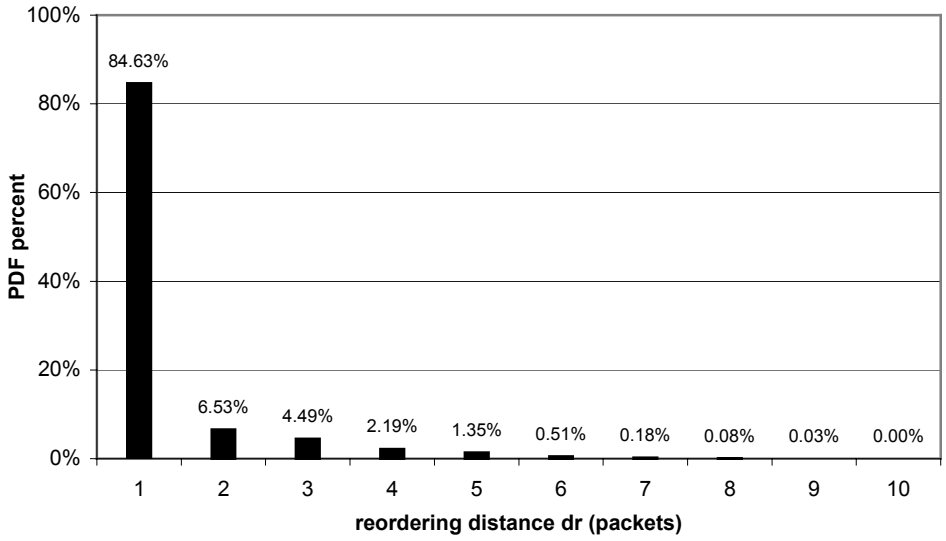

Figure 20. The PDF of reordering distance d_r in $\{D_{1p} \cup D_{2p}\}$.

3.3.7 Asymmetric Paths

Recall that during the initial executions of traceroute, our dialer recorded the TTL fields of each received "TTL expired" packet. The TTL fields of these packets allowed the dialer to compute the number of hops between the router that generated a particular "TTL expired" message and the client. Suppose some router i was found at hop f_i in the *upstream* (i.e., forward) direction and at hop r_i in the *downstream* (i.e., reverse) direction.

Hence, we can conclusively establish that an *n*-hop end-to-end path is *asymmetric*, if there exists a router for which the number of downstream hops is different from the number of upstream hops (i.e., $\exists i,\ 1 \le i \le n: f_i \ne r_i$). However, the opposite is not always true – if each router has the same number of downstream and upstream hops, we cannot conclude that the path is symmetric (i.e., it could be asymmetric as well). Hence, we call such paths *possibly-symmetric*.[22]

In $\{D_{1p} \cup D_{2p}\}$, 72% of the sessions sent their packets over *definitely* asymmetric paths. In that regard, two questions prompt for an answer. First, does path asymmetry depend on the number of end-to-end hops? To answer this question, we extracted path information from $\{D_{1p} \cup D_{2p}\}$ and counted each end-to-end path through a particular access point exactly once. Figure 21 shows the percentage of asymmetric paths as a function of the number of end-to-end hops in the path. As the figure shows, almost all paths with 14 hops or more were asymmetric, as well as that even the shortest paths (with only 6 hops) were prone to asymmetry. This result can be explained by the fact that longer paths are more likely to cross over AS boundaries or intra-AS administrative domains. In both cases, "hot-potato" routing policies can cause path asymmetry.

---

[22] In the most general case, even performing a reverse traceroute from the server to the client could not have conclusively established each path's symmetry, because traceroute identifies router interfaces rather than individual routers. See [209] for more discussion.

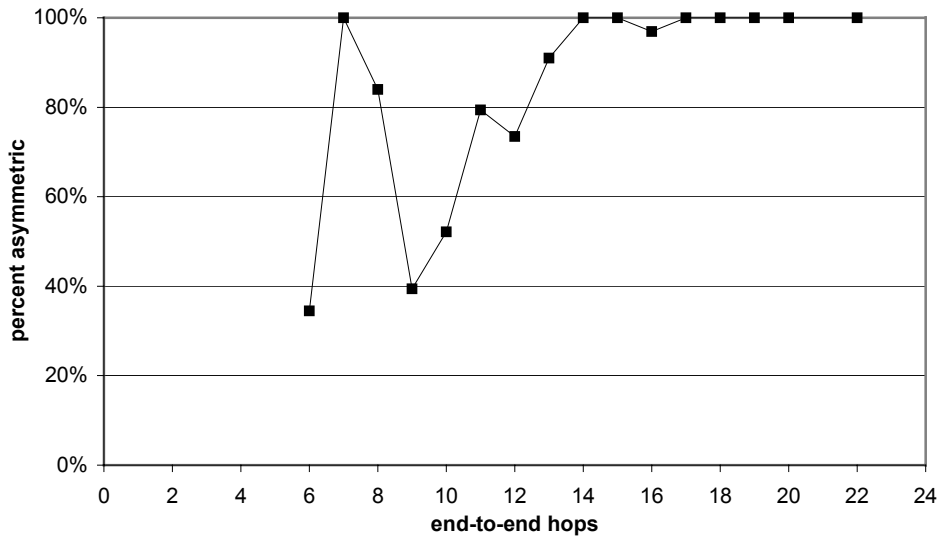

Figure 21. Percentage of asymmetric routes in $\{D_{1p} \cup D_{2p}\}$ as a function of the number of end-to-end hops.

The second question we attempt to answer is whether path asymmetry has anything to do with reordering. In $\{D_{1p} \cup D_{2p}\}$, 95% of all sessions with at least one reordered packet were running over an *asymmetric* path. Consequently, we can conclude that if a session in our datasets experiences a reordering event along a path, then the path is most likely asymmetric. However, a new question that arises is whether the opposite is true as well: if a path is asymmetric, will a session be more likely to experience a reordering? To answer the last question, we have the following numbers. Out of 12,057 sessions running over a definitely asymmetric path, 1,522 experienced a reordering event, which translates into 12.6% reordering rate. On the other hand, out of 4,795 sessions running over a possibly-symmetric path, only 77 (1.6%) experienced a reordering event. Hence, an asymmetric path is 8 times more likely to experience a reordering event than a possibly-symmetric path.

Even though there is a clear link between reordering and asymmetry in our data-sets, we speculate that the two could be related through the length of each end-to-end path. In other words, longer paths were found to be more likely to experience reordering as well as be asymmetric. Hence, rather than saying that reordering causes asymmetry or vice versa, we can explain the result by noting that longer paths are more likely to cross inter-AS boundaries or intra-AS administrative domains, during which both "hot potato" routing (which causes asymmetry) and IP-level load-balancing (which causes reordering) are apparently quite frequent.

Clearly, the findings in this section depend on the particular ISP employed by the end-user and the autonomous systems that user traffic traverses. Large ISPs (such as the ones studied in this work) often employ numerous peering points (hundreds in our case), and path asymmetry rates found in this section may not hold for smaller ISPs. Nevertheless, our data allow us to conclude that the majority of home users in the US experience asymmetric end-to-end paths with a much higher frequency than symmetric ones.

Chapter Four

4 Real-time Retransmission: Evaluation Model and Performance

This chapter presents a trace-driven simulation study of two classes of retransmission timeout (RTO) estimators in the context of real-time streaming over the Internet. We explore the viability of employing retransmission timeouts in NACK-based (i.e., rate-based) streaming applications to support multiple retransmission attempts per lost packet. The first part of our simulation is based on trace data collected during a number of real-time streaming tests between dialup clients in all 50 states in the U.S. (including 653 major U.S. cities) and a backbone video server. The second part of the study is based on streaming tests over DSL and ISDN access links. First, we define a generic performance measure for assessing the accuracy of hypothetical RTO estimators based on the samples of the round-trip delay (RTT) recorded in the trace data. Second, using this performance measure, we evaluate the class of TCP-like estimators and find the optimal estimator

given our performance measure. Third, we introduce a new class of estimators based on delay jitter and show that they significantly outperform TCP-like estimators in NACK-based applications with *low-frequency* RTT sampling. Finally, we show that high-frequency sampling of the RTT completely changes the situation and makes the class of TCP-like estimators as accurate as the class of delay-jitter estimators.

4.1 Introduction

Many Internet transport protocols rely on retransmission to recover lost packets. Reliable protocols (such as TCP) utilize a well-established sender-initiated retransmission scheme that employs retransmission timeouts (RTO) and duplicate acknowledgements (ACKs) to detect lost packets [110]. RTO estimation in the context of retransmission refers to the problem of predicting the next value of the round-trip delay (RTT) based on the previous samples of the RTT. RTO estimation is usually a more complicated problem than simply predicting the *most likely* value of the next RTT. For example, an RTO estimator that always underestimates the next RTT by 10% is significantly worse than the one that always overestimates the next RTT by 10%. Although both estimators are within 10% of the correct value, the former estimator generates 100% duplicate packets, while the latter one avoids all duplicate packets with only 10% unnecessary waiting.

Even though Jacobson's RTO estimator [110] is readily accepted by many TCP-like protocols, the problem of estimating the RTO in streaming protocols has not been

addressed before. Current streaming protocols [232] deployed in the Internet rely on NACK-based flow control and usually do not implement congestion control, and the question of whether TCP's RTO estimator is suitable for such protocols remains an open issue. This chapter sheds new light on the problem of RTO estimation in NACK-based protocols and shows the performance of several classes of RTO estimators in realistic Internet streaming scenarios.

Traditionally, NACK-based protocols sample the RTT only at times of packet loss (see below for details of how this is done). Even though there is nothing that inherently stops NACK-based protocols from sampling the RTT at a higher rate, our study for the most part follows the assumptions of the existing NACK-based applications [232] (i.e., the receiver sends messages to the server only upon packet loss and the RTT is measured only for the retransmitted packets).

As a result of our investigation, we found that TCP's RTO was an inadequate predictor of future values of the RTT when used in a NACK-based protocol over paths with *low-frequency* RTT sampling (i.e., low packet loss). We further found that along such paths, the accuracy of estimation could be substantially improved if the client used delay jitter in its computation of the RTO. On the other hand, when the RTT sampling rate was increased, TCP's RTO performed very well and the benefits of delay jitter were much less significant. Since an application typically does not know its future packet loss rates, we find that NACK-based protocols, augmented with high-frequency (i.e., in the order of once per RTT) sampling of the round-trip delay, will perform very well regardless of the end-to-end characteristics of a particular path (for example, high-frequency

RTT sampling in real-time streaming can be implemented by using congestion control messages and once-per-RTT receiver-based feedback [157]).

In addition, this chapter presents a generalized (i.e., suitable for many real-time applications) NACK-based, real-time retransmission model for multimedia streaming over the Internet and assess the effectiveness of various RTO estimators in the context of Internet streaming and our retransmission model. While the primary goal of our study is to develop a better retransmission mechanism for real-time applications, our retransmission model and new performance measure introduced in this chapter are generic enough to apply to TCP as well.

Our characterization of RTO estimators is based on a reasonably large number of real-time streaming tests conducted between dialup clients from all 50 states in the U.S. and a backbone server during a seven-month period. We believe that this setup accurately reflects the current situation with real-time streaming in the Internet since the majority (i.e., 87-89%) of households in the U.S. still connect to the Internet through dialup modems [108], [217].

A good RTO estimator is the basis of any retransmission scheme. An application utilizing an RTO estimator that consistently *underestimates* the round-trip delay generates a large number of duplicate packets. The effect of duplicate packets ranges from being simply wasteful to actually causing serious network congestion. Note that in NACK-based applications, the receiver (i.e., the client) is responsible for estimating the RTO and the server is no longer in charge of deciding when to initiate a particular retransmission.

This is illustrated in Figure 22 (left), in which the client sends three NACKs in response to a single lost packet and produces two duplicate packets due to insufficient RTO.

On the other hand, *overestimation* of the RTT defers the generation of subsequent retransmission requests and leads to lower throughput performance in TCP and causes an increased number of *underflow events* (which are generated by packets arriving after their decoding deadlines) in real-time applications. In either case, the amount of overestimation can be measured by the duration of unnecessary waiting for timeouts (i.e., waiting longer than the RTT of the lost retransmission). This is illustrated in Figure 22 (right). In the figure, the first retransmission is lost as well, and the generation of the second NACK is significantly delayed because the RTO is higher than the network RTT.

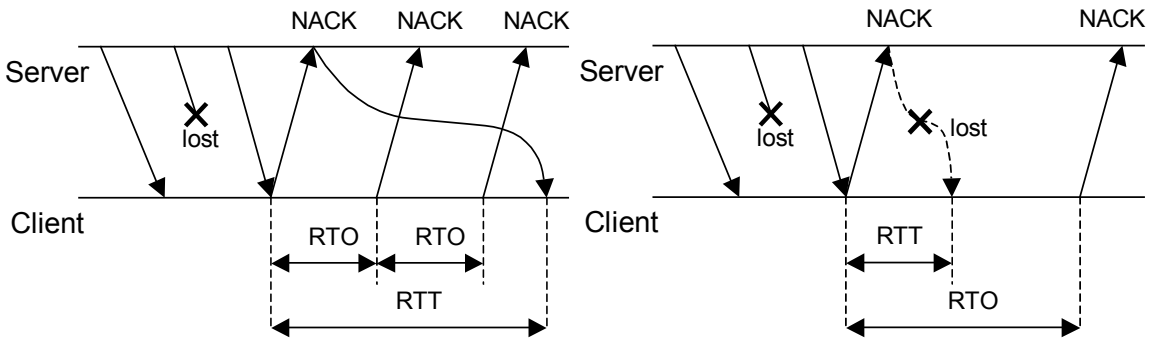

Figure 22. Underestimation results in duplicate packets (left) and overestimation results in unnecessary waiting (right).

Therefore, the performance (i.e., accuracy) of an RTO estimator is fully described by two parameters (quantified later in this chapter) – the number of duplicate packets and the amount of unnecessary timeout waiting. These two parameters cannot be minimized at the same time, since they represent a basic trade-off of any RTO estimator (i.e., decreasing one parameter will increase the other). To study the performance of RTO esti-

mators, we define a weighted sum of these two parameters and study a multidimensional optimization problem in order to find the tuning parameters that make an RTO estimator optimal within its class. The minimization problem is not straightforward because the function to be minimized is non-continuous, has unknown (and often non-existent) derivatives, and contains a large number of local minima.

The chapter is organized as follows. Section 4.2 provides the background on the problem of estimating retransmission timeouts and discusses some of the related work. Section 4.3 describes the methodology of our experiment. Section 4.4 introduces a novel performance measure that is used to judge the accuracy of hypothetical RTO estimators throughout this chapter. Section 4.5 studies the class of TCP-like RTO estimators and models their performance. Section 4.6 discusses a new class of jitter-based RTO estimators and shows their superior performance in our modem datasets. Section 4.7 studies the performance of RTO estimators along high-speed Internet paths with high-frequency RTT sampling and shows that these paths require a different estimator.

4.2 Background

Recall that TCP's RTO estimation consists of three algorithms. The first algorithm, *smoothed RTT estimator* (SRTT), is an exponentially-weighted moving average (EWMA) of the past RTT samples [4], [110], [212]:

$$SRTT_i = \begin{cases} RTT_0, & i = 0 \\ (1-\alpha) \cdot SRTT_{i-1} + \alpha \cdot RTT_i, & i \geq 1 \end{cases}, \tag{6}$$

where RTT_i is the i-th sample of the round-trip delay produced at time t_i and α (set by default to 1/8) is a smoothing factor that can be varied to give more or less weight to the history of RTT samples. In the original RFC 793 [221], the RTO was obtained by multiplying the latest value of the SRTT by a fixed factor between 1.3 and 2.0. In the late 1980s, Jacobson [110] found that the RFC 793 RTO estimator produced an excessive amount of duplicate packets when employed over the Internet and proposed that the second algorithm, *smoothed RTT variance estimator* (*SVAR*), be added to TCP's retransmission scheme [4], [110], [212]:

$$SVAR_i = \begin{cases} RTT_0 / 2, & i = 0 \\ (1-\beta) \cdot SVAR_{i-1} + \beta \cdot VAR_i, & i \geq 1 \end{cases}, \tag{7}$$

where β (set by default to ¼) is an EWMA smoothing factor and VAR_i is the absolute deviation of the i-th RTT sample from the previous smoothed average: $VAR_i = |SRTT_{i-1} - RTT_i|$. Current implementations of TCP compute the RTO by multiplying the smoothed variance by four and adding it to the smoothed round-trip delay [4], [212]:

$$RTO(t) = n \cdot SRTT_i + k \cdot SVAR_i, \tag{8}$$

where t is the time at which the RTO is computed, $n = 1$, $k = 4$, and $i = \max i$: $t_i \leq t$.

The third algorithm involved in retransmission, *exponential timer backoff*, refers to Jacobson's algorithm [110] that exponentially increases the timeout value each time

the same packet is retransmitted by the sender. Exponential timer backoff does not increase the accuracy of an RTO estimator, but rather conceals the negative effects of underestimating the actual RTT.[23] Since this chapter focuses on tuning the accuracy of RTO estimators, we consider the timer backoff algorithm to be an orthogonal issue, to which we will not pay much attention. Furthermore, real-time applications have the ability to utilize a different technique that conceals RTT underestimation, which involves setting a deterministic limit on the number of retransmission attempts for each lost packet based on real-time decoding deadlines.

Rigorous tuning of TCP's retransmission mechanism has not been attempted in the past (possibly with the exception of [4]), and the study of TCP's RTO over diverse Internet paths is limited to [205], [209] in which Paxson found that 40% of retransmissions in the studied TCP implementations were redundant.[24]

Recently, Allman and Paxson [4] conducted a trace-driven simulation study based on TCP traffic to investigate the performance of hypothetical TCP-like RTO estimators (8) for several values of α, β, and k (n was kept at 1). The authors compared the performance of eight estimators by varying (α,β) and keeping k fixed at 4 and examined eight additional estimators by running k through eight integer values and keeping (α,β) fixed at their default values. The paper further concluded that no TCP-like RTO estimator could perform significantly better in the future versions of TCP than Jacobson's de-facto

---

[23] Another method of reducing the number of duplicate packets in TCP is to use a minimum of 1 second in (8), as suggested in a recent IETF document [212].

[24] Note that not all redundant retransmissions were due to an insufficient RTO.

standard [110] and that even varying parameter *n* in (8) would not make the estimator substantially better.

Among other reliable protocols with non-Jacobson RTO estimation, Keshav *et al*. [124] employed sender-based retransmission timeouts equal to twice the *SRTT* (i.e., the RFC 793 estimator), and Gupta *et al*. [96] used a NACK-based retransmission scheme, in which receiver timeouts and detection of lost packets were based on inter-packet arrival delay jitter.

The situation with RTO estimation in real-time streaming applications is somewhat different – the majority of real-time protocols either use TCP's RTO or rely on novel RTO estimation methods whose performance in the real Internet is unknown. Papadopoulos *et al*. [199] proposed a real-time retransmission scheme in which the receiver used the value of the SRTT in (1) to decide which packets were eligible for the first retransmission and employed special packet headers to support subsequent retransmissions. The benefit of avoiding timeouts was offset by the inability of the proposed scheme to overcome NACK loss. Rhee [241] employed a retransmission scheme in which the sender used three *frame durations* (instead of an estimate of the RTT) to decide on subsequent retransmissions of the same packet. A similar sender-based retransmission scheme was proposed by Gong *et al*. [91], with the exception that the sender used an undisclosed estimate of the RTT to decide when the same packet was eligible for a repeated retransmission.

4.3 Methodology

4.3.1 Experiment

Our evaluation study of RTO estimators is based on experimental data collected in a large-scale real-time streaming experiment over the Internet during November 1999 – May 2000. Aiming to create a setup the reflects the current use of real-time streaming in the Internet by the majority of home users [232], we implemented an MPEG-4 real-time streaming client-server architecture with NACK-based retransmission and used it to sample the RTT process along diverse paths in the dialup Internet.

To achieve an extensive coverage of dialup points in the U.S., we selected three major national dialup ISPs (which we call ISP_a, ISP_b, and ISP_c), each with at least five hundred V.90 (i.e., 56 kb/s) dialup numbers in the U.S. and several million active subscribers. We further designed an experiment in which hypothetical Internet users of all 50 states dialed a local access number to reach the Internet and streamed video sequences from our backbone server. Although the clients were physically located in our lab in the state of New York, they dialed long-distance phone numbers (see Figure 23) and connected to the Internet through a subset of the ISPs' 1813 different V.90 access points located in 1188 U.S. cities. A detailed description of the experiment can be found in [156].

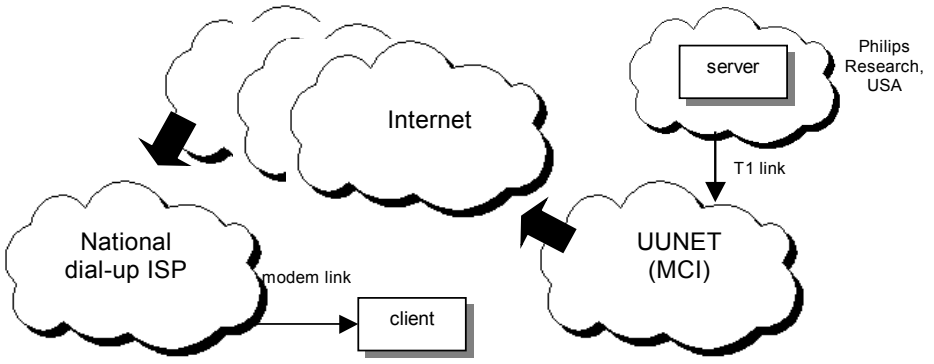

Figure 23. The setup of the modem experiment.

We used two 10-minute QCIF (176x144) MPEG-4 sequences coded at video bitrates of 14 and 25 kb/s. The corresponding *IP bitrates* (i.e., including IP, UDP, and our streaming headers) were 16.0 and 27.4 kb/s, respectively.

During the experiment, the clients performed over 34 thousand long-distance phone calls and received 85 million packets (27.1 GBytes of video data) from the server. The majority of end-to-end paths between the server and the clients contained between 10 and 13 hops (with 6 minimum and 22 maximum). Moreover, server packets in our experiment traversed 5,266 distinct Internet routers, passing through 1003 dialup access points in 653 major U.S. cities (see Figure 24) [156].

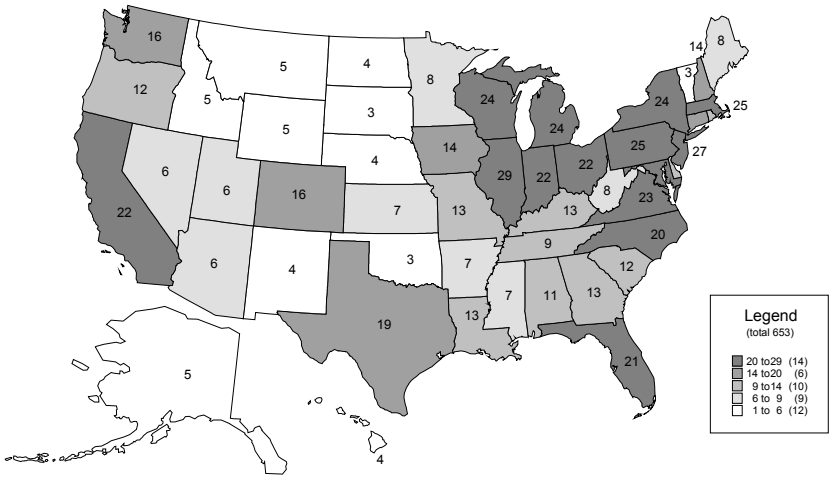

Figure 24. The number of cities per state that participated in the streaming experiment.

4.3.2 RTT Measurement

In order to maintain an RTO estimator, the receiver in a real-time session must periodically measure the round-trip delay. In our experiment, the client obtained RTT measurements by utilizing the following two methods. The first method used packet loss to measure the round-trip delay – each successfully recovered packet provided a sample of the RTT (i.e., the RTT was the duration between sending a NACK and receiving the corresponding retransmission). In order to avoid the ambiguity of which retransmission of the same packet actually returned to the client, the header of each NACK request and each retransmitted packet contained an extra field specifying the retransmission attempt for that particular packet. Thus, the client was able to pair retransmitted packets with the exact times when the corresponding NACKs were sent to the server (i.e., Karn's [120] retransmission ambiguity problem was avoided).

The second method of measuring the RTT was used by the client to obtain *additional* samples of the round-trip delay in cases when network packet loss was too low. The method involved periodically sending *simulated* retransmission requests to the server if packet loss was below a certain threshold. In response to these simulated NACKs, the server included the usual overhead[25] of fetching the needed packets from the storage and sending them to the client. Note that even though we call these retransmissions "simulated," the round-trip delays they generated were 100% real and the use of these RTTs in updating the RTO estimator was fully justified. During the experiment, the client activated simulated NACKs, spaced 30 seconds apart, if packet loss was below 1%.

Note that all NACKs were sent using UDP, which made them susceptible to packet loss as well. Further discussion of the sampled RTTs, heavy-tailed distributions of the RTT, and various "sanity checks" can be found in [156].

4.4 Performance

4.4.1 Retransmission Model

In real-time streaming, RTO estimation is necessary when the client supports multiple retransmission attempts per lost packet. After studying our traces, we found that 95.7% of all lost packets, which were recovered *before their deadline*, required a single retransmission attempt, 3.8% two attempts, 0.4% three attempts, and 0.1% four attempts. These results are important for two reasons.

---

[25] Server logs showed that the overhead was below 10 ms for all retransmitted packets.

First, 4.3% of all lost packets in our experiment could not be recovered with a single retransmission attempt. Even though it does not seem like a large number, our experiments with MPEG-4 indicate that there is no "acceptable" number of underflow events that a user of a real-time video application can feel completely comfortable with, and therefore, we believe that each lost packet must be recovered with as much *reasonable* persistence as possible.

Furthermore, since the average packet loss during the experiment was only 0.5% [156], the majority of retransmitted packets were able to successfully arrive to the client. However, in environments with a much lower end-to-end delay and/or higher packet loss[26], the percentage of packets recovered with a single retransmission attempt will be much lower than 95.7%. Besides the obvious higher probability of losing a retransmission or a NACK (due to higher packet loss), the RTT in such environments is likely to be much lower than the startup delay, which naturally allows more retransmission attempts per lost packet before the packet's deadline. Therefore, the existence of paths with lower delays and higher packet loss provides a strong justification for using more than one per-packet retransmission attempt in future streaming applications.

Second, our trace data show that if a lost packet in our experiment was successfully recovered *before its deadline*, the recovery was performed in no more than four attempts. The latter observation is used in our retransmission model (described later in this section) to limit the number of per-packet retransmission attempts (which we call

R_{max}) to four. Note that this limit applies only to the collected traces and is not an inherent restriction of our model.

Ideally, an RTO estimator should be able to predict the exact value of the next round-trip delay. However, in reality, it is quite unlikely that any RTO estimator would be able to do that. Hence, there will be times when the estimator will predict smaller, as well as larger values than the next RTT. To quantify the deviation of the RTO estimate from the real value of the RTT, we utilize the following methodology.

Imagine that we sequentially number all *successfully recovered* packets in the trace (excluding simulated retransmissions) and let rtt_k be the value of the round-trip delay produced by the k-th successfully recovered packet at time t_k (see Figure 25). Note that we distinguish rtt_k from RTT_i, where the latter notation includes RTT samples generated by simulated retransmissions, and former one does not.

---

[26] For example, in certain DSL experiments with higher average packet loss, only 70% of the lost packets were recovered using one retransmission attempt.

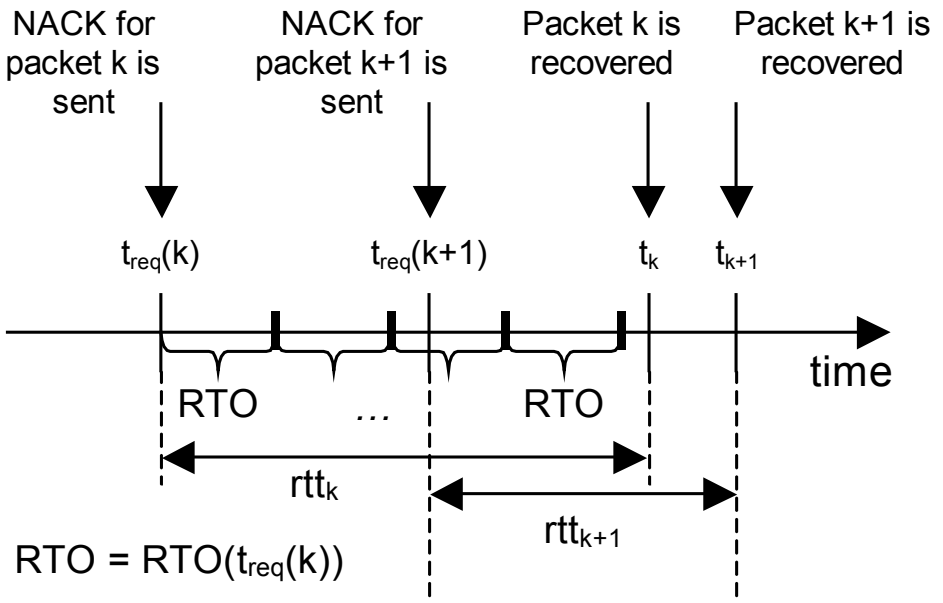

Figure 25. Operation of an RTO estimator given our trace data.

In Figure 25, the effective RTO for recovered packet k is computed at the time of the retransmission request, i.e., at time $t_{req}(k) = t_k - rtt_k$. Therefore, assuming that $RTO(t)$ is the value of the retransmission timeout at time t and assuming that the client uses the *latest* value of the RTO for each subsequent retransmission of a particular lost packet, it makes sense to examine how well the value of the RTO at the time of the request, $RTO(t_{req}(k))$, predicts the real value of the round-trip delay rtt_k. Hence, the accuracy of an RTO estimator in predicting the RTT of lost packets based on our trace data can be established by computing the *timeout waiting factor* w_k for each successfully recovered packet k in the trace:

$$w_k = \frac{RTO(t_k - rtt_k)}{rtt_k}. \tag{9}$$

Note that although our model does not use RTT samples measured by simulated retransmissions in computing w_k's (because they do not represent an actual loss), it uses them in updating the RTO estimator.

Since the exact effect of overestimation and underestimation of the RTT depends on whether the first retransmission of a particular packet was lost or not (and in some cases on whether subsequent retransmissions were lost or not), we simplify the problem and study the performance of RTO estimators assuming the worst case: values of w_k less than 1 always indicate that the estimator would have tried (if not limited by R_{max}) to produce $\lfloor rtt_k / RTO(t_k - rtt_k) \rfloor = \lfloor 1/w_k \rfloor$ duplicate packets given our trace data (i.e., assuming that all retransmissions arrived to the client), and values of w_k greater than 1 always indicate that the estimator would have waited longer than necessary before detecting that a subsequent retransmission was needed (i.e., assuming that the first retransmission initiated at time $t_{req}(k)$ was lost). In Figure 25, given our assumptions, the RTO estimator generates four (i.e., $\lfloor 1/w_k \rfloor$) duplicate packets while recovering packet k.

The negative effects of duplicate packets (i.e., wasted bandwidth and aggravation of congestion) are understood fairly well. On the other hand, the exact effect of unnecessary timeout waiting in real-time applications depends on a particular video stream (i.e., the decoding delay of each frame), video coding scheme (i.e., the type of motion compensation, scalability, and transform used), individual lost packets (i.e., which frames they belong to), and the video startup delay.

Nevertheless, we can make a generic observation that RTO estimators with higher timeout overwaiting factors w_k suffer a lower probability of recovering a lost packet and

consequently incur more underflow events. To keep our results universal and applicable to any video stream, we chose not to convert w_k's into the probability of an underflow event (or any other performance metric related to the video quality), and instead, study the tradeoff between a generic *average timeout overwaiting factor w* and the *percentage of duplicate packets d*:

$$w = \frac{1}{N_+} \sum_{w_k \geq 1} w_k \ , \tag{10}$$

$$d = \frac{1}{N} \sum_{w_k < 1} \min\left(\left\lfloor \frac{1}{w_k} \right\rfloor, R_{max} \right), \tag{11}$$

where N_+ is the number of times the RTO overestimated the next RTT (i.e., the number of times w_k was greater than or equal to 1) and N is the total number of lost packets. Parameter w is always above 1 and represents the average factor by which the RTO overestimates the RTT. Parameter d is the percentage of duplicate packets (relative to the number of lost packets) generated by the RTO estimator assuming that all requested retransmissions successfully arrived to the client.

In addition, we should note that the use of exponential backoff[27] instead of R_{max} provides similar, but numerically different results. However, in order to properly study the tradeoff between exponential backoff and R_{max}, our model must take into account re-

---

[27] In which case, (11) should read $d = \frac{1}{N} \sum_{w_k < 1} \left\lfloor \log_2\left(\frac{1}{w_k} + 1 \right) \right\rfloor$.

transmission attempts beyond the first one and study the probability of an underflow event in that context (i.e., the model must include a video coding scheme, video sequence, particular lost packets, and an actual startup delay). We consider such analysis to be beyond the scope of this thesis.

Finally, we should point out that all RTO estimators under consideration in this chapter depend on a vector of tuning parameters $\mathbf{a} = (a_1, \ldots, a_n)$. For example, the class of TCP-like RTO estimators in (8) can be viewed as a function of four tuning parameters α, β, k, and n. Therefore, the goal of the minimization problem that we define in the next section is to select such vector **a** that optimizes the performance of a particular RTO estimator $RTO(\mathbf{a}; t)$. By the word *performance* throughout this chapter, we mean tuple (d,w) defined in (10) and (11).

4.4.2 Optimality and Performance

As we mentioned before, the problem of estimating the RTT is different from simply minimizing the deviation of the predicted value $RTO(\mathbf{a}; t_k - rtt_k)$ from the observed value rtt_k. If that were the case, we would have to solve a well-defined least-squares minimization problem (i.e., the maximum likelihood estimator):

$$\min_{(a_1,\ldots,a_n)} \sum_k \left(RTO(\mathbf{a}; t_k - rtt_k) - rtt_k\right)^2 . \tag{12}$$

The main problem with the maximum likelihood estimator (MLE) lies in the fact that the MLE cannot distinguish between over and underestimation of the RTT, which

allows the MLE to assign equal cost to estimators that produce a substantially different number of duplicate packets.

Instead, we introduce two *performance functions* $\mathbf{H}(\mathbf{a})$ and $G(\mathbf{a})$ and use them to judge the accuracy of RTO estimators in the following way. We consider tuning parameter $\mathbf{a}_{opt}$ of an RTO estimator to be "optimal" within tuning domain S of the estimator ($\mathbf{a}_{opt} \in S$), if $\mathbf{a}_{opt}$ minimizes the corresponding performance function (i.e., either $\mathbf{H}$ or G) within domain S. Later in this section, we will show that given the classes of RTO estimators studied in this chapter and given our experimental data, the two performance measures (i.e., functions) produce equivalent results. Note that "optimality" is meaningful only within a given class of estimators, its tuning domain S, and the trace data used in the simulation.

In the first formulation, our goal is to minimize an *RTO performance vector-function* $\mathbf{H}(\mathbf{a}) = (d(\mathbf{a}), w(\mathbf{a}))$:

$$\min_{\mathbf{a} \in S} \mathbf{H}(\mathbf{a}) = \min_{\mathbf{a} \in S} \big(d(\mathbf{a}), w(\mathbf{a})\big). \tag{13}$$

For the minimization problem in (13) to make sense, we must also define vector comparison operators *greater than* and *less than*. The following are a natural choice:

$$(d_1, w_1) < (d_2, w_2) \Leftrightarrow \big((d_1 < d_2) \wedge (w_1 \leq w_2)\big) \vee \big((d_1 \leq d_2) \wedge (w_1 < w_2)\big), \tag{14}$$

$$(d_1, w_1) > (d_2, w_2) \Leftrightarrow \big((d_1 > d_2) \wedge (w_1 \geq w_2)\big) \vee \big((d_1 \geq d_2) \wedge (w_1 > w_2)\big), \tag{15}$$

and otherwise we consider tuples (d_1,w_1) and (d_2,w_2) to be *equivalent*. Figure 26 illustrates the above operators for a given RTO estimator and provides a graphical map-

ping between the performance of an RTO estimator and points on a 2-D plane. The shaded convex area in Figure 26 is the range of a hypothetical RTO estimator, where the range is produced by varying tuning parameter **a** within the estimator's tuning domain S (i.e., the convex area consists of points $\mathbf{H}(\mathbf{a})$, $\forall \mathbf{a} \in S$). Given a particular point $D = (d,w)$ in the range, points to the left and down from D (e.g., D_1) clearly represent a better estimator; points to the right and up from D (i.e., D_3) represent a worse estimator; and points in the other two quadrants may or may not be better (i.e., D_2 and D_4).

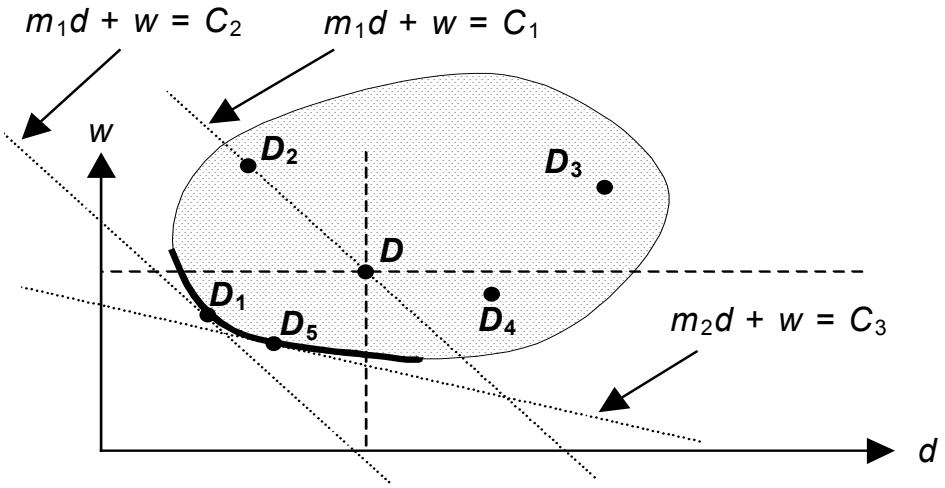

Figure 26. Comparison between RTO performance vector points (d,w).

In order to help us understand which performance points in Figure 26 are optimal, we define the *optimal RTO curve* to be such points in the (d,w) space, produced by the RTO estimator, that are *less than or equal* to any other point produced by the RTO estimator, i.e., all points $(d_{opt},w_{opt}) = \mathbf{H}(\mathbf{a}_{opt})$, $\mathbf{a}_{opt} \in S$, such that $\forall \mathbf{a} \in S$: $\mathbf{H}(\mathbf{a}_{opt}) \leq \mathbf{H}(\mathbf{a})$. In Figure 26, the optimal RTO curve is shown in bold along the left bottom side of the

shaded area. Hence, finding the set of tuning parameters **a** that map to the optimal RTO curve is equivalent to solving the minimization problem in (13).

In the second formulation, we can state the problem of finding a better RTO estimator as that of minimizing a weighted sum of the percentage of duplicate packets d and the average overwaiting factor w (similar methods are frequently used in rate-distortion theory). The problem in the new formulation is easier to solve since it involves the minimization of a *scalar* function instead of a *vector* function. In addition, our reformulation allows us to decide on the exact relationship between *equivalent* points (i.e., in cases when neither (14) nor (15) holds) by assigning proper weight to one of the parameters in the (d,w) tuple.

Hence, we define a *weighted RTO performance function* $G(\mathbf{a}, M)$ as following:

$$G(\mathbf{a}, M) = M \cdot d(\mathbf{a}) + w(\mathbf{a}),\ 0 \le M < \infty, \tag{16}$$

where M is a weight, which assigns desired importance to duplicate packets d (large M) or overwaiting factor w (small M). As we will see below, by running M through a range of values and optimizing $G(\mathbf{a}, M)$ for each weight M, we can build the optimal RTO curve; however, the actual values of M used to build the curve are not important.

Note that using performance function G we can unambiguously establish a relationship between *equivalent* points in the (d,w) space, given a certain weight M (i.e., points **a** with smaller $G(\mathbf{a}, M)$ are better). Specifically, for each weight M and for any constant $C > 0$, there exists a *performance equivalence* line $Md + w = C$, along which all points (d,w) are *equal* given the performance function in (16); points below the line are

better (i.e., they belong to lines with smaller C); and points above the line are worse. In Figure 26, two parallel lines are drawn for $M = m_1$ using two different values of the constant ($C_2 < C_1$). Given weight m_1, point D_2 is now *equal* (not just equivalent) to D, point D_1 is still better, point D_3 is still worse, while point D_4 is now also worse.

In addition, not only is point D_1 better than D given performance function $G(\mathbf{a}, M)$ and weight m_1, but D_1 is also the "optimal" point of the RTO estimator in Figure 26 for weight m_1, i.e., point D_1 minimizes function (16) for weight m_1 within tuning domain S. In other words, to graphically minimize function $G(\mathbf{a}, M)$ for any weight M, one needs to slide the performance equivalence line $Md + w$ as far left and down as possible, while maintaining the contact with the range of the RTO estimator.

Notice how point D_1 found by minimizing function $G(\mathbf{a}, M)$ lies on the optimal RTO curve earlier defined using the performance measure in (13). We can further generalize this observation by saying that if the optimal RTO curve is given by a convex continuous function similar to the one in Figure 26, all points that optimize the weighted performance function $G(\mathbf{a}, M)$ will lie on the optimal RTO curve (and vice versa).

Consequently, using intuition, we can attempt to build the entire optimal RTO curve out of points $D_{opt}(M) = (d_{opt}(M), w_{opt}(M))$, where $d_{opt}(M)$ and $w_{opt}(M)$ are the result of minimizing $G(\mathbf{a}, M)$ for a particular weight M. For example, from Figure 26, we can conclude that optimal point $D_{opt}(m_1)$ is given by D_1 and optimal point $D_{opt}(m_2)$ is given by D_5. Hence, by varying M in $D_{opt}(M)$ between zero (flat performance equivalence line) and infinity (vertical performance equivalence line) we can produce (ideally) any point along the optimal RTO curve.

Note that we view the above retransmission model and both performance measures as an important contribution of this work. These techniques can be used to study the performance of RTO estimators in other datasets and even in ACK-based protocols (with properly taking into account exponential timer backoff as shown in section 4.4.1). The rest of the chapter describes how our model and performance functions can be applied to the traces of our wide-scale Internet experiment [156] and discusses the important lessons learned.

Now we are ready to plot the values of vector function **H**(**a**) for different values of the tuning parameter $\mathbf{a} = (a_1, \ldots, a_n)$ in different RTO estimators, as well as identify the optimal points and understand which values of parameter **a** give us the best performance. Throughout the rest of the chapter, in order to conserve space, we show the results derived from streaming traces through ISP_a (129,656 RTT samples). Streaming data collected through the other two ISPs produce similar results.

4.5 TCP-like Estimators

4.5.1 Performance

We start our analysis of RTO estimators with a generalized TCP-like RTO estimator given in (8). We call this estimator RTO_4, because its tuning parameter **a** consists of four variables: $\mathbf{a} = (\alpha, \beta, k, n)$. Recall that $\mathbf{a}_{TCP} = (0.125, 0.25, 4, 1)$ corresponds to Jacobson's RTO [110] and $\mathbf{a}_{793} = (0.125, 0, 0, 2)$ corresponds to the RFC 793 RTO [221].

In order to properly understand which parameters in (8) contribute to the improvements in the performance of the TCP-like estimator, we define two *reduced* RTO

estimators depending on which tuning parameters (α, β, k, n) are allowed to vary. In the first reduced estimator, which we call RTO_2, we use only (α,β) to tune its performance, i.e., $\mathbf{a} = (\alpha, \beta, 4, 1)$. In the second reduced estimator, which we call RTO_3, we additionally allow k to vary, i.e., $\mathbf{a} = (\alpha, \beta, k, 1)$.

Figure 27 shows the optimal RTO_4 curve and the range of values $\mathbf{H}(\mathbf{a})$ produced by both reduced estimators. The ranges of RTO_2 (900 points) and RTO_3 (29,000 points) were obtained by conducting a uniform exhaustive search of the corresponding tuning domain S, and the optimal RTO_4 curve was obtained by extracting the minimum values of $\mathbf{H}(\mathbf{a})$ after a similar exhaustive search through more than 1 million points. In addition, Figure 27 shows the performance of Jacobson's RTO estimator, $\mathbf{H}(\mathbf{a}_{TCP}) = (12.63\%, 4.12)$, by a square and the performance of the RFC 793 RTO estimator, $\mathbf{H}(\mathbf{a}_{793}) = (15.34\%, 2.84)$, by a diamond. Clearly, Jacobson's and the RFC 793 RTO estimators are equivalent, since neither one is located below and to the left of the other.

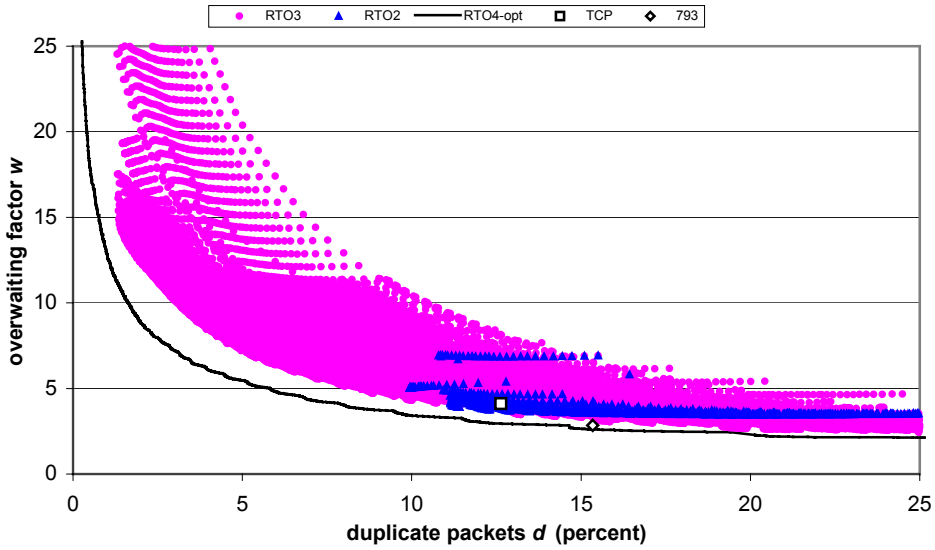

Figure 27. Performance of TCP-like estimators.

The performance of RTO estimators in Figure 27 certainly gets better with the increase in the number of free tuning variables. For a given average overwaiting factor $w = 4.12$, RTO_2 and RTO_3 both achieve optimality in the same point and offer only a slight improvement in the number of duplicate packets over TCP RTO – 11.15% compared to 12.63%. RTO_4, however, offers a more substantial improvement, generating only $d = 7.84\%$ duplicate packets.

Furthermore, Figure 27 shows that the optimal RTO_4 curve (built by the exhaustive search) is convex and fairly continuous until approximately 20% duplicate packets. Consequently, we can build another optimal RTO_4 curve using scalar weighted performance function $G(\mathbf{a})$ and compare the results with those in Figure 27. A scalar function such as $G(\mathbf{a})$ allows us to use various numerical multidimensional minimization methods, which usually do not work with vector functions. In addition, we find that numerical op-

timization methods produce points along the optimal RTO curve with more accuracy than the exhaustive search (given a reasonable amount of time) and with fewer computations of functions $d(\mathbf{a})$ and $w(\mathbf{a})$ (i.e., faster).

To verify that weighted performance function $G(\mathbf{a})$ does in fact produce the same optimal RTO_4 curve, we focused on the following minimization problem for a range of values of weight M:

$$\min_{\mathbf{a}\in S} G(\mathbf{a}, M) = \min_{\mathbf{a}\in S}\left(M \cdot d(\mathbf{a}) + w(\mathbf{a})\right) \tag{17}$$

The fact that function $G(\mathbf{a}, M)$ has unknown (and non-existent) partial derivatives $\partial G(\mathbf{a}, M)/\partial a_k$ suggests that we are limited to numerical optimization methods that do not use derivatives. After applying the Downhill Simplex Method in Multidimensions (due to Nelder and Mead [188]) and quadratically convergent Powell's method [36], we found that the former method performed significantly better and arrived at (local) minima in fewer iterations. To improve the found minima, we discovered that restarting the Simplex method in random locations in the N-dimensional space ten times per weight M produced very good results.

Figure 28 shows the points built by the Downhill Simplex method for the RTO_4 estimator (each point corresponds to a different weight M) and the corresponding optimal RTO_4 curve previously derived from the exhaustive search. As the figure shows, points built by Downhill Simplex are no worse (and often slightly better) than those found in the exhaustive search.

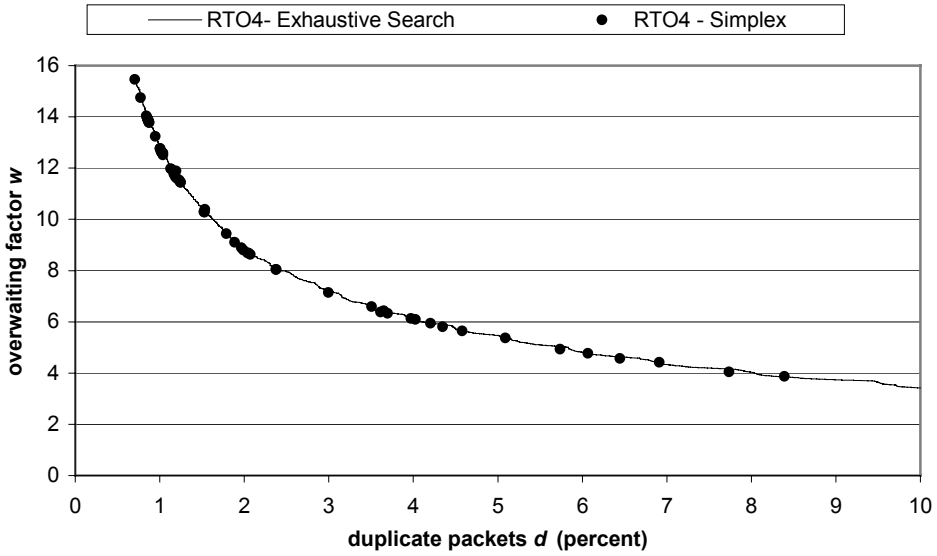

Figure 28. Points built by Downhill Simplex and the exhaustive search in the optimal RTO_4 curve.

Interestingly, the optimal curves in Figure 28 resemble power functions in the form of:

$$w_{opt} = C(d_{opt})^{-p}, p > 0. \quad (18)$$

To investigate this observation further, Figure 29 replots the points of the Downhill Simplex curve from Figure 28 on a log-log scale with a straight line fitted to the points. A straight line provides an excellent fit (with correlation 0.99) and suggests that the optimal RTO curve could be modeled as a power function (18) with $C = 1.022$ and $p = 0.55$.

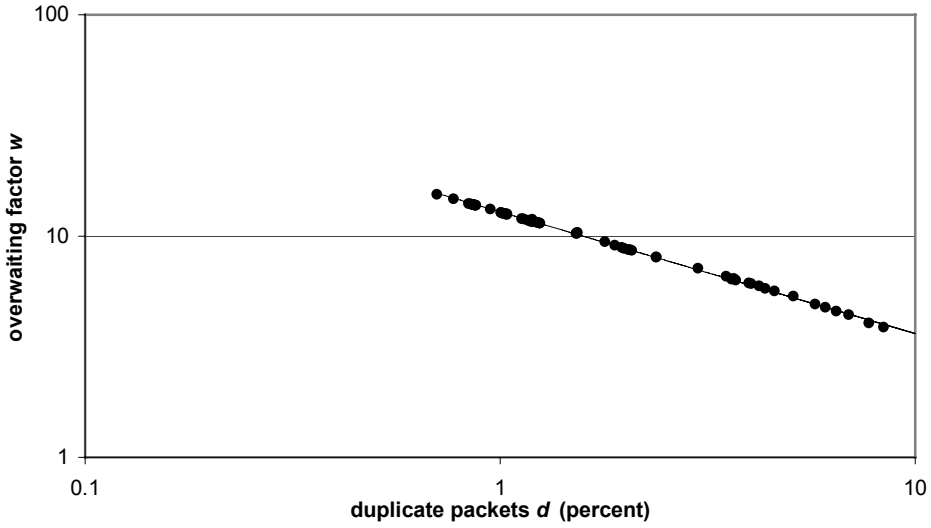

Figure 29. Log-log plot of the optimal (Simplex) RTO_4 curve.

Assuming that the relationship between w and d in the optimal RTO_4 curve is a power function (18), we can now analytically compute optimal points (d_{opt},w_{opt}) that minimize function $G(\mathbf{a})$ for a given weight M. Rewriting (16) using the function from (18), taking the first derivative, and equating it to zero we get:

$$\frac{\partial G(\mathbf{a})}{\partial d_{opt}} = \frac{\partial}{\partial d_{opt}}\left(M d_{opt} + C d_{opt}^{-p}\right) = M - Cp d_{opt}^{-p-1} = 0\,. \tag{19}$$

Solving (19) for d_{opt} and using (18) one more time, we can express the optimal values of both the number of duplicate packets d_{opt} and the average overwaiting factor w_{opt} as a function of weight M:

$$d_{opt} = \left(\frac{Cp}{M}\right)^{\frac{1}{p+1}} \text{ and } w_{opt} = \frac{M}{p}\left(\frac{Cp}{M}\right)^{\frac{1}{p+1}}. \tag{20}$$

4.5.2 Tuning Parameters

In this section, we provide a reverse mapping from optimal performance points **H**(**a**) in Figure 28 to points **a** in tuning domain *S* (i.e., describe how to construct optimal RTO_4 estimators). While analyzing RTO_2, we noticed that for each given β, larger values of α produced fewer duplicate packets, as well as that for each fixed value of α, smaller values of β similarly produced fewer duplicate packets. To further study this phenomenon, we examined the correlation between the RTO_2 estimates and the corresponding future round-trip delays rtt_k for different values of (α,β). Interestingly, the highest correlation was reached in point (1.0, 0.044), which suggests that an RTO estimator with (α,β) fixed at (1,0) should provide estimates with a reasonably high correlation with the future RTT, as well as that it could be possible to achieve the values of the optimal RTO_4 curve by just varying parameters *n* and *k* in RTO_4.

To investigate this hypothesis, we constructed another reduced estimator called $RTO_{4(1,0)}$, which is produced by RTO_4 at input points (1, 0, *k*, *n*). The results of an exhaustive search of the reduced tuning domain (*k*, *n*) for $RTO_{4(1,0)}$ are plotted in Figure 30 (lightly shaded area). As the figure shows, the optimal RTO_4 curve (shown as squares in Figure 30) touches the range of $RTO_{4(1,0)}$, which means that the reduced estimator can achieve the points along the optimal RTO_4 curve while keeping α and β constant. This fact implies that it is not necessary to maintain a smoothed RTT average to achieve optimality within our datasets, because α = 1.0 means that the *SRTT* always equals the latest RTT sample.

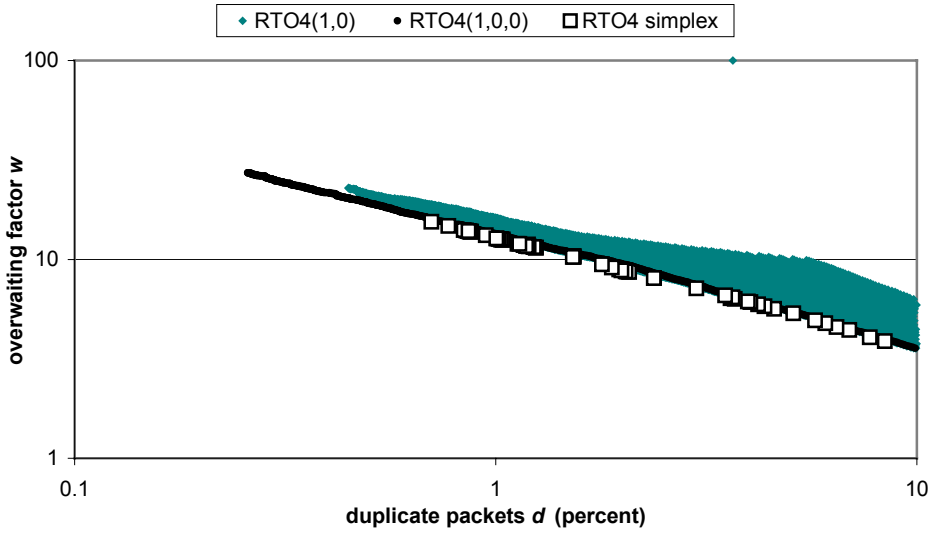

Figure 30. RTO_4-Simplex and two reduced RTO_4 estimators on a log-log scale.

The next logical step is to question the need for *SVAR* in RTO_4 since *SVAR* turns out to be a constant when β equals zero. In the same Figure 30, we plotted an additional optimal curve for estimator $RTO_{4(1,0,0)}$, which represents RTO_4 at input points (1, 0, 0, *n*). As the figure shows, all values of the $RTO_{4(1,0,0)}$ estimator lie next to the optimal curve as opposed to many sub-optimal points produced by $RTO_{4(1,0)}$. At the end of this section, we discuss the explanation of why smoothing of RTT samples does not increase the accuracy of RTO_4, but first show how to construct an $RTO_{4(1,0,0)}$ estimator with a given performance.

A straight line fitted to the $RTO_{4(1,0,0)}$ curve in Figure 30 produces a power function (18) with $C = 1.07$ and $p = 0.546$. Further investigation discovered that there is a strong linear dependency between the optimal value of n_{opt} in $RTO_{4(1,0,0)}$ and the optimal value of the average overwaiting factor w_{opt}:

$$n_{opt} = mw_{opt} + b, \tag{21}$$

where $m = 0.86$ and $b = -0.13$. Since we already know the dependency between w_{opt} and d_{opt} in (18), we can derive the relationship between n_{opt} and d_{opt} in $RTO_{4(1,0,0)}$:

$$n_{opt} = mC(d_{opt})^{-p} + b. \tag{22}$$

Consequently, (22) can be used to build optimal $RTO_{4(1,0,0)}$ estimators given any desired value of duplicate packets d_{opt}. For example, if an application specifies that the maximum number of duplicate packets it is willing to tolerate is $d_{opt} = 2\%$, using (18), the optimal overwaiting factor w_{opt} is 9.12 (the corresponding weight M is 248) and using (22), the optimal RTO estimator is given by $RTO_{4(1,0,0)}$ with $n_{opt} = 7.31$.

4.5.3 Discussion

This is the point when we must address a major conceptual difference between ACK and NACK-based retransmission schemes, as well as point out several properties of our experiment. The difference between ACK and NACK-based protocols lies in the fact that NACK-based applications obtain RTT samples only upon packet loss, while ACK-based applications consistently obtain RTT samples on a per-packet basis. Consequently, the distance between RTT samples in a NACK-based application is often large and fluctuates widely (i.e., between tens of milliseconds and tens of seconds). Given a low average packet loss of 0.5% during our Internet experiment, the average distance between consecutive RTT samples in our datasets was 15.7 seconds.

Hence, we observed that NACK-based protocols in the presence of low packet loss greatly undersample the RTT process, and further smoothing of already rare RTT samples with EWMA formulas produces a very sluggish and slow-responding moving average. Such moving average in the form of (1) and (7) can rarely keep up with the actual RTT and turns out to be a poor predictor of the future values of the round-trip delay. This observation represents the first major conclusion of our study – *NACK-based protocols in our experiment combined with low-frequency RTT sampling (i.e., low packet loss) required a different RTO estimation method than the classical Jacobson's RTO; specifically, smoothed averaging of RTT samples proved to be hurtful, and the latest RTT sample turned out to be the best predictor of the future RTTs.*

4.6 Jitter-Based Estimators

4.6.1 Structure and Performance

The second class of RTO estimators, which we call RTO_J, is derived from $RTO_{4(1,0,0)}$ by adding to it a smoothed variance of the inter-packet arrival delay (quantified later in this section). As we will show below, RTO_J reduces the number of duplicate packets in our trace data compared to RTO_4 by up to 60%.

The receiver in a real-time protocol usually has access to a large number of delay jitter samples between the times when it measures the RTT. It would only be logical to utilize tens or hundreds of delay jitter samples between retransmissions to fine-tune RTO estimation. This fine-tuning is receiver-oriented and is not available to TCP senders (which they do not need since TCP obtains a substantial amount of RTT samples through

its ACK-based operation). In fact, TCP's ability to derive an RTT sample from (almost) each ACK gave it an advantage that may now be available to NACK-based protocols in the form of delay jitter.

Before we describe our computation of delay jitter, we must introduce the notion of a packet burst. In practice, many real-time streaming servers are implemented to transmit their data in bursts of packets [173], [225], [226] instead of sending one packet every so many milliseconds. Although the latter is considered to be an ideal way of sending video traffic by many researchers (e.g., [86]), in practice, there are limitations that do not allow us to follow this ideal model [156].

In our server, we implemented bursty streaming with the burst duration D_b (i.e., the distance between the first packets in successive bursts) varying between 340 and 500 ms depending on the streaming bitrate (for comparison, RealAudio servers use D_b = 1,800 ms [173]). Each packet in our real-time application carried a burst identifier, which allowed the receiver to distinguish between packets from different bursts. After analyzing the traces, we found that *inter-burst* delay jitter had more correlation with the future RTT than *inter-packet* delay jitter (we speculate that one of the reasons for this was that more cross traffic was able to queue between the bursts than between individual packets).

To be more specific, suppose for each burst j, the last packet of the burst arrived to the client at time $t_{last}(j)$, and the first packet of the burst arrived at time $t_{first}(j)$. Consequently, the *inter-burst delay* for burst j is defined as:

$$\Delta_j = t_{first}(j) - t_{last}(k), j \geq 1 \tag{23}$$

where burst k is the last burst received before burst j (unless there is packet loss, $k = j - 1$). For each burst, using EWMA formulas similar to those in TCP, we compute *smoothed inter-burst delay* $S\Delta_j$ and *smoothed inter-burst delay variance* $SVAR\Delta_j$:

$$S\Delta_j = \begin{cases} \Delta_1, & j = 1 \\ (1-\alpha_1)\cdot S\Delta_{j-1} + \alpha_1 \cdot \Delta_j, & j \geq 2 \end{cases} \tag{24}$$

and

$$SVAR\Delta_j = \begin{cases} \Delta_1 / 2, & j = 1 \\ (1-\beta_1)\cdot SVAR\Delta_{j-1} + \beta_1 \cdot VAR\Delta_j, & j \geq 2 \end{cases}, \tag{25}$$

where α_1 and β_1 are exponential weights, and $VAR\Delta_j$ is the absolute deviation of Δ_j from its smoothed version $S\Delta_{j-1}$. In our experience, $S\Delta_j$ is usually proportional to burst duration D_b and thus, cannot be used the same way in real-time applications with different burst durations. On the other hand, smoothed variance $SVAR\Delta_j$ is fairly independent of the burst duration and reflects the variation in the amount of cross traffic in router queues along the path from the server to the client.

Given our definition of delay variation in (25), suppose that t_i is the time when our trace recorded the i-th RTT sample (including simulated retransmissions), then the effective *jitter-based* RTO at time t is:

$$RTO_J(t) = n \cdot RTT_i + m \cdot SVAR\Delta_j, \tag{26}$$

where $i = \max i$: $t_i \leq t$ and $j = \max j$: $t_{first}(j) \leq t$.

Figure 31 compares the performance of the RTO_J estimator with that of RTO_4 (both optimal curves were built using the Downhill Simplex method). Given a particular value of the average overwaiting factor w, RTO_J offers a 45-60% improvement over RTO_4 in terms of duplicate packets. Recall that for an average overwaiting factor w = 4.12, Jacobson's RTO estimator produced 12.63% duplicate packets and RTO_4 achieved 7.84%. At the same time, RTO_J is now able to improve this value to 3.25%.

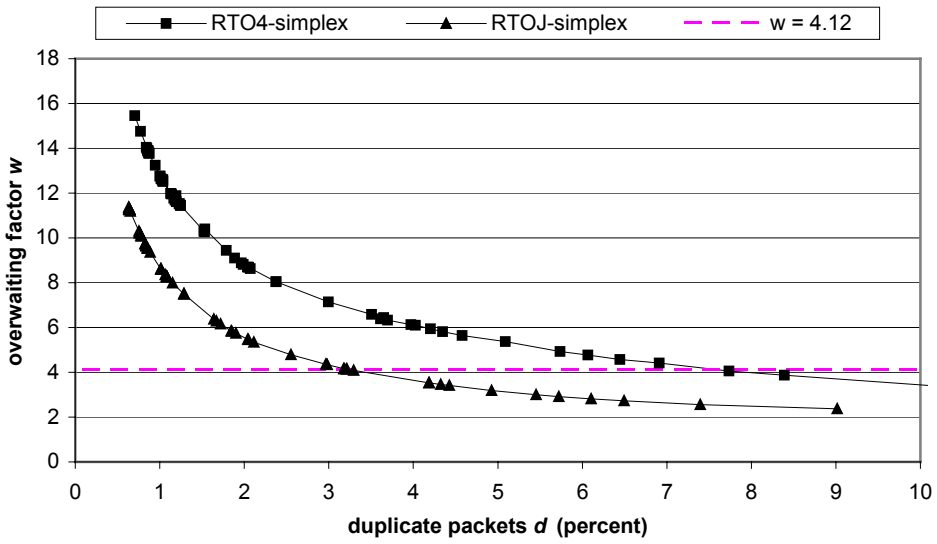

Figure 31. The jitter-based RTO estimator compared with the RTO_4 estimator.

4.6.2 Tuning Parameters

RTO_J contains four tuning variables $\mathbf{a} = (\alpha_1, \beta_1, m, n)$, just like the RTO_4 estimator. This time, however, the performance of the estimator does not strongly depend on the first two variables. Several values in the proximity of $\alpha_1 = 0.5$ give optimal performance. For β_1, the optimal performance is achieved at $\beta_1 = 0.125$, which is helpful if $SVAR\Delta_j$ is

to be computed using only integer arithmetics. Just as in the $RTO_{4(1,0,0)}$ estimator, (α_1,β_1) can be fixed at their optimal values and the optimal RTO_J curve can be entirely built using n and m.

To further reduce the number of free variables in jitter-based estimators, we examined the relationship between n_{opt} and m_{opt} in the optimal RTO_J curve shown in Figure 31. Although the relationship is somewhat random, there is an obvious linear trend, which fitted with a straight line (with correlation $\rho = 0.88$) establishes that function

$$m_{opt} = 4.2792 \cdot n_{opt} - 2.6646 \tag{27}$$

describes the optimal parameters n and m reasonably well. Consequently, we created a reduced estimator, which we call RTO_{J427}, by always keeping m as a function of n shown in (27) and compared its performance (by running n through a range of values) to that of RTO_J in Figure 32. As the figure shows, the reduced estimator RTO_{J427} reaches the corresponding optimal RTO_J curve with high accuracy.

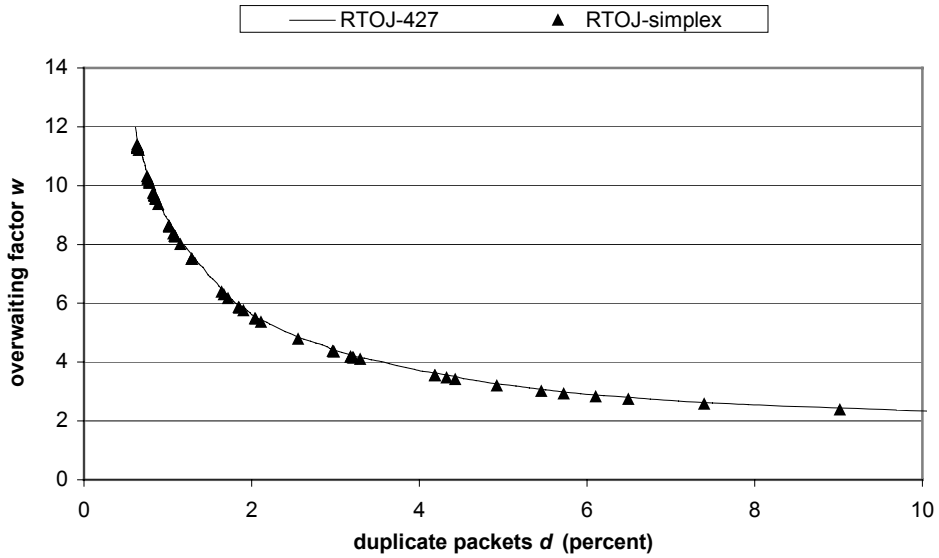

Figure 32. Reduced jitter-based estimator compared with the optimal RTO_J estimator.

Similar power functions (18) and (22) apply to the optimal RTO_J and RTO_{J427} curves. Table II summarizes the values of constants in both equations (18) and (22).

Part I. Power function for optimal RTO curves: $w_{opt} = C(d_{opt})^{-p}$.			
RTO estimator	***C***	***p***	**correlation**
RTO_4	1.02	0.5500	0.9994
$RTO_{4(1,0,0)}$	1.07	0.5456	0.9991
RTO_J	0.50	0.6158	0.9997
RTO_{J427}	0.53	0.6098	0.9991
Part II. Power function for optimal parameter n: $n_{opt} = C_1(d_{opt})^{-p} + C_2$.			
Reduced estimator	C_1	C_2	***p***
$RTO_{4(1,0,0)}$	0.88	–0.13	0.5456
RTO_{J427}	0.20	0.31	0.6098

Table II. Summary of constants in various power laws

Using the same example from section 4.5, for d_{opt} = 2%, we find that w_{opt} is 5.75 in RTO_{J427} (compared to 9.12 in $RTO_{4(1,0,0)}$). Given parameters in the second half of Table II, the value of n_{opt} in RTO_{J427} is 2.47 (compared to 7.31 in $RTO_{4(1,0,0)}$), and the value of

m_{opt} using (27) is 7.91. As we can see, the superior performance of the RTO_{J427} estimator over RTO_4 and $RTO_{4(1,0,0)}$ is achieved by placing lower weight on RTT samples and deriving more information about the network from the more frequent delay jitter samples.

Hence, we can summarize our second major conclusion as following – during the experiment, a NACK-based RTO estimator running over paths with low-frequency RTT sampling (over 10 seconds between samples) could be significantly improved by adding smoothed delay jitter to the scaled value of the latest RTT.

4.7 High-frequency sampling

The final question left to resolve is whether the performance of RTO_4 and RTO_J is different in environments with high-frequency RTT sampling. In NACK-based protocols, high-frequency RTT sampling comes either from high packet loss rates or from frequent congestion control messages exchanged between the client and the server (in the latter case, the frequency of sampling is approximately equal to one sample per RTT [86]).

This section investigates the performance of RTO_4 and RTO_J in several environments with high-frequency RTT sampling and verifies whether the conclusions reached in previous sections hold for such Internet paths. We only show the results based on trace data collected along a single Internet path; however, the observations made in this section were also verified along multiple other paths with relatively high packet loss, as well as in a *congestion-controlled* streaming application with once-per-RTT sampling of the round-trip delay.

In this section, we apply trace-driven simulation to the datasets collected between a symmetric DSL (SDSL) client and a video server placed at the City College of New York (CCNY) during December 2000. This setup is shown in Figure 33. The CCNY backbone connected to the Internet through the CUNY (City University of New York) backbone via a series of T3 links. The client's dedicated SDSL circuit operated at 1.04 mb/s in both directions. The end-to-end path between the client and the server contained 15 routers from five Autonomous Systems (AS). During the experiment, we used a video stream coded at the video bitrate of 80 kb/s (86 kb/s IP bitrate). The collected dataset contains traces of 55 million packets, or 26 GBytes of data, obtained during the period of three weeks.

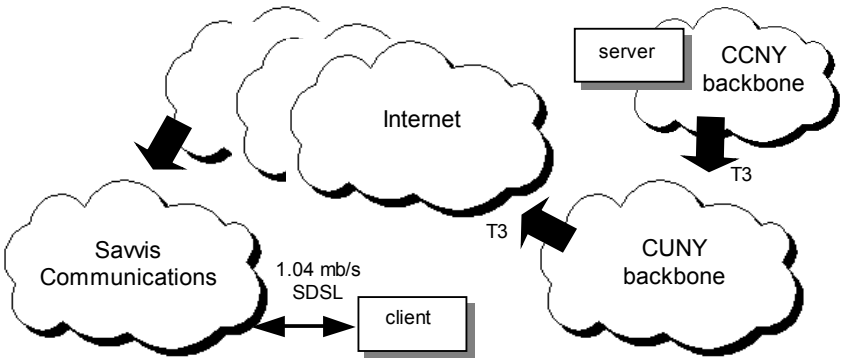

Figure 33. The setup of the high-speed experiment.

One interesting property of this end-to-end path is that the CUNY border router dropped a large fraction of packets during this experiment, regardless of the time of day or the sending rate of our flows. Thus, the average packet loss rate recorded in this trace was substantially higher than in the modem experiment (i.e., 7.4% vs. 0.5%), and the av-

erage delay between obtaining new RTT samples was only 740 ms, which is by a factor of 20 less than that in the wide-scale modem experiment.

Figure 34 shows the performance of the three estimators studied earlier in this chapter in the CUNY dataset. All three optimal curves were built using Downhill Simplex. As the figure shows, both RTO_4 and RTO_J achieve the same optimal performance, which means that the addition of delay jitter to already-frequent RTT samples is not as beneficial as previously discovered. In addition, note that $RTO_{4(1,0,0)}$ is no longer optimal within the dataset. Both results were expected, because the higher sampling frequency in the CUNY dataset allows RTO_4 to be a much better predictor than it was possible in the modem datasets.

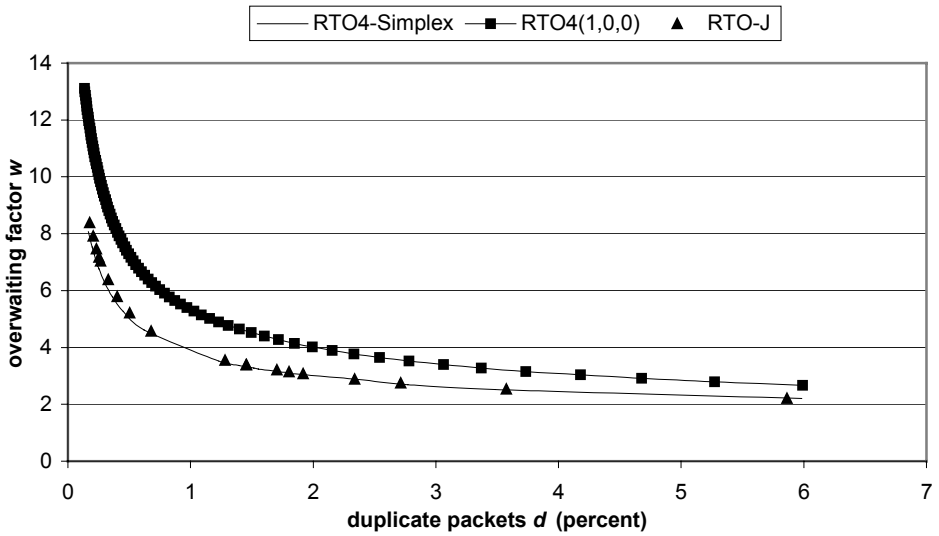

Figure 34. Performance of RTO_4, RTO_J and $RTO_{4(1,0,0)}$ in the CUNY dataset.

The final question that stands is what values of tuning variable **a** make RTO_4 optimal along paths with high-frequency RTT sampling (i.e., the CUNY dataset)? Our

analysis of the data shows that variance estimator *SVAR* is still redundant and that RTO_4 can be reduced to a simpler estimator, which this time assumes the following form: **a** = (α, 0, 0, n). Downhill Simplex optimization of RTO_4 shows that values of α between 0.12 and 0.13 are equally optimal and produce an estimator with performance equal to that of RTO_4. Note that Jacobson's value of $\alpha = 0.125$ falls within this range and agrees with the results derived from the CUNY dataset.

To verify that the reduced estimator $RTO_{4(0.125,\ 0,\ 0)}$ performs as well as RTO_4, we plotted both optimal RTO curves in Figure 35, which shows that the reduced estimator is almost identical to RTO_4.

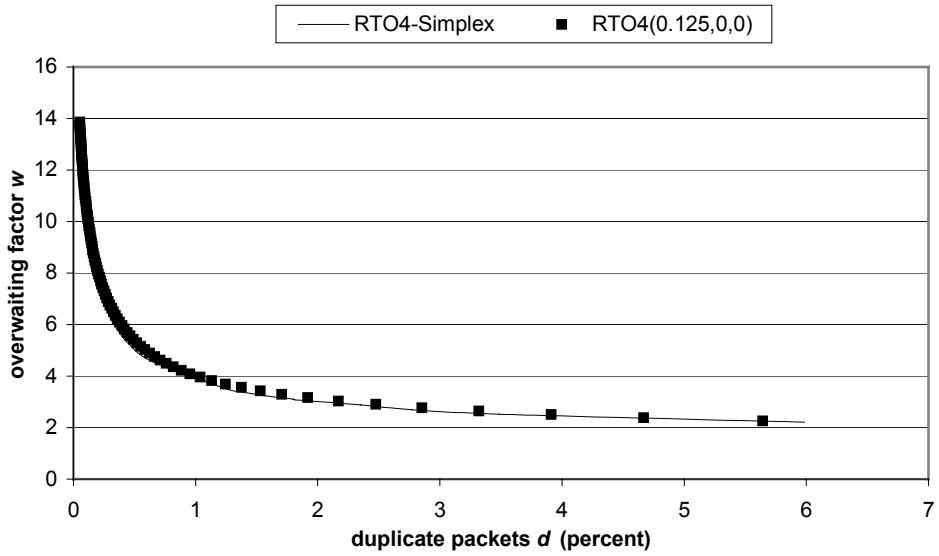

Figure 35. Performance of RTO_4 and $RTO_{4(0.125,0,0)}$ in the CUNY dataset.

Note that the above observations about the optimality of $RTO_{4(0.125,\ 0,\ 0)}$ were also found to hold when the CUNY server was replaced with a server located at Michigan State University, 21 hops from the client (the experiment was conducted in January 2001

and involved the transfer of over 17 million packets). Furthermore, similar results were obtained in various streaming tests over ISDN (over 77 million packets): in low packet-loss scenarios, RTO_J was significantly better than RTO_4, and $RTO_{4(1,0,0)}$ was optimal within the class of TCP-like estimators; however, in high packet loss scenarios, RTO_J did not offer much improvement over RTO_4.

We finish this section by reaching our third major conclusion – *in our experiments along paths with high-frequency RTT sampling, a simple smoothed round-trip delay estimator SRTT with parameter α between 0.12 and 0.13 was the optimal estimator, and neither delay jitter nor delay variance estimator SVAR provided any added benefits.*

PART II

Scalable Congestion Control for Real-time Streaming

Chapter Five

5 Scalability of Rate-based Congestion Control

Typically, NACK-based (i.e., rate-based) congestion control is dismissed as being not viable due to the common notion that "open-loop" congestion control is simply "difficult." Emerging real-time streaming applications, however, often rely on rate-based flow control and would benefit greatly from scalable NACK-based congestion control. This chapter sheds new light on the performance of NACK-based congestion control and measures the amount of "difficulty" inherently present in such protocols. We specifically focus on *increase-decrease* (I-D) congestion control methods for real-time, rate-based streaming. First, we introduce and study several new performance measures that can be used to analyze the class of general I-D congestion control methods. These measures include *monotonicity* of convergence to fairness and *packet-loss scalability* (explained later in the chapter). Second, under the assumptions that the only feedback from the network is

packet loss, we show that AIMD is the only TCP-friendly method with monotonic convergence to fairness. Furthermore, we find that AIMD possesses the best packet-loss scalability among all TCP-friendly binomial schemes [13] and show how poorly all of the existing methods scale as the number of flows is increased. Third, we show that if the flows can obtain the knowledge of an additional network parameter (i.e., the bottleneck bandwidth), the scalability of AIMD can be substantially improved. We conclude the chapter by studying the performance of a new scheme, called *Ideally-Scalable Congestion Control* (ISCC), both in simulation and a NACK-based MPEG-4 streaming application over a Cisco testbed.

5.1 Introduction

Congestion is an inherent property of the currently best-effort Internet. Consequently, transport protocols (such as TCP) commonly implement *congestion control*, which refers to end-to-end algorithms executed by a protocol in order to properly adapt the sending rate of a network flow to the available bandwidth in the path along which the flow sends its packets. Protocols with ACK-based flow control utilize one or another version of TCP-friendly congestion control, which includes Jacobson's modifications to TCP [5], [110], TCP-like congestion control (e.g., [239]), increase-decrease algorithms (e.g., [13], [14], [49], [81], [125], [141], [186], [286]), and equation-based methods (e.g., [86], [196]). These algorithms are shown to work well in the environment where the

sender relies on "*self-clocking*," which refers to the use of positive acknowledgements in congestion control.

However, current real-time streaming applications in the Internet [232] typically rely on NACK-based (i.e., rate-based) flow control[28], for which congestion control either does not exist, or assumes a very rudimentary form [232]. Furthermore, congestion control in NACK-based applications is typically labeled as being "difficult" due to the "open-loop" operation of its flow control, and the actual extent of "difficulty" remains neither documented nor measured.

At the same time, before emerging real-time streaming applications can gain wide-spread acceptance, we believe that they first must implement some form of scalable congestion control. Therefore, in this chapter, we undertake an analysis and performance study that sheds the light on both the exact difficulties found in "open-loop" congestion control and the extent of penalty incurred by a NACK-based protocol in an Internet-like environment. In the course of our investigation, we found that traditional NACK-based congestion control possessed poor scalability (i.e., their use resulted in high packet loss when the number of simultaneous flows was large) and that the stability of existing NACK-based schemes was much lower than that of similar ACK-based schemes. Note that this chapter does not study a fundamental question of whether NACK-based congestion control can achieve the same level of stability as its ACK-based counterparts, but rather investigates previously-undocumented drawbacks of NACK-based congestion con-

trol and attempts to improve the performance of the existing schemes in rate-based applications.

Studying new congestion control methods in this chapter, we sometimes drift away from TCP-friendly schemes. Hence, we must mention a few words about why find such practice acceptable. We argue that in the future Internet, it is quite possible that UDP traffic will *not* compete with TCP in the same router queues (e.g., DiffServ may be used to separate these types of traffic at the router level). This intuition is driven by the fact that real-time flows have substantially different delay requirements from those of TCP, and it may not be practical to mix the two types of traffic in the same queues. Furthermore, NACK-based applications are unlikely to be fully TCP-friendly, because they often do not follow TCP's *fast retransmit* and *timeout backoff* algorithms and do not rely on the "packet-conservation" principle [110] in their flow control.

The remainder of the chapter is organized as follows. Section 5.2 provides the necessary background on increase-decrease (I-D) congestion control. In section 5.3, we define the notion of *monotonic convergence* to fairness of general I-D congestion control and derive certain desired properties of control functions that guarantee such monotonic convergence. We next focus on binomial algorithms [13] in section 5.4 and, under simple assumptions, derive their average link utilization and packet loss rate in the stable state. In section 5.5, we study packet-loss scalability of binomial congestion control and show that AIMD possesses the best scalability among all TCP-friendly schemes. In addition,

---

[28] Note that ACK-based flow control could be used in real-time streaming, but it typically results in some

we show that to achieve optimal scalability (i.e., constant packet loss), a congestion control scheme must have the knowledge of the bottleneck bandwidth. In section 5.6, we investigate the feasibility of using real-time bottleneck bandwidth estimates as a supplement to binomial congestion control and study whether the new schemes can achieve better scalability than AIMD in a real network.

5.2 Background

Within the class of end-to-end congestion control protocols, we specifically focus on the class of *increase-decrease* (I-D) methods. I-D congestion control implements a simple reactive control system, which responds to congestion by decreasing the sending rate and responds to the absence of congestion by increasing the sending rate. Hence, at any stage, the decision of I-D congestion control is binary.

Furthermore, the increase and decrease functions are *local* [13], [49], which means that they only use the local state of a flow in computing the next value of the sending rate. In addition, I-D congestion control usually assumes a *memoryless* model [13], [49], in which the amount of increase and decrease is based only on the value of the current sending rate rather than on the history of the sending rate (e.g., several flavors of "AIMD with history" are examined in [140], [141]). In this chapter, we explicitly assume a local and memoryless model of I-D congestion control.

---

form of QoS penalty (such as longer startup delays, more frequent buffer underflow events, etc.).

To prevent high-frequency oscillations on timescales smaller than it is needed to receive the feedback from the network, I-D congestion control is executed on discrete timescales of R time units long. Typically, R is a multiple of the round-trip delay (RTT) and in many cases, simply equals the RTT.

Many chapters study congestion control in the context of *window-based* flow control [13], [81], [141], [286] and apply I-D formulas to the size of congestion window *cwnd*. In such notation, assuming that the size of congestion window *cwnd* during interval i for a particular flow is given by w_i, I-D congestion control can be summarized as:

$$w_{i+1} = \begin{cases} w_i + W_I(w_i), & f = 0 \\ w_i - W_D(w_i), & f > 0 \end{cases}. \tag{28}$$

where f is the congestion feedback (positive values indicate congestion), and W_I and W_D are the increase and decrease functions of *window-based* I-D congestion control, respectively. In practice, feedback f is usually equal to the packet loss rate observed by the flow during the last interval (i.e., interval i).

Since our work focuses on *rate-based* streaming applications (in which *cwnd* has little meaning), we must write an equivalent formulation of increase-decrease congestion control using the value of each flow's sending rate r_i instead of congestion window w_i. The conversion from the packet-based notation to the rate-based notation is straightforward, i.e., each unit of w_i is equivalent to a rate of MTU/RTT bits/s, where the MTU (Maximum Transmission Unit) is given in bits and the RTT is given in seconds. In other words, $r_i = MTU/RTTw_i$.

Therefore, assuming that r_i is the sending rate of a particular flow during discrete interval i, the I-D congestion control (28) for that flow can be re-written as:

$$r_{i+1} = \begin{cases} r_i + R_I(r_i), \ f = 0 \\ r_i - R_D(r_i), \ f > 0 \end{cases}, \tag{29}$$

where R_I and R_D are the increase and decrease functions of *rate-based* I-D congestion control, respectively.

One special case of I-D congestion control is given by *binomial algorithms*, where the increase and decrease functions are simple power functions [13]:

$$\begin{cases} W_I(w) = \alpha w^{-k} \\ W_D(w) = \beta w^{l} \end{cases} \text{ or } \begin{cases} R_I(r) = \lambda r^{-k} \\ R_D(r) = \sigma r^{l} \end{cases}, \tag{30}$$

where all constants α, β, λ, σ are positive. For binomial algorithms, the difference between the two notations lies only in the constants in front of the corresponding power functions. Hence, the conversion from the window-based to the rate-based notation is supplied by the following formulas:

$$\lambda = \alpha\left(\frac{MTU}{RTT}\right)^{k+1} \text{ and } \sigma = \beta\left(\frac{MTU}{RTT}\right)^{1-l}. \tag{31}$$

Throughout the rest of the chapter, we will use both versions of binomial algorithms in (30), sometimes referring to constants (λ,σ) instead of constants (α,β), while keeping in mind the conversion in (31).

A special case of binomial congestion control that is implemented in TCP is called AIMD (Additive Increase, Multiplicative Decrease) [49], [110]. In AIMD, k equals 0, i.e., $W_I(w) = \alpha$ ($\alpha > 0$), and l equals 1, i.e., $W_D(w) = \beta w$ ($0 < \beta < 1$). AIMD(α,β) is *TCP long-term fair* [29], if it achieves the same average throughput when competing with a TCP connection under the same end-to-end conditions. The necessary condition for such long-term fairness is [81], [141], [286]: [30]

$$\alpha = \frac{3\beta}{2-\beta}. \tag{32}$$

On the other hand, for binomial congestion control (30) to be TCP-friendly, Bansal *et al*. [13] show that $k + l$ must be equal to 1. Among such (non-AIMD) TCP-friendly binomial congestion control, they propose two methods called IIAD (Inverse Increase, Additive Decrease) with $k = 1$, $l = 0$, and SQRT (Square Root) with $k = l = ½$.

Finally, we should mention that the analysis of increase-decrease congestion control typically assumes an ideal network with *synchronized* and *immediate* feedback [13], [49], [125], [139], [141]. *Synchronized* feedback means that all flows sharing a congested link receive notifications about packet loss at the same time. *Immediate* feedback means that if the capacity of any link along an end-to-end path is exceeded during interval i, feedback f is positive for interval i. Under these ideal conditions, Chiu and Jain [49] show

---

[29] Sometimes called *TCP-compatible* [13], [81] or *TCP-friendly* [286].

[30] Note that some papers [13], [285], [286] use a different notation, in which $W_D(w) = (1–\beta)w$ and this formula has a different form. Furthermore, if the rate of AIMD is dominated by timeouts, the formula assumes yet another form [286].

that all AIMD schemes converge to a fair state. In addition, Bansal *et al*. [13] show that for binomial algorithms (30) to converge to fairness, $k + l$ must be strictly greater than zero.

5.3 General I-D Control

Not all increase-decrease functions R_I and R_D guarantee convergence to *fairness*. In the context of I-D congestion control, convergence to fairness is usually defined as the ability of any number of identical flows sharing a common bottleneck link to reach a state in which their rates become equal and stay equal infinitely long. Even though in practice this is a very difficult goal to achieve, under the ideal conditions of synchronized and immediate feedback, many schemes can guarantee convergence to fairness.

One of the interesting properties of I-D congestion control that we introduce in this chapter is the ability of a scheme to approach fairness *monotonically*, i.e., if the fairness during interval i is given by f_i, $0 \le f_i \le 1$, then the following conditions are necessary for monotonic convergence:

$$\forall i : f_{i+1} \ge f_i \text{ and } \lim_{i \to \infty} f_i = 1 . \tag{33}$$

Generally, monotonic convergence is not necessary, but it is beneficial, because non-monotonic convergence tends to temporarily drive the system into extremely unfair states (i.e., one flow receiving much higher bandwidth), especially in the presence of random packet losses and heterogeneous feedback delays. Later in this chapter, we will relax

relax the above condition of monotonic convergence, but will keep the rest of the results in this section as they are applicable to both binomial algorithms and the ideally-scalable schemes studied in section 5.5.

It is common [13], [141] to examine the case of two flows sharing a link, since the extension to n flows can be easily performed by considering flows pair-wise. It is also common to use a continuous fluid approximation model [13] and disregard the discrete nature of packets (i.e., all packets are infinitely divisible). Furthermore, in this chapter, we use a max-min fairness function f_i instead of Chiu's fairness index [49]. Recall that max-min fairness of n flows with *non-zero* sending rates $(x_1,\ldots, x_n)$ is given by:

$$f = \min_{i \neq j} \left(\frac{x_i}{x_j} \right). \tag{34}$$

Consider two flows X and Y sharing a bottleneck link under the above assumptions. Suppose that during interval i, the flows' sending rates are given by x_i and y_i, respectively. To help us understand the behavior of a two-flow I-D control system, we use Figure 36 from [49]. In the figure, the axes represent the sending rate of each of the two flows. Furthermore, line $y = x$ is known as the *fairness line* and represents points (x, y) in which fairness f equals 1. Assuming that the capacity of the bottleneck link is C, line $x + y = C$ is called the *efficiency line* and represents points in which the bottleneck link is about to overflow. Given a particular point $P_i = (x_i, y_i)$ in the figure, line $y = mx$ connecting P_i to the origin is called the *equi-fairness line* (i.e., points along the line have the

same fairness $f_i = x_i/y_i = 1/m$). Furthermore, we define *efficiency* e_i of point P_i as the combined rate of both flows in that point, i.e., $e_i = x_i + y_i$.

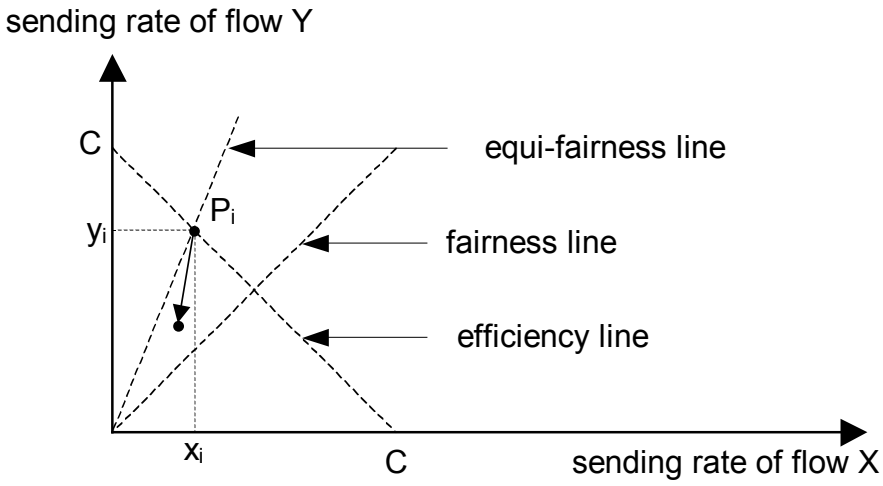

Figure 36. Two-flow I-D control system.

5.3.1 Decrease Function

To ensure monotonic convergence and proper response to congestion signals, the following four conditions must hold during each *decrease* step assuming that the system is in some point P_i just before the decrease step.

First, the efficiency in the new state must be strictly less than that in the old state, i.e., $e_{i+1} < e_i$. This condition ensures that flows backoff during congestion. Second, the fairness must not decrease in the new state, i.e., $f_{i+1} \geq f_i$. This condition guarantees monotonic convergence to fairness, and as pointed out before, although desired, it is often not available in practice. Consequently, we will relax this condition later in the chapter. Third, to properly maintain convergence, the system must not arbitrarily cross or oscillate

around the fairness line, i.e., it must stay on the same side of the fairness line at all times. For the case in Figure 36, we can write: $(y_i > x_i) \Rightarrow (y_{i+1} > x_{i+1})$. Finally, the system must not allow rates below or equal to zero, i.e., given an arbitrary state with $x_i > 0$ and $y_i > 0$, we must guarantee that $y_{i+1} > 0$ and $x_{i+1} > 0$.

The first condition is equivalent to:

$$x_{i+1} - x_i + y_{i+1} - y_i = -R_D(x_i) - R_D(y_i) < 0, \tag{35}$$

which can be satisfied with any positive function $R_D(x) > 0$, $\forall x > 0$. The second condition is equivalent to:

$$\frac{x_{i+1}}{y_{i+1}} \geq \frac{x_i}{y_i}, x_i > 0, y_i > 0. \tag{36}$$

Expanding the last inequality using (1) and generalizing by dropping the indexes (the inequality depends only on x_i and y_i), we get:

$$xR_D(y) - yR_D(x) \geq 0, \text{ for all } x > 0, y > 0, x < y. \tag{37}$$

Writing $y = x + \Delta x$, for $\Delta x > 0$:

$$x\big(R_D(x + \Delta x) - R_D(x)\big) - \Delta x R_D(x) \geq 0, \tag{38}$$

$$\frac{R_D(x + \Delta x) - R_D(x)}{\Delta x} \geq \frac{R_D(x)}{x}, \text{ for } x > 0, \Delta x > 0. \tag{39}$$

Restricting $R_D(x)$ to be a differentiable function for all $x > 0$, (39) is equivalent to:

$$R'_D(x) \geq \frac{R_D(x)}{x}, \text{ for all } x > 0. \tag{40}$$

Bringing $R_D(x)$ to the left and taking the integral (both x and $R_D(x)$ are known to be positive):

$$\int \frac{dR_D(x)}{R_D(x)} \geq \int \frac{dx}{x}, \text{ for all } x > 0. \tag{41}$$

$$\ln R_D(x) \geq \ln x + m_1, \text{ for all } x > 0. \tag{42}$$

$$R_D(x) \geq m_2 x, \text{ for all } x > 0. \tag{43}$$

The result in (43) shows that the original condition (40) restricts $R_D(x)$ to grow no slower than some linear function m_2x.

Using similar derivations, we find that the third condition (i.e., the non-cross-over condition) results in:

$$R'_D(x) < 1, \text{ for all } x > 0, \tag{44}$$

which means that $R_D(x)$ must grow slower than function x (i.e., the slope of $R_D(x)$ in all points $x > 0$ must be less than 1). Finally, the fourth condition

$$x - R_D(x) > 0, \text{ for all } x > 0 \tag{45}$$

is automatically satisfied by combining (40) and (44) above.

To summarize by combining (43) and (44), function $R_D(x)$ must be positive and differentiable for all values of $x > 0$, and must be an asymptotically (i.e., for substantially large x) linear function of x, with the slope strictly less than 1. For example, AIMD function $R_D(x) = \sigma x$ clearly satisfies these conditions for $0 < \sigma < 1$.

5.3.2 Increase Function

The analysis of increase function $R_I(x)$ is similar to the above. This time, instead of four conditions, we have only three. First, the efficiency in the new state must increase (i.e., $e_{i+1} > e_i$), which guarantees that flows will probe for new bandwidth in the absence of congestion. Second, the fairness must not decrease (i.e., $f_{i+1} \geq f_i$), which is the result of the same monotonicity requirement as before. And third, the system must not cross the fairness line (i.e., $y_{i+1} > x_{i+1}$). Crossing the fairness line violates monotonic converge to fairness and, as we will see later, never happens in practice (i.e., among binomial schemes).

The first condition is satisfied with any positive function $R_I(x)$, i.e., $R_I(x) > 0$, $\forall x > 0$. The second condition is the opposite of (40) due to a different sign in (1):

$$R_I'(x) \leq \frac{R_I(x)}{x}, \text{ for all } x > 0. \tag{46}$$

Finally, the third condition is similar to (44), but assumes the following shape:

$$R_I'(x) > -1, \text{ for all } x > 0. \tag{47}$$

Using (46), we find that R_I must grow no faster than some linear function $m_3 x$ and using (47), R_I cannot decay quicker than $-x$. For example, AIMD increase function $R_I(x) = \lambda$ again satisfies all conditions of monotonic convergence for $\lambda > 0$. We will look at other examples in the next section while studying binomial congestion control methods [13].

5.3.3 Convergence

Note that the above conditions still do not guarantee convergence to fairness. In other words, the conditions guarantee that if the system converges, it will do so monotonically, but the fact of convergence has not been established yet. Hence, we impose a final restriction on R_D and R_I – either the decrease or the increase step must strictly improve fairness, i.e., one of (40), (46) must be a *strict* inequality. If (40) is made into a strict inequality, we can no longer satisfy the condition in (44). Consequently, (40) must remain in its present form, and (46) must become a strict inequality.

5.4 Properties of Binomial Algorithms

5.4.1 Overview

Consider binomial algorithms in (30). Clearly, both functions R_I and R_D are positive for $x > 0$ and therefore, satisfy the first condition. The second condition (i.e., monotonically non-decreasing fairness) results in the following restrictions on k and l from applying (40) and the strict form of (46):

$$\begin{cases} -\lambda k x^{-(k+1)} < \lambda x^{-k} / x \\ \sigma l x^{l-1} \geq \sigma x^{l} / x \end{cases} \Rightarrow \begin{cases} k > -1 \\ l \geq 1 \end{cases}. \tag{48}$$

The third (i.e., non-cross-over) condition derived from (44) and (47) restricts l even further, but does not impose any limit on k (assuming sufficiently large x):

$$\begin{cases} k\lambda < x^{k+1} \\ l\sigma < x^{1-l} \end{cases} \Rightarrow \begin{cases} k = anything \\ l \leq 1 \end{cases}. \tag{49}$$

Note that restriction on l in (49) is dictated by the fact that sending rate x of a flow is not limited *a-priori* and the selection of a *positive* constant σ such that it is less than expression x^{1-l}/l, for substantially large $x > 0$, is feasible only when power $1-l$ is strictly non-negative.[31] Later in this chapter, we will show how restriction $l \leq 1$ can be lifted and what kind of advantages such schemes bring to congestion control protocols.

Consequently, assuming that the upper limit on x is not known, for a binomial algorithm to possess monotonic convergence to fairness, both (48) and (49) must be satisfied. In practice, this means that l must be strictly 1. Knowing that for TCP-friendly binomial congestion control $k + l$ must be one [13], we arrive at the fact that AIMD is the *only* TCP-friendly binomial algorithm with monotonic convergence to fairness. Hence, for the rest of the chapter, we will study schemes with non-monotonic convergence to fairness, because we want to go beyond what AIMD has to offer.

---

[31] Note that we implicitly assume that x is limited from below by some constant x_{min}. In window-based congestion control, x_{min} is equivalent to one unit of *cwnd* (i.e., *MTU/RTT*), and in rate-based congestion

In the absence of monotonic convergence, [13] shows that the necessary condition for convergence is $k + l > 0$ (i.e., flows make due progress towards the fairness line not necessarily at *every* step, but between every two consecutive *decrease* steps). Hence, dropping the monotonicity requirement and combining (49) with the convergence rule $k + l > 0$, we notice that the necessary restrictions on k and l for convergence of *non-monotonic* binomial algorithms are: $k > -1$ and $l \leq 1$.

5.4.2 Efficiency

The average efficiency is an important property of a congestion control scheme, which reflects how well the scheme utilizes the bottleneck bandwidth in the stable state. Clearly, higher efficiency is more desirable (but not necessarily at the expense of other properties of the scheme, such as packet loss or convergence speed). Formulas derived in this section not only help us study the efficiency of binomial schemes, but also are a necessary background for our packet-loss scalability analysis in the next section.

We define the *average efficiency* of a scheme as the percentage of the bottleneck link utilized by the scheme over a long period of time once the scheme has reached its stable state. In the stable state, each flow's sending rate will oscillate between two points, which we call the *upper point* (U) and the *lower point* (L) as shown in Figure 37. When a single flow is present in the network, U equals the capacity of the bottleneck link C. When n flows compete over a shared link of capacity C, U equals C/n for each flow (be-

---

control, x_{min} is the minimum rate at which real-time material can be received (e.g., the rate of the base video layer).

cause the flows have reached fairness by this time). In both cases, $L = U - \sigma U^l$ according to (30). In addition, since the pattern in Figure 37 is repetitive, it is sufficient to determine the average throughput of a flow during a single oscillation (i.e., between points *A* and *B*) rather than over a longer period of time. Note that in the window-based notation of congestion control, the maximum capacity of the link is given by $W = C \cdot RTT/MTU$.

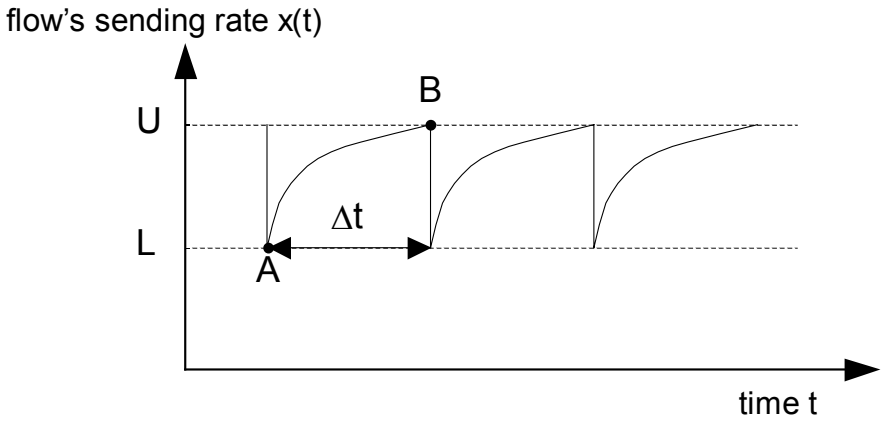

Figure 37. Oscillation of the sending rate in the stable state.

Using a continuous fluid approximation and results from [13], each flow's rate $x(t)$ *during the increase phase* (i.e., between points *A* and *B*) is given by:

$$x(t) = \left(\frac{\lambda(k+1)t}{R} \right)^{\frac{1}{k+1}}, \tag{50}$$

where *R* is a fixed duration of the control interval (which is typically equal to the value of the RTT). Following [13], the duration between points *A* and *B* in Figure 37 is:

$$\Delta t = \frac{U^{k+1}\left(1-\left(1-\sigma U^{l-1}\right)^{k+1}\right)R}{\lambda(k+1)}, \tag{51}$$

and the total amount of bits transmitted during the same interval is:

$$X = \frac{U^{k+2}\left(1-\left(1-\sigma U^{l-1}\right)^{k+2}\right)R}{\lambda(k+2)}. \tag{52}$$

Consequently, we derive that the flow's average sending rate during the interval is $X/\Delta t$ and the *average efficiency* (i.e., percent utilization) of a binomial congestion control scheme is:

$$e = \frac{X}{U\Delta t} = \frac{(k+1)\left(1-\left(1-\sigma U^{l-1}\right)^{k+2}\right)}{(k+2)\left(1-\left(1-\sigma U^{l-1}\right)^{k+1}\right)}. \tag{53}$$

Note that (53) can be converted to the window-based notation by replacing σ with β and rate U with its window equivalent. We also note that for large n, the exact model of efficiency e in (53) becomes inapplicable when $U = C/n$ drops below $\sigma^{1/(1-l)}$. We can no longer use any of the above derivations due to the fact that $1-\sigma(C/n)^{l-1}$ becomes negative, which is caused by the "drop-below-zero" effect (i.e., rate $x(t)$ becomes negative) that we tried to avoid before in (45). This condition was automatically satisfied given monotonic convergence to fairness in (40), but in the absence of monotonicity, we must explicitly restrict n to the following:

$$n < \frac{C}{\sigma^{1/(1-l)}} = \frac{W}{\beta^{1/(1-l)}}. \tag{54}$$

We next focus on simplifying the expression in (53). Equation (53) contains two terms of the form $1–(1–z)^q$, which can be expanded using Taylor series to:

$$1-(1-z)^q = qz\left(1-\frac{q-1}{2}z+\frac{(q-1)(q-2)}{6}z^2...\right). \quad (55)$$

Note that for $l < 1$, the value of z is less than 1, which means that the higher order terms in (55) get progressively smaller. Hence, by keeping the first two terms[32] in (55), we arrive at the following approximation to the exact formula in (53):

$$e = 1-\frac{\sigma U^{l-1}}{2-k\sigma U^{l-1}}. \quad (56)$$

To perform a self-check, we plug AIMD parameters ($l = 1$, $k = 0$) into (56) and get the familiar (and exact) formula of the average efficiency of an AIMD scheme: $e = (2–\beta)/2$ (recall that $\sigma = \beta$ in AIMD).

5.4.3 Packet Loss

The amount of packet loss during the stable state is another important property of a congestion control scheme. Consider one oscillation cycle between points *A* and *B* in Figure 37 and the case of a single flow. The maximum amount of overshoot under non-ideal (i.e., non-continuous) conditions will be the value of the increase function just before the flow reaches its upper boundary *C* in point *B*. Hence, the amount of the maximum overshoot for a single flow is given by $\lambda C^{-k}R$, where *R* is the fixed duration be-

tween control actions. Knowing how many bits X were sent by the flow during the same interval of duration Δt, we can write the average percentage of lost data p_1 using (52) and assuming the worst case of the maximum overshoot as:

$$\begin{aligned} p_1 &= \frac{\lambda C^{-k} R}{X + \lambda C^{-k} R} \\ &= \frac{\lambda^2 (k+2)}{C^{2k+2}\left(1-\left(1-\sigma C^{l-1}\right)^{k+2}\right)+\lambda^2 (k+2)} \\ &\approx \frac{\lambda^2 (k+2)}{C^{2k+2}\left(1-\left(1-\sigma C^{l-1}\right)^{k+2}\right)}, \end{aligned} \tag{57}$$

when $\lambda C^{-k} R << X$. In particular, for AIMD schemes, the packet loss rate in the worst case is given by:

$$p_1 = \frac{2\lambda^2}{C^2\sigma(1-\sigma)+2\lambda^2} \approx \frac{2\lambda^2}{C^2\sigma(1-\sigma)} = \frac{2\alpha^2}{W^2\beta(1-\beta)}. \tag{58}$$

A close look at the last equation reveals that as the number of flows increases (i.e., C is replaced by C/n), AIMD's packet loss rate will also increase. Furthermore, the amount of increase is proportional to n^2, where n is the number of flows. This confirms a well-known fact that AIMD scales as n^2 when it comes to packet loss [186]. Note that as $n \to \infty$, the amount of overshoot $\lambda C^{-k} R$ will become large compared to the value of X, and the approximations above will no longer work. However, the exact formulas in (57) and (58) will asymptotically approach the correct value of 100%.

---

[32] A one-term approximation used in [13] typically possesses an insufficient accuracy.

Consider a simple explanation of why AIMD scales quadratically. In AIMD, the increase in packet loss by a factor of n^2 comes from two places – from the reduction in the number of discrete increase steps N during interval Δt by a factor of n (because the increase distance $U–L$ becomes n times smaller), and from the reduction of duration Δt by the same factor of n (due to the same reason). As a result, the number of bits sent during the interval (which is proportional to $N\Delta t$) is reduced by a factor of n^2, and the amount of overshoot is unchanged (i.e., λR). Consequently, the total amount of lost packets relative to the number of sent packets is increased by a factor of n^2.

There are two reasons why we do not see this kind of performance degradation in practice. First, our results in (58) are based on a continuous fluid model, which assumes that packets are infinitely divisible. However, in practice, this approximation is true only when the amount of increase λR is negligible compared to the difference between the upper and lower limits, i.e., $U–L$ in Figure 37. Hence, when the number of *discrete* increase steps N becomes equal to 1 (or approaches 1), it can no longer be reduced by a factor of n, because it must remain an integer. Taking into account a fixed value of $N = 1$, the increase in packet loss becomes a linear rather than a quadratic function of n.

Second, most protocols employing AIMD rely on positive ACKs in implementing congestion control. This "self-clocking" [110], or "packet conservation," is capable of significantly improving the scalability aspects of AIMD, because the sender does not inject more packets into the network than the network can handle at any given time. "Open-loop" congestion control (i.e., NACK-based flow control) does not have this nice cushion to fall back on, and NACK-based AIMD schemes suffer a higher packet loss increase

than equivalent ACK-based schemes. In the next section, we will look at the scalability of general binomial algorithms and study how we can reduce the amount of packet loss as the number of flows increases.

5.5 Packet-loss Scalability of Congestion Control

5.5.1 Overview

Suppose the average packet loss when n flows share a link of capacity C is given by p_n. Let *packet loss increase factor* s_n be the ratio of p_n to p_1. Parameter s_n specifies how fast packet loss increases when more flows share a common link and directly relates to the ability of the scheme to support a large number of flows (i.e., schemes with lower s_n scale better). Using (57), we derive:

$$p_n \approx \frac{\lambda^2 (k+2) n^{2k+2}}{C^{2k+2}\left(1-\left(1-\sigma(C/n)^{l-1}\right)^{k+2}\right)}, \tag{59}$$

and using a two-term approximation from (55):

$$s_n \approx \frac{n^{l+2k+1}\left(2-(k+1)\sigma C^{l-1}\right)}{2-(k+1)\sigma(C/n)^{l-1}} = O(n^{l+2k+1}). \tag{60}$$

Hence, packet loss increase factor s_n of binomial algorithms is proportional to n^{l+2k+1} for small n and grows no faster than n^{l+2k+1} for the rest of n. For AIMD, we get the familiar scalability formula of n^2, whereas the IIAD (i.e., $k = 1$, $l = 0$) and SQRT (i.e., $k = l = ½$) algorithms scale as n^3 and $n^{2.5}$, respectively. Furthermore, among all *TCP-friendly*

schemes (i.e., $k + l = 1$), packet loss increase s_n is proportional to n^{3-l}, which means that TCP-friendly schemes with the *largest l* scale best. Since we already established that l must be no more than 1 (the non-cross-over condition), we arrive at our first major conclusion – *among TCP-friendly binomial schemes, AIMD scales best*.

We should make several observations about the applicability of (60) in practice. First, we assumed in (57) that the overshoot will be as large as possible, i.e., $\lambda U^{-k} R$. However, in many cases the actual overshoot will be some random value distributed between zero and $\lambda U^{-k} R$. Second, recall our discussion of AIMD's scalability in the previous section. When the increase distance $U–L$ becomes small compared to the value of the increase step, AIMD starts scaling as a linear function rather than a quadratic function. Hence, (60) is accurate only when the increase steps are small compared to C/n. The results based on the above model can be further skewed, if $\lambda U^{-k} R$ becomes large compared to X, in which case we must use the exact formula in (57).

5.5.2 Simulation

To verify these theoretical results and show some examples, we present simulation results of AIMD(1,½) and IIAD(1,½) schemes over a T1 link (i.e., C = 1,544 kb/s). For AIMD, we set *MTU/RTT* at two constant values of 5,000 and 50,000 bps (the corresponding schemes will be called $AIMD_1$ and $AIMD_2$) to show how their scalability changes when λ becomes large compared to the upper boundary $U = C/n$. For IIAD we selected *MTU/RTT* = 10,000 bps to allow the scheme to maintain $p_n << 100\%$ (otherwise, IIAD loses its n^3 packet loss increase). We used a discrete event simulation, in which n

flows of the same type shared a common link. We used our prior assumption of immediate and synchronized feedback, as well as the assumption that the flows employed a NACK-based protocol.

Figure 38 shows the variation of parameter s_n (based on the *actual*, rather than the *maximum* overshoot) during the simulation as a function of n for the three flows. In $AIMD_1$, packet loss increase ratio s_{100} reaches a factor of 6,755, which is equivalent to scalability of $n^{1.91}$ (just below the predicted n^2). On the other hand, $AIMD_2$ maintains its quadratic packet-loss increase only until $n = 7$, at which time it switches to a linear increase. The $AIMD_2$ scheme reaches an increase factor of $s_{100} = 352$, which is equivalent to an overall scalability of $n^{1.27}$. It may seem at first that the larger increase step λ of the $AIMD_2$ scheme is better; however, due to larger λ, $AIMD_2$ is much more aggressive in searching for bandwidth and suffers a lot more packet loss than $AIMD_1$ for all values of n. Thus, for example, for $n = 100$, $AIMD_2$ loses 55% of all sent packets, while $AIMD_1$ loses only 10%.

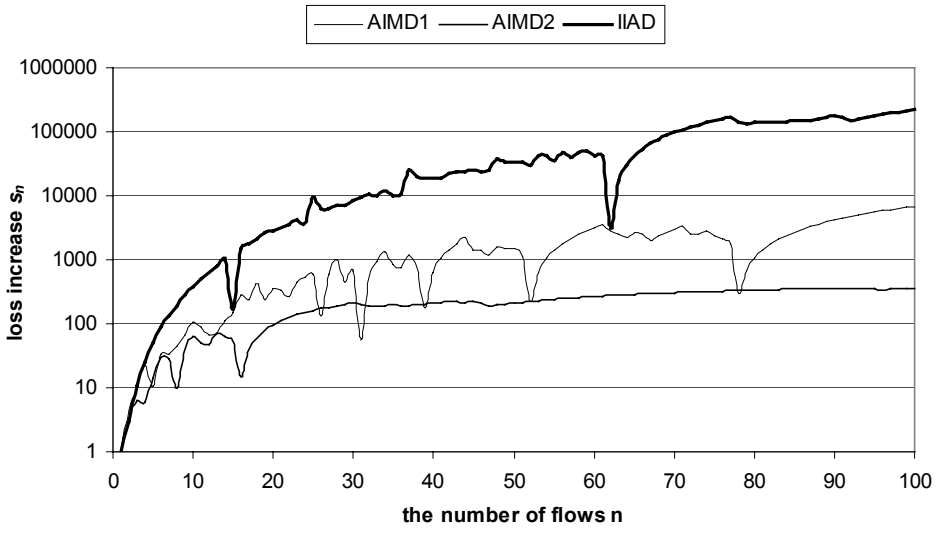

Figure 38. Parameter s_n (i.e., packet-loss scalability) of AIMD and IIAD in simulation based on actual packet loss.

Finally, IIAD's scalability performance is much worse than that of either of AIMD schemes as can be seen in the figure. The packet loss with 100 flows (i.e., p_{100}) is 219,889 times larger than the packet loss with one flow (i.e., p_1). Hence, under the given conditions, the overall scalability of IIAD is approximately $n^{2.67}$ (again slightly less than the predicted n^3).

As pointed out before and as shown in Figure 38, the *actual* increase in packet loss under non-ideal (i.e., discrete) conditions may be lower than that predicted by (60). Nevertheless, the theoretical result in (60) can be used as a good performance measure in comparing the scalability of different binomial schemes (e.g., as predicted, AIMD scales much better than IIAD).

5.5.3 Feasibility of Ideal Scalability

We next examine the "ideal scalability" of binomial schemes and derive its necessary conditions. We define a scheme to have *ideal scalability*, if s_n is constant for all n. This definition is driven by the fact that no matter how small packet loss p_1 can be made in a *non-scalable* scheme (i.e., a scheme with quickly-growing s_n), there will be a link of sufficient capacity that will accommodate such large number of concurrent flows n that p_n will be unacceptably high. This is especially true given the no-better-than-quadratic scalability of binomial congestion control. Consequently, the ideal situation would be to have a scheme that maintains a consistent packet loss rate regardless of the number of flows utilizing the scheme over a shared link, i.e., $p_n = p_1$ for all n. Furthermore, we would like to have a scheme that maintains the *same* packet loss over links of *different* capacity C.

To solve the above problem, we examine (60) again in order to find congestion control schemes that allow s_n to remain constant. Clearly, the necessary conditions for this ideal scalability are (the second condition is needed for convergence):

$$\begin{cases} l + 2k + 1 = 0 \\ k + l > 0 \end{cases} \Rightarrow \begin{cases} k < -1 \\ l > 1 \end{cases}. \tag{61}$$

The $l > 1$ condition means that if we plan to satisfy the non-cross-over conditions (44), (49), or prevent the scheme from reducing its rate below zero, ideal scalability requires the knowledge of some tight upper limit on sending rate x (see discussion following (49) earlier). Consequently, only assuming that x is limited by a constant (i.e., C), is it

possible to find such σ that will satisfy the necessary condition $\sigma < 1/(lx^{l-1})$ in (49) for all rates $0 < x \leq C$. Hence, we come to our second major conclusion – *among I-D congestion control schemes, ideal scalability is possible only when sending rates x are limited from above by a constant, i.e., when flows have the knowledge of the bottleneck capacity.*

There are two simple ways how an application can learn the value of C – by using real-time end-to-end measurements or by asking the network to provide an explicit feedback with the value of C. In the next section, we will examine the viability of applying the former method to sampling the capacity of the bottleneck link and the possibility of using such estimates in ideally-scalable congestion control.

Note that *all* flows sharing a single link must receive an estimate of C that is fairly close to the true capacity of the link[33]. A major drawback of employing congestion control that relies on real-time estimates of C is that different flows may form a different estimate, which may result in poor convergence and/or scalability depending on the amount of error. Hence, our approach in this section relaxes one condition (i.e., $l \leq 1$), but imposes a new one – all flows must measure the bottleneck capacity with high *consistency*. Note that a thorough evaluation of various bandwidth estimation methods for the purpose of ideally-scalable congestion control is beyond the scope of this chapter.

We also speculate that schemes with ideal scalability may be somewhat difficult to use in practice due to two factors – errors in measuring capacity C [66] and typically slower convergence to fairness due to less-aggressive probing for bandwidth. Neverthe-

---

[33] The more the error, the slower will the convergence be.

less, we investigated ideally-scalable congestion control until we established a working version of the algorithm, which we will present in the remainder of the chapter. Note that much more work in this area is required before we can recommend an I-D congestion control method other than AIMD for practical use over the Internet.

5.5.4 Ideally-Scalable Congestion Control

In this section, we introduce a new method, called *Ideally-Scalable Congestion Control* (ISCC), and show how values of the bottleneck capacity can be used to select the values of (λ,σ). Note that other ways of selecting (λ,σ) may be possible to achieve the same goal of constant s_n. We use notation ISCC(x) to refer to the ideally-scalable scheme described in this section with parameter l equal to x and parameter k equal to $-(l+1)/2$.

Assuming that C is known and assuming that $x(t) \le C$ at all times t (i.e., each application will limit its sending rate to be no higher than C), we can satisfy $\sigma < 1/(lx^{l-1})$ in (49) by choosing the following σ:

$$\sigma = \frac{1}{m_D C^{l-1}}, \tag{62}$$

where $l > 1$ and m_D is some constant greater than or equal to l. It is easy to show that the decrease step of schemes with σ according to (62) is no more than x/m_D for any given state $x > 0$. Hence, rate x is guaranteed to stay positive at all times. By varying constant m_D, the scheme can adjust its average efficiency, where larger values of m_D mean higher efficiency.

In addition, we must carefully select the value of λ so that the negative value of power k is not allowed to cause uncontrollably-high increase steps. One way to attempt to achieve this is to select a fixed value α and then multiply it by $(MTU/RTT)^{k+1}$ as shown in (31). However, the increase steps will still remain virtually unlimited, because the value of *MTU/RTT* has little relationship to the value of C (which is needed to effectively limit λx^{-k}). In addition, different flows may use different multiplicative factors in (31) due to the differences in the RTT or the MTU. An alternative approach is to apply a similar thinking to that used before in selecting σ – choose λ so that the increase step is always no more than x/m_I for any given rate x, where m_I is some constant greater than or equal to one. This can be written as:

$$\lambda x^{-k} \le x / m_I, m_I \ge 1, \tag{63}$$

which is satisfied with the following choice of λ:

$$\lambda = \frac{C^{k+1}}{m_I}, m_I \ge 1. \tag{64}$$

Parameter m_I can be used to vary the aggressiveness of the scheme in searching for new bandwidth, where larger values of m_I result in less aggressive behavior of the scheme.

Furthermore, the above selection of λ and σ allows us to separate the value of packet loss p_1 from the capacity of the bottleneck link C. Combining (57), (62) and (64), we get that packet loss of ISCC schemes is *link-independent*:

$$p_1 = \frac{k+2}{m_I^2\left(1-\left(1-1/m_D\right)^{k+2}\right)+k+2} \tag{65}$$

In the next section, we compare the performance of one particular ISCC congestion control scheme in a NACK-based real-time streaming application with that of IIAD, AIMD, and TFRC (TCP-Friendly Rate Control) [86].

5.6 Experiments

5.6.1 Choice of Powers Functions

We start with an observation that if *l* becomes much larger than 1.0 in an ISCC scheme and sending rate *x* is much smaller than capacity *C* (e.g., when *n* is large), such congestion control becomes less responsive to packet loss. Being less responsive usually results in very small rate reductions that often cannot elevate congestion in a single step. Thus, schemes with large *l* usually need multiple back-to-back decrease steps to move the system below the efficiency line in Figure 36. Our assumptions above do not model this behavior and the actual resulting packet loss in these schemes turns out to be higher than predicted by (60) and the convergence time is sometimes substantially increased.

Hence, from this perspective, larger values of *l* are not desirable. The only value of *l* that guarantees ideal scalability among *TCP-friendly* schemes (i.e., $k + l = 1$ and $l + 2k + 1 = 0$) is quite high and, specifically, equals 3. In practice, this scheme converges

very slowly[34] and may not be a feasible solution for the real Internet. Among non-TCP-friendly schemes, values of l close to 1.0 force k to come close to –1.0 (because l+2k+1 must still remain zero), which also results in slower convergence to fairness as sum $k + l$ approaches zero[35].

Among an infinite number of ISCC schemes, we arbitrarily selected a scheme with $l = 2$ ($k = –1.5$), which achieves reasonable performance in terms of both packet loss and convergence, and show its performance in this chapter. Note that this particular scheme is somewhat less aggressive that TCP and typically would yield bandwidth to TCP, if employed over a shared path (however, this effect becomes noticeable only when the number of flows n is large). Hence, the practical application of this ISCC scheme in the Internet would require the use of new QoS methods in routers (i.e., DiffServ) as discussed in the introduction. Alternatively, it may be possible to use other ISCC schemes (with a different l), which are not penalized by TCP and which do not suffer from much slower convergence. We consider finding the best ISCC scheme to be beyond the scope of this thesis.

5.6.2 Real-time Bandwidth Estimation

In this section, we briefly examine the accuracy of real-time bandwidth estimation in our NACK-based streaming application and in the next section, we show the perform-

---

[34] Slow convergence was found experimentally.

[35] Values of $k + l$ close to zero mean that the system makes very small steps toward the fairness line and thus, converges very slowly.

ance of ISCC(2), which relies on these real-time estimates for computing the values of λ and σ.

We used a Cisco network depicted in Figure 39 for all real-life experiments in this chapter. During the experiment, we disabled WRED and WFQ on all T1 interfaces to reflect the current setup of backbone routers. The server supplied real-time bandwidth-scalable MPEG-4 video, which included the FGS (Fine-Granular Scalable) enhancement layer [225], [226] and the regular base layer, to the client. Consequently, at any time t, the server was able to adapt its streaming rate to the rate $x(t)$ requested by the client, as long as $x(t)$ was no less than the rate of the base layer b_0 and no more than the combined rate of both layers.

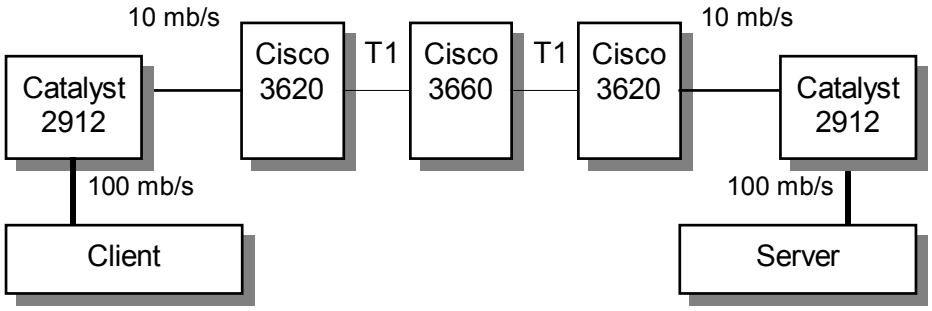

Figure 39. Setup of the experiment.

We used a 10-minute MPEG-4 video sequence with the base layer coded at b_0 = 14 kb/s and the enhancement layer coded up to the maximum rate of 1,190 kb/s. Note that two concurrent flows were needed to fully load the bottleneck link. Hence, our experiments below do not cover the case of $n = 1$, and s_n is defined as the ratio of p_n to p_2.

During the experiment, the client applied a simple packet-bunch estimation technique [44], [209] to server's video packets. To simplify the estimation of the bottleneck bandwidth, the server sent its packets in bursts of a pre-defined length. A bandwidth sample was derived from each burst that contained at least three packets.

To establish a baseline performance, Figure 40 (left) plots the PDFs of IP bandwidth estimates[36] obtained by two AIMD(1,½) flows over the T1 link in Figure 39 (both flows used a fixed value of *MTU/RTT* equal to 30 kb/s). As the figure shows, the flows measured the IP bottleneck bandwidth to be 1,510 kb/s, which is very close to the actual T1 rate of 1,544 kb/s (the discrepancy is easily explained by the data-link/physical layer overhead on the T1 line). Furthermore, both flows were in perfect agreement, and 99.5% of estimates of each flow were between 1,500 and 1,520 kb/s.

Figure 40 (right) shows the PDFs of bandwidth estimates obtained by 32 simultaneous AIMD(1,½) flows running over the same topology in Figure 39 and with the same value of *MTU/RTT*. This time, the majority of estimates lie in the proximity of 1,490 kb/s, and 95.5% of estimates are contained between 1,400 and 1,620 kb/s (i.e., within 7% of 1,510 kb/s). The lower accuracy of bandwidth estimation in the second case is explained by the lower average sending rate of each flow (i.e., 36 kb/s compared to 559 kb/s in the first case).

---

[36] Note that bandwidth *estimates* were derived from bandwidth *samples* by using the median of the past 20-seconds worth of samples.

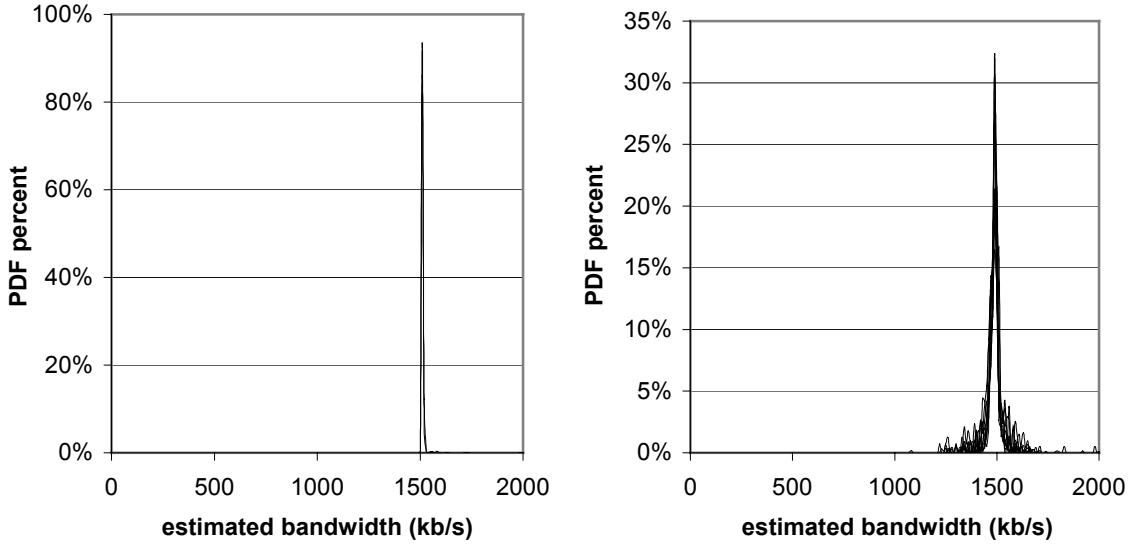

Figure 40. The PDFs of bandwidth estimates with 2 (left) and 32 (right) AIMD(1,½) flows over a shared T1 link.

Nevertheless, what matters most to the ISCC congestion control is the ability of flows to establish *consistent* estimates, rather than *accurate* estimates. To this extent, we found that the actual disagreement between the flows during the experiment was negligible and did not noticeably impact packet loss rates or fairness.

5.6.3 Scalability Results

We extensively tested ISCC in simulation and found that it performed very well, in fact achieving virtually constant packet loss. Below, we focus on more interesting results obtained over an experimental Cisco network. Recall that both the theoretical model and the simulation assumed *immediate* and *simultaneous* feedback from the network. These two conditions often do not hold in practice. Consequently, a real test of our results can only be obtained over a real network where receiver feedback can be arbitrarily

delayed and flows are not equally notified about packet loss (i.e., packets from different flows are dropped randomly without any fairness).

In our application with NACK-based congestion control, all methods used slow start at the beginning of each transfer; however, the results below exclude the behavior of the network during slow start and focus on the performance of the schemes in the interval starting 5 seconds after the *last* flow finished its slow start and ending when the *first* flow terminated.[37] This interval was 520 to 600 seconds long (depending on the number of flows) and included a combined transfer of approximately 60,000 packets.

During the experiment, we tried to select the parameters of the schemes so that the average packet loss of two competing flows using each scheme was between 0.3% and 0.6%. This constraint resulted in selecting *MTU/RTT* equal to 30 kb/s for AIMD(1,½), and 50 kb/s for IIAD(½,2). The value of the *MTU variable* in TFRC's equation [86] was selected to be 180 bytes, whereas the *actual* MTU used during the experiment was 1,500 bytes for all schemes. Note that TFRC was the only protocol, which used real-time measurements of the RTT in its computation of the rate.

The efficiency and aggressiveness parameters of the ISCC(2) scheme were set with the same goal in mind to maintain low initial packet loss p_2: $m_D = 2$ and $m_I = 20$. These parameters guarantee that each flow does not decrease its rate by more than ½ and does not probe for new bandwidth more aggressively that by 5% (i.e., 1/20) of the current sending rate.

---

[37] Flows were started with a 1.5-second delay.

The results of the experiment are summarized in Figure 41, which shows packet-loss increase factor s_n for four different schemes and values of n between 2 and 50. The results of the experiment show that all non-scalable schemes maintained a steady packet-loss increase to well over 15%. For example, IIAD reached $p_{50} = 45\%$ ($p_2 = 0.29\%$), AIMD 22% ($p_2 = 0.38\%$), and TFRC 20% ($p_2 = 0.26\%$).

On the other hand, the packet loss of the ISCC(2) scheme climbed only to 3.1% over the same range of flows n ($p_2 = 0.57\%$). A least-squares fit suggests that the increase in ISCC's packet loss is very slow, but noticeable (i.e., $n^{0.47}$). Thus, even though the ISCC scheme was not able to achieve constant packet loss in practice, it did show a substantially better performance than any other scheme.

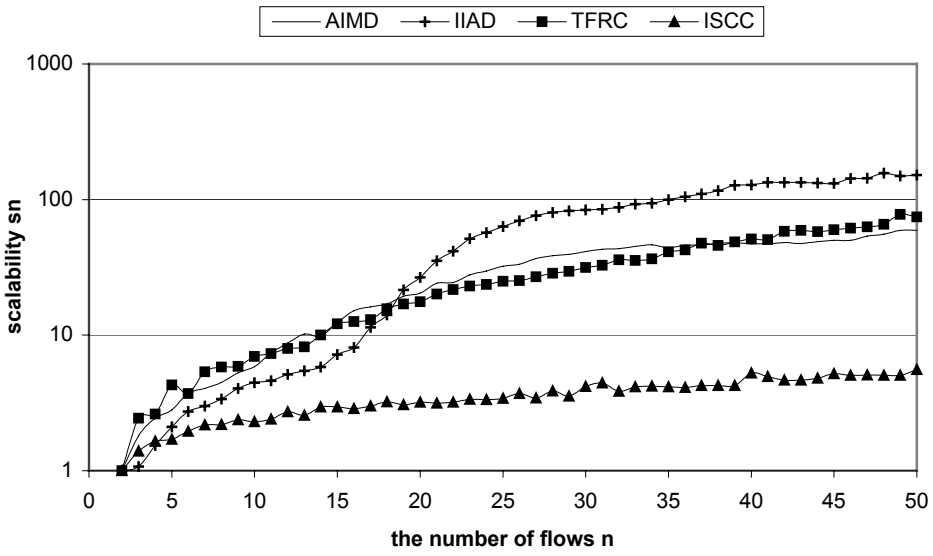

Figure 41. Packet-loss increase factor s_n for the Cisco experiment.

In addition, under the worst conditions (i.e., $n \approx 50$), our data show that the non-scalable protocols maintained a "frozen" picture between 11% and 42% of the corre-

sponding session due to *underflow events* (which are produced when a frame is missing from the decoder buffer at the time of its decoding). Clearly, these results indicate that high packet loss is very harmful, even in the presence of low RTTs (50-200 ms), large startup delays (3 seconds in our case), and an efficient packet loss recovery mechanism (our retransmission scheme were able to recover all base-layer packets before their deadlines until loss rates exceeded approximately 15%).

At the same time, the ISCC(2) scheme was able to recover *all* frames (including base and enhancement layer) before their decoding deadlines, representing an ideal streaming situation for an end-user.

Therefore, we come to the conclusion that non-scalable schemes are poorly suited for rate-based protocols that do not utilize self-clocking, and that ideally-scalable schemes promise to provide a constant packet-loss scalability not only in simulation, but also in practice. Nevertheless, further study is required in this area to understand the tradeoffs between the different values of l and k, as well as establish whether slower convergence to fairness found in simulation has any strong implications in large networks (i.e., in the real Internet).

Chapter Six

6 Real-time Estimation of the Bottleneck Bandwidth

This chapter examines the problem of real-time estimation of the capacity of a network path using end-to-end measurements applied to the application traffic (i.e., special bandwidth probes are not allowed). We start with two basic packet-pair bandwidth *sampling* methods and show how they can be applied in real-time to the application traffic. We then show how these two methods fail over multi-channel links and develop a new sampling method that remains robust over such links. We then examine the performance of the three sampling methods in a number of tests conducted using an MPEG-4 congestion-controlled streaming application over a Cisco network under a variety of conditions (including high link utilization scenarios). In addition, we study three *estimation* techniques, which can be applied in real-time to the collected samples, and show their performance in the same Cisco network with each of the sampling methods. We find that

two of the sampling methods combined with the best estimator maintain very accurate estimates for the majority of the corresponding session in a variety of scenarios and could in fact be used by the application for congestion control or other purposes.

6.1 Introduction

Several studies [2], [4], [44], [67], [68], [132], [133], [172], [209] focused on the problem of estimating the bottleneck bandwidth of an Internet path using end-to-end measurements. However, the majority of these studies targeted *off-line* estimation of the bottleneck bandwidth, where each estimation technique was applied to *all* collected bandwidth samples at the end of the actual experiment. Furthermore, even though a few studies (such as [270]) proposed the use of *real-time* estimates of the bottleneck bandwidth in end-to-end protocols, they did not report the performance of the proposed methods over real networks.

In addition, all of the above studies except [209] utilized special end-to-end probes (often of a fixed pre-defined size) and specific inter-packet transmission delays to sample the bandwidth. The use of additional probes typically required sending a reasonably large amount of probe traffic and involved a time-consuming sampling process before a reliable estimate could be established. Note that extensions of some of these methods to multicast applications can be found, e.g., in [231], [255].

In this chapter, we study the problem of estimating the bottleneck bandwidth in a unicast application both in *real-time* and using *only* intrinsic application traffic. New

emerging real-time streaming applications are a prime candidate for utilizing such real-time estimation techniques, which they can use in several ways. The first and most obvious use of a bottleneck bandwidth estimate is to limit the sending rate of the application during the *congestion avoidance* or *slow start* phases (i.e., while searching for new bandwidth) to be no higher than the estimate of the bottleneck bandwidth (thus resulting in less severe overshoot and lower packet loss[38] during these phases).

The second use, in applications like RealPlayer [232], is to avoid asking the user to set the speed of the bottleneck bandwidth in the preferences dialog. Real-time estimation of the bottleneck bandwidth (assuming it is sufficiently reliable) allows streaming applications to bypass this step and compute the initial streaming rate with more accuracy (especially, when the user has no idea about their bottleneck bandwidth). In addition, in file-sharing applications like Napster [187], the application will be more likely to report correct user connection bandwidth by not allowing the user to intentionally misrepresent his or her connection rate (which apparently happens quite often [94]).

Finally, several new increase-decrease congestion control methods [157] can take advantage of bottleneck bandwidth estimates (by adapting the constants of the control equations based on the bottleneck bandwidth) and can significantly increase the scalability of congestion control in *rate-based* protocols (which are the predominant type of protocols used in real-time streaming today [173], [232]).

---

[38] Note that even though the bottleneck bandwidth in the core Internet rarely equals the *available* bandwidth, many home users are limited by the speed of their access link. Therefore, in cases when a single application runs over the access link, the bottleneck bandwidth will often equal the available bandwidth.

In this chapter, we first define the problem of estimating the bottleneck bandwidth in *real-time* in section 6.2 and show how an existing method called Receiver-Based Packet Pair (RBPP) can be adapted to work in real time. In addition, we show how a simple extension of RBPP to several packets can be used in real-time to sample the bottleneck bandwidth and explain why it should perform better than RBPP. In section 6.3, we develop a new sampling method, which can be used over multi-channel bottleneck links (such as ISDN or several parallel T1 lines) and verify its performance in a large number of tests over the Internet. In section 6.4, we study the performance of all three real-time sampling methods using a congestion-controlled MPEG-4 streaming application over a Cisco testbed. In section 6.5, we examine three *estimation* techniques and compare their performance in the same datasets. We find that two of the methods perform equally well in a wide range of scenarios, while the third method possesses a much lower accuracy. We also find that estimates based on RBPP samples are often quite inaccurate, because they are affected by random scheduling delays introduced by the operating system (OS) of the client machine (i.e., in real networks, RBPP's performance is inadequate for fast bottleneck links).

6.2 Background

6.2.1 Packet Pair Concept

In this section, we formulate the problem of estimating capacity C of the bottleneck link of an end-to-end path using *real-time* end-to-end measurements. The estimation involves two steps – the collection of bandwidth samples and the actual estimation of C

based on the collected samples. In addition, note that we restrict the application to use only packets that are being sent by the sender as part of the application's normal operation (i.e., out-of-band probe traffic is not allowed).

The central principle used in all packet-pair bandwidth sampling methods [44], [67], [209] is the premise that if two (or more) packets leave the sender with spacing smaller than the transmission delay τ of the packets over the bottleneck link, this spacing will be expanded by the bottleneck link to a value, which can be used by the receiver to reconstruct the speed of the bottleneck link. This concept (assuming ideal conditions and a single-channel bottleneck link) is illustrated in Figure 42, which was modified from [110]. The sender injects two packets at high speed into the path, and the small spacing between the packets is expanded by the bottleneck link. Hence, assuming no interfering traffic before or after the bottleneck link, the receiver is able to compute the value of the bottleneck link by inverting spacing $\Delta T = \tau$ between the arriving packets. Since the receiver (rather than the sender) computes the bandwidth, the method is called *Receiver-Based Packet Pair* (RBPP) [209].

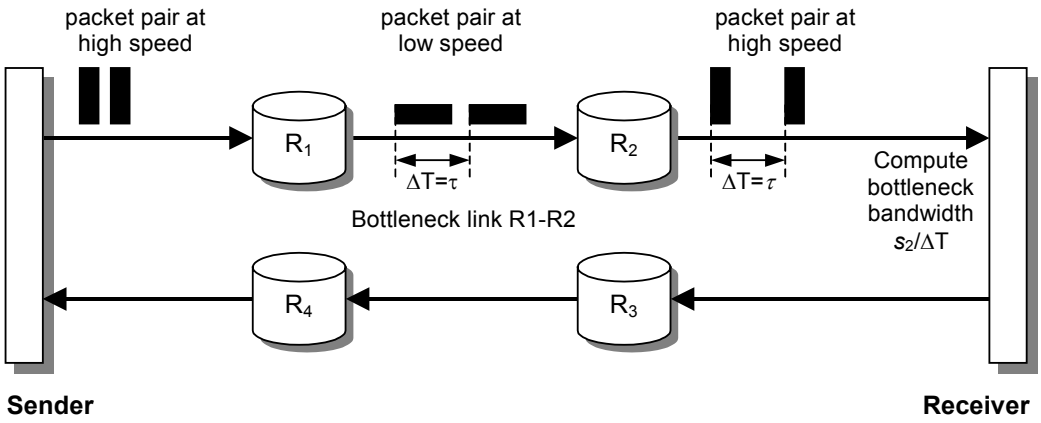

Figure 42. Receiver-Based Packet Pair.

In practice, however, the second packet may suffer large queuing delays before or after the bottleneck link due to interfering traffic, in which case, the packet pair may suffer an *expansion event* (i.e., spacing ΔT is larger than the transmission delay τ of the packets over the bottleneck link). In addition, if the first packet is delayed after it passes the bottleneck link and the second packet "catches up" with it, the packet pair may suffer a *compression event* (i.e., spacing ΔT is less than τ). Expansion events result in under-estimation, and compression events result in over-estimation of the actual bandwidth C. Therefore, in real-life situations, a great deal of samples will be inaccurate, and an estimation technique will be required to extract the most likely value of C from the collected samples (more on this in section 6.5).

6.2.2 Sampling

Assuming each packet can be of a different size (because the size of each packet is determined by the application), we use the following methodology for computing

bandwidth samples in RBPP. Suppose the distance between the two packets (of size s_1 and s_2, respectively) as they arrive to the application is ΔT (i.e., ΔT is the distance between the last bits of each packet, as shown in Figure 43) and suppose that ΔT is larger than the spacing between packets when they left the sender. Consequently (assuming ideal conditions, i.e., no expansion or compression events), distance ΔT is equal to the transmission delay of the second packet over the bottleneck link. Therefore, the receiver in RBPP computes its bandwidth samples for each pair of eligible packets as following:

$$b = \frac{s_2}{\Delta T}. \tag{66}$$

Note that it is the responsibility of the application to identify which packet pairs comply with the above requirement (i.e., transmission spacing is smaller than the receipt spacing). In protocols like TCP, it is necessary to use timestamps in each packet to make sure that packets were in fact expanded along the path to the receiver [209]. However, in rate-based streaming applications, it is possible to send packets in bursts (which is currently the most popular way of sending real-time traffic [173]) and compute bandwidth samples based on packets within each burst (i.e., knowing that each burst left the server at a higher rate that then bottleneck link[39]). Regardless of the actual implementation, we assume that the receiver can reliably detect back-to-back packets that were expanded by the network.

---

[39] If this is not the case, the server itself is the bottleneck.

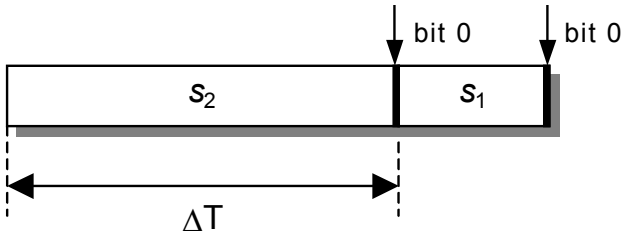

Figure 43. Computing bandwidth from inter-packet spacing.

Since many protocols send bursty traffic (including TCP), it often happens that several packets leave the sender back-to-back and are expanded by the end-to-end path. Hence, in such cases, a straightforward extension of the RBPP concept is to use the spacing between the first and the last packets of the burst in computing the bandwidth (e.g., similar methods were used in [67], [172], [209]). Suppose s_i is the size of packet i in the burst, n is the number of packets in the burst, and ΔT is the distance between packet p_1 and packet p_n. We call this scheme *Extended RBPP* (ERBPP) and compute bandwidth samples as follows:

$$b = \frac{1}{\Delta T} \sum_{i=2}^{n} s_i \, . \tag{67}$$

Note that the transmission delay of the first packet is not included in spacing ΔT, and therefore, the size of packet p_1 is not included in the sum in (67). The apparent benefit of using ERBPP over RBPP is the longer duration ΔT, which can be measured with higher accuracy in the presence of low-resolution clocks and random OS scheduling delays. However, as pointed out in [66], longer bursts of packets in ERBPP are more likely to be delayed by cross-traffic at the bottleneck router, thus resulting in under-estimation

of capacity C (i.e., due to burst-expansion events). We evaluate these tradeoffs in section 6.4.

Furthermore, instead of computing a single ERBPP sample for each burst, the application may decide to compute n–1 RBPP samples using the spacing between each two consecutive packets within the burst. As suggested in [66], RBPP samples may provide more accurate information about the true value of capacity C compared to the methods that use multiple back-to-back packets (i.e., ERBPP). Therefore, throughout this chapter, we will use notation RBPP to refer to n–1 samples computed using the size of the inner packets (i.e., packets $p_2, \ldots, p_n$) of each burst. Note that we apply both RBPP and ERBPP only to bursts that contain no lost packets, reordered packets, or retransmissions arriving in the middle of the burst.

We delay the performance study of RBPP and ERBPP until section 6.4. Next, we show how both of these methods fail over multi-channel links and develop a new bandwidth sampling technique that remains robust over N-channel ($N \geq 2$) links.

6.3 Multi-channel Links

6.3.1 Introduction

In this section, we analyze the problem of estimating the bottleneck bandwidth along paths where the bottleneck link consists of multiple parallel links, along which packets of a connection are load-balanced on the *data-link* layer[40] by the attached routers.

[40] Note that multi-path routing (i.e., load-balancing on the network layer) is not covered by our study.

Using our analysis, we develop a new sampling method called ERBPP+, which is immune to problems associated with multi-channel bottleneck links. We further evaluate our theoretical findings in tests conducted over ISDN links to two different ISPs and find that ERBPP+ performs very well.

We study the bandwidth estimation problem using the example of ISDN; however, the formulas derived in this section apply to any two-channel link. Furthermore, at the end of this section, we extend our results to N-channel ($N \geq 2$) links while creating a foundation for ERBPP+.

ISDN BRI (Basic-Rate Interface) is implemented as two parallel data links, where each link is called a *B-channel* and has the capacity of 64 Kbps. In some data-link protocols (such as multi-link PPP [254], [256]), the sending side has the option of splitting each packet into *fragments*, each of which is load-balanced over the two channels. Clearly, multi-link PPP (MLP) fragmentation involves extra overhead (i.e., splitting packets into fragments, additional MLP headers for each fragment, reassembly, etc.), and is often disabled in the Internet routers. Furthermore, some routers are not configured to use or do not support MLP, and instead, always load-balance whole packets by simply alternating packets between the available channels[41] (e.g., this was the situation studied in [209]).

Therefore, as we will see below, if the routers at the bottleneck link decide to load-balance entire packets (rather than small fragments), the accuracy of both RBPP and

---

[41] Such as Cisco-supported Bandwidth On Demand (BOD).

ERBPP sampling methods is severely reduced. Figure 44 shows a typical setup that we will study in this section. Load-balancing routers R_1 and R_2 are attached to each other with two links (each could be an ISDN B-channel, a T1 link, or any other point-to-point link) of capacity C_A each[42]. The second router is connected to the client through a path with the bottleneck capacity C_B, which we will consider, without loss of generality, to be equivalent to an abstract link of rate C_B directly connecting R_2 to the client.

We assume that the path from the server to R_1 is likely to consist of high-speed links, which means that packets in each burst arrive to R_1 almost instantaneously compared to their transmission time over the ISDN link (this assumption usually holds in practice since the ISDN link is the bottleneck in this scenario). We simplify the model even further and assume packets of equal size.

In Figure 44, four packets $p_1, \ldots, p_4$ simultaneously arrive to R_1 and are being transmitted across the ISDN link. Router R_2 receives each pair of packets and forwards them to the client at the maximum rate of the remaining path (i.e., C_B). Consequently, every *other* pair of packets (i.e., pairs (p_1, p_2), (p_3, p_4), etc.) arrives to the client at rate C_B.

---

[42] Note that many routers support load balancing over links of equal, as well as unequal capacity. In our simplified model, we use links of the same speed.

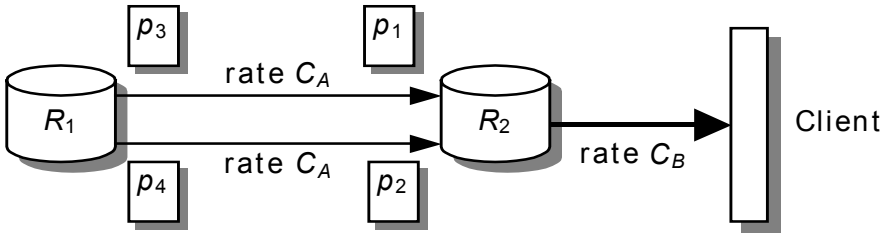

Figure 44. Model of a typical two-channel ISDN link.

Furthermore, as we will see below, the *remaining* pairs of packets similarly produce an incorrect measurement of the bottleneck capacity $2C_A$. The next section will establish this result more conclusively.

6.3.2 Receiver-Based Packet Pair

Using Figure 44, we already established that the RBPP method based on packets p_{2k-1} and p_{2k} ($k \geq 1$) from each burst, will measure rate C_B instead of $2C_A$. Our analysis above and for the rest of this section assumes that rate C_B is no less than $2C_A$ and that all packets are of the same size, which simply means that packets in each pair arrive to R_2 simultaneously and packets from different pairs do not queue behind each other at router R_2.

The issue left to resolve is that of what measurements are produced by the RBPP scheme based on packets p_{2k} and p_{2k+1} ($k \geq 1$). To simplify the explanation of our derivation, we assume that $k = 1$ and use the scenario in Figure 44. Hence, our goal is to determine the spacing between packets p_2 and p_3 when they arrive to the client. Let the time when packets p_1 and p_2 arrive to router R_2 be t_1. Consequently, packet p_1 leaves R_2 at the same time t_1, packet p_2 leaves R_2 after p_1 is fully transmitted, i.e., at time $t_2 = t_1 + s_1/C_B$,

and p_3 leaves R_2 at time $t_3 = t_1 + s_3/C_A$, where s_i is the size of packet i. The corresponding arriving times of the packets to the client are $t_1 + s_1/C_B$ for packet p_1, t_2+s_2/C_B for packet p_2, and t_3+s_3/C_B for packet p_3. Therefore, assuming packets of equal size s, the distance between packets p_2 and p_3 when they arrive to the client is $[s/C_A - s/C_B]$ and the corresponding bandwidth sample is:

$$b_{RBPP} = \frac{C_A C_B}{C_B - C_A}. \tag{68}$$

Hence, approximately half of RBPP samples collected over a path with a two-channel ISDN bottleneck should be equal to C_B and the other half are given by (68) (note that none of the samples will be equal to the bottleneck capacity $2C_A$). In practice, however, this ideal scenario may be violated, if smaller packets are queued behind larger packets at router R_1 (in which case, any arbitrary bandwidth between b_{RBPP} and C_B can be measured), or if larger packets are queued behind smaller packets (in which case, any bandwidth between $2/3 \cdot C_A$ and b_{RBPP} can be measured)

6.3.2.1 Experimental Verification

In our experiments, we found that certain routers on the ISP's side did not support MLP fragmentation, and thus, the situation depicted in Figure 44 was quite common. For example, our Cisco router was never able to connect to an ISP's router that fully supported MLP fragmentation. In the experiments analyzed below, the Cisco router used two different ISPs to connect to the Internet – a nationwide ISP (which we call ISP_a) and a local dialup ISP in the state of New York (which we call ISP_b).

In this setup, the router connected to a local client through a 10-mb/s Ethernet (i.e., C_B was equal to 10 mb/s) as shown in Figure 45 (left). Furthermore, the path from the server to the client consisted of 11 hops through ISP_a and 13 hops through ISP_b. In all experiments, we used a Windows2000 client and a Solaris2.7 server. The two video streams used in this experiment were coded using MPEG-4 at the IP rates of 59 kb/s (stream S_1) and 87 kb/s (stream S_2). We call this dataset D_1.

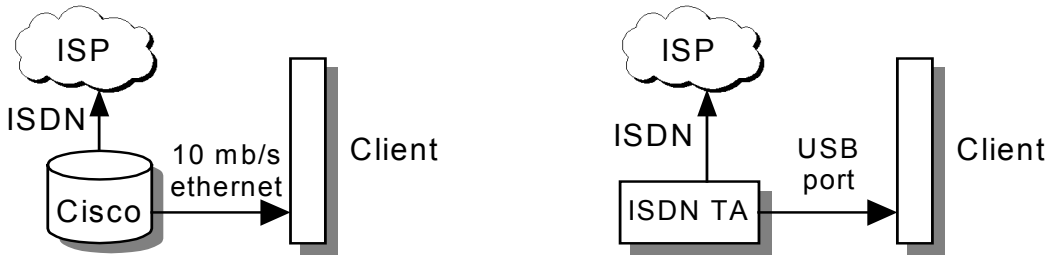

Figure 45. Setup of ISDN experiments with a Cisco router (left) and ISDN TA (right).

Another bandwidth measurement experiment over ISDN links employed a popular model of an ISDN Terminal Adapter (TA) used by Windows 95/NT/2000 users. The TA connected to the client machine through a 12-Mbps USB interface as shown in Figure 45 (right). The experiment with the ISDN TA used ISP_a and video stream S_2. This experiment transported over 73 million packets, whose dynamics were recorded in the second dataset D_2. We should further note that during the experiments documented in datasets D_1 and D_2, we forced the Cisco router and the TA to always bring both channels up at the beginning of each connection (i.e., the bottleneck bandwidth was always 128 Kbps).

In our first attempt to verify the behavior of RBPP over multi-channel links, we focus on set D_1. Our analysis above predicts that the first bandwidth peak in the distribution of RBPP samples should be at 10 mb/s (i.e., C_B) and the second peak should be at 64.4 kb/s (i.e., b_{RBPP} expressed in (68)). In reality, RBPP samples in set D_1 were clustered around two slightly different values – 9.7 mb/s and 62 kb/s; however, keeping in mind that PPP and Ethernet headers were not included in our RBPP estimates, it appears that 9.7 mb/s closely matched the rate of Ethernet excluding data-link headers and 62 kb/s closely matched 64.4 kb/s excluding PPP headers.

Our second verification attempt is based on set D_2. The USB link between the ISDN TA and the client was no slower than 128 Kbps and, in fact, was significantly faster. Nevertheless, we had doubts that the TA would be able to deliver packets at the full 12 Mbps supported by the USB port. From the documentation, it was clear that the TA supported rates above 230.4 kb/s (which is the speed of a fast asynchronous serial port), but we were unable to find the maximum "official" supported rate C_B. Hence, we had to use our data recorded in D_2 to first infer the value of C_B and then apply (68) to derive the value of b_{RBPP}. The dataset showed that almost all RBPP samples were clustered around two strong peaks – the first one at 70.9 kb/s and the second one at 461 kb/s. The peak at 461 Kbps closely matches twice the speed of a fast serial port (230.4*2), which tempts us to speculate that rate C_B supported by the TA was in fact equal to 460.8 kb/s. Using C_B = 460.8 kb/s in (68), we derive that the b_{RBPP} samples should have been equal to 74.3 kb/s. Again, subtracting PPP headers from 74.3 kb/s, it appears that an IP-level estimate of 70.9 kb/s closely matched that predicted by (68).

Therefore, both experiments verified our previous finding that none of RBPP samples is likely to produce correct measurements of the bottleneck capacity of a multi-channel link.

6.3.3 Extended Receiver-Based Packet Pair

The ISDN scenario that we study in the context of ERBPP is similar to the one shown in Figure 44. However, this time, we use more than two packets from each burst to compute the bandwidth. As before, suppose that $C_B \geq 2C_A$ (i.e., the first packet of each burst encounters an idle ISDN link, and packets from different pairs do not queue behind each other), all data packets are of the same size, and the client uses $m \geq 2$ packets from the beginning of each burst to produce bandwidth sample b_m using ERBPP, as if the burst contained exactly m packets.

Using Figure 44, we notice that for all odd values of $m \geq 2$ (i.e., $m = 2k+1$, $k \geq 1$), bandwidth samples b_m are equal to $2C_A$, which is the correct value of bottleneck capacity C. For example, let us consider the case of the smallest odd value of $m \geq 2$, i.e., $m = 3$. Using the calculations in the previous section and assuming packets of equal size, the inter-arrival distance between packets p_1 and p_3 at the client machine is maintained the same as at router R_2, i.e., s_3/C_A. Hence, the resulting ERBPP bandwidth sample is $(s_2+s_3)/(s_3/C_A) = 2C_A$. The formula for the rest of odd values of m is easily proven using math induction.

On the other hand, *none* of the even samples b_m produces the desired value of $2C_A$. For $m = 2$, we already know that $b_2 = C_B$. Consider a simple case of $m = 4$. Since we

established above that the distance between p_1 and p_3 when they arrive to the client is s_3/C_A, the distance between p_1 and p_4 is simply s_4/C_B time units larger, i.e., the total distance between p_1 and p_4 at the client is $[s_3/C_A+s_4/C_B]$. Hence, ERBPP sample b_4 is equal to $(s_2+s_3+s_4)/(s_3/C_A+s_4/C_B) = 3C_AC_B/(C_B+C_A)$. Using math induction for the rest of even values of m, we get the resulting formula: $b_m = (m–1)C_AC_B/[(m–2)/2\cdot C_B+C_A]$. Combining both cases together, we arrive at the final formula for b_m ($m \geq 2$):

$$b_m = \begin{cases} 2C_A, & m = 2k+1 \\ \dfrac{2(m-1)C_AC_B}{(m-2)C_B + 2C_A}, & m = 2k \end{cases} . \qquad (69)$$

Analyzing our result in (69), it is easy to notice that the even values of m will produce bandwidth samples that are always *higher* than the desired bandwidth $2C_A$. The amount of over-estimation, $b_m/(2C_A)$, is not limited by a fixed constant only when $m = 2$ (which argues against using RBPP), and is limited by 1.5 for all $m \geq 4$ and by 1.25 for all $m \geq 6$. Furthermore, function b_m for even m is monotonically decreasing and asymptotically converges to the desired estimate $2C_A$ as m tends to infinity (which argues in favor of using more packets per burst).

Using simple calculation, we further establish that for an even estimate b_m to converge within α percent of $2C_A$, m has to be at least $(2+1/\alpha)$ (i.e., convergence is asymptotically linear). Finally, we should note that for higher streaming rates, longer bursts of packets will allow ERBPP to operate over multi-channel bottlenecks with high accuracy

(i.e., both odd and even samples b_m will produce values close to $2C_A$). For example, 12-packet bursts generate samples within 10% of $2C_A$, and 102-packet bursts within 1%.

6.3.3.1 Experimental Verification

Assuming again that C_A = 64 kb/s and C_B = 460.8 kb/s, our goal is to compare samples b_m computed according to (69) with those derived from set D_2 (set D_1 produced similar results). Suppose each burst i recorded in the dataset contains n_i packets and suppose that the first e_i packets ($2 \le e_i \le n_i$) of the burst have the *same size* (which is a necessary condition given variable-size packets in our experiments). Then, using burst i from the dataset, we can compute samples b_m for $m = 2,\ldots, e_i$.

Table III compares the predicted values of b_m to those observed in dataset D_2. As the table shows, an overwhelming majority of samples b_m, for each value of m, were clustered around a single peak, and the peak closely matched the one predicted by (69) (as before, data-link/physical layer headers account for the slight difference).

Consequently, our experiments confirm that the accuracy of ERBPP over multi-channel links is reasonably low and also establish that the behavior of the majority of ERBPP samples can be described by a simple formula in (69).

m	Predicted value of b_m, kb/s	Observed peak in b_m samples, kb/s	Observed peak range, kb/s	Percent samples in the range
2	460.8	461	457-467	97.1%
3	128.0	122	119-127	99.5%
4	168.5	162	158-168	99.3%
5	128.0	123	121-126	99.7%
6	149.6	144	141-147	99.4%
7	128.0	123	121-126	99.9%
8	142.7	137	137-138	100%

Table III. Comparison of observed samples of b_m with those predicted by the model.

6.3.4 ERBPP+

We conclude this section by deriving a simple and low-overhead bandwidth estimation technique, which we call ERBPP+, from our model in (69) that remains robust along paths with *N*-channel bottlenecks. Using intuition, we can extend our previous results to conclude that along a path with *N* parallel channels of *equal* bandwidth C_A, samples b_m, $m = N \cdot k + 1$, $k \geq 1$, will measure the exact (and desired) bandwidth of the path (i.e., $N \cdot C_A$), and the remaining samples b_m will measure an *over-estimate* of the desired value. Therefore, if the client uses the first e_i equal-size packets from each burst i to compute samples b_m, $m = 3,\ldots, e_i$, and keeps the *smallest* sample from each burst, it will effectively filter out over-estimates and will identify the sample closest to the desired value of $N \cdot C_A$. Of course, this method assumes that each e_i is greater than or equal to $N + 1$ (for example, to handle up to 4 parallel channels of equal capacity, our method must analyze bursts with e_i at least 5 packets).

In Figure 46, we compare the performance of ERBPP (left) and ERBPP+ (right) in dataset D_2. Note that ERBPP's peaks at 162 and 137 kb/s (produced by samples b_4 and

b_8) are very small due to a small number of bursts with 4 and 8 packets during the experiment. However, the erroneous peak at 144 kb/s[43] (due to samples b_6) is quite noticeable. In addition, the performance of ERBPP gets even worse than predicted by (69), because smaller packets queued behind larger packets (and vice versa) at the ISDN link, in violation of our assumptions, are responsible for a large number of inaccurate samples in-between the two strong peaks (at 123 and 144 kb/s) and as low as 96 kb/s.

On the other hand, ERBPP+ achieves almost perfect performance with 99.9% of the samples located between 119 and 126 Kbps, and 56% of the samples between 122 and 123 Kbps. Even though ERBPP+ is much more reliable over multi-channel links than ERBPP, the penalty in accuracy of ERBPP+ over *single-channel* links remains unclear. Therefore, we will address this issue in the next section, where we compare the performance of all three sampling methods over a loaded T1 bottleneck in our experimental Cisco network.

[43] Note that this bandwidth peak is completely unrelated to the 144 kb/s (i.e., channels 2B+D) physical bandwidth of an ISDN link.

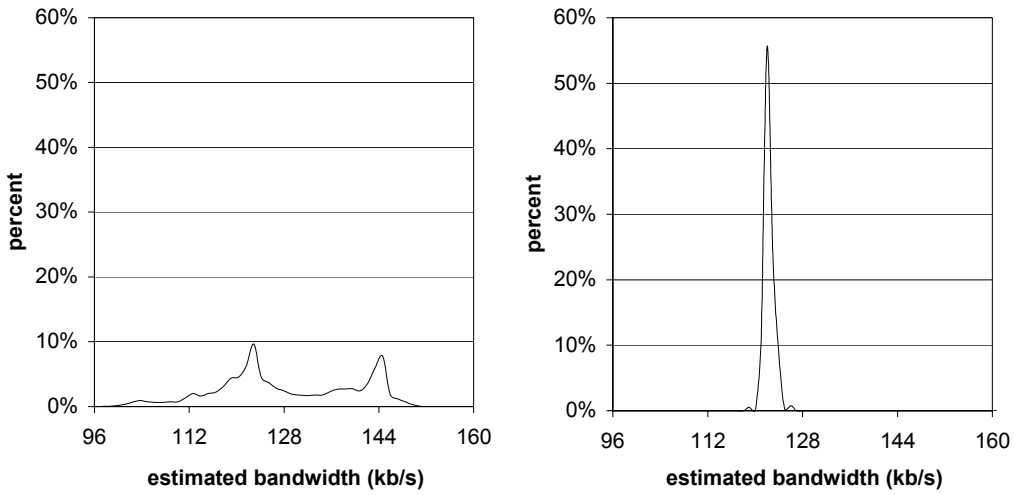

Figure 46. Comparison of ERBPP (left) with ERBPP+ (right) in D_2.

6.4 Sampling

6.4.1 Setup of the Experiment

We used a Cisco network depicted in Figure 47 for all experiments reported in the remainder of the chapter. The server and the client were connected to Catalyst switches via two 100 mb/s Ethernets. The switches in turn were connected to Cisco 3620 routers via 10 mb/s Ethernets. The 3620 routers connected to each other via T1 links passing through an additional Cisco 3660 router. During the experiment, we disabled both WRED and WFQ [61] on all T1 interfaces to reflect the current setup of backbone routers.

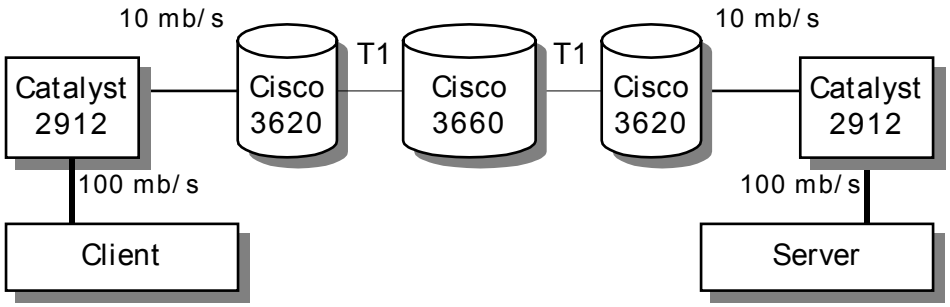

Figure 47. Setup of the experiment.

The server supplied real-time bandwidth-scalable MPEG-4 video, which included the FGS (Fine-Granular Scalable) enhancement layer [226] and the regular base layer, to the client. Consequently, at any time t, the server was able to adapt its streaming rate to the rate $r(t)$ requested by the client, as long as $r(t)$ was no less than the rate of the base layer b_0 and no more than the combined rate of both layers. In the experiments reported in this chapter, we used a simple AIMD (Additive Increase, Multiplicative Decrease) congestion control with parameters α and β set at 1 and ½, respectively (i.e., the parameters used in TCP – the search for new bandwidth during congestion avoidance was by 1 packet per RTT and the reduction of the rate upon packet loss was by ½). Furthermore, all clients used TCP-like *slow start* at the beginning of each session.

We used a 10-minute MPEG-4 video sequence with the base layer coded at b_0 = 14 kb/s and the enhancement layer coded up to the maximum rate of 1,190 kb/s. Note that two concurrent flows were needed to fully load the bottleneck link. In this chapter, we focus on two different cases – low and high link utilization. In the former case, we used two simultaneous AIMD flows running between the server and the client machines

(the average link utilization was 71% and the average packet loss was 0.3%). We call this dataset Φ_2. In the latter case, we used 32 simultaneous AIMD flows over the same topology (the average link utilization was 90% and the average packet loss was 7.6%). We call this dataset Φ_{32}. In both cases, we randomly selected a single flow and focused on its bandwidth samples (this section) and *real-time* estimates (next section).

6.4.2 RBPP Samples

The performance of the RBPP sampling method is shown in Figure 48. The left chart shows the PDF of RBPP samples in Φ_2 and the right chart shows the same in Φ_{32}. Note the different scale on the x-axis.

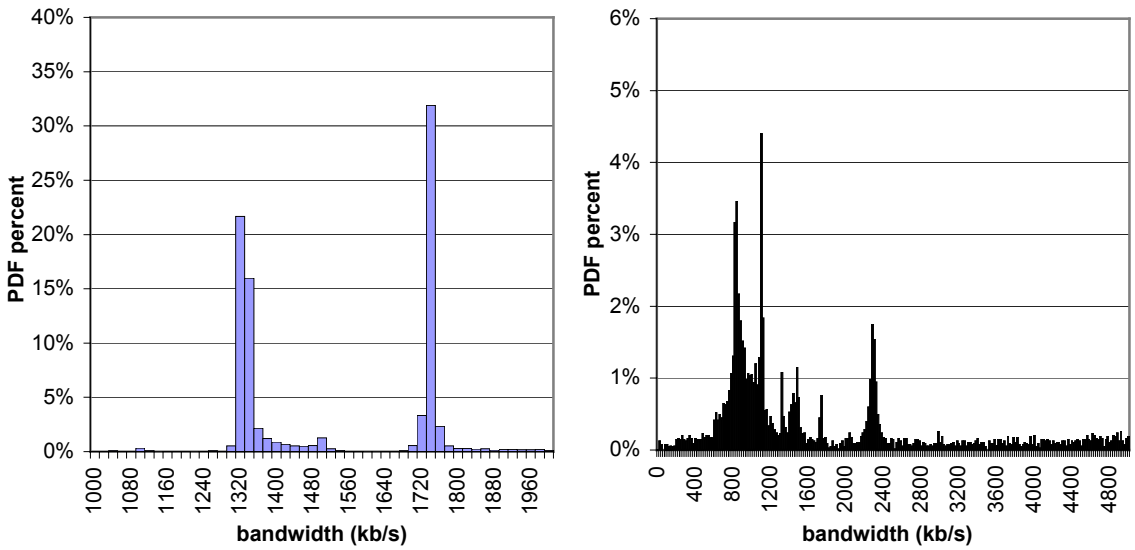

Figure 48. PDF of RBPP samples with 2 (left) and 32 AIMD (right) flows.

First, we notice that the majority of RBPP samples in Φ_2 are clustered around two peaks at 1,320 kb/s (42% of all samples) and 1,740 kb/s (39%). Furthermore, virtually

none of samples are located in the desired area of 1,500 kb/s[44], and over 10% of all samples are higher than 2,000 kb/s (not shown in the figure).

We should note that in all our experiments, we used a clock with resolution 1 microsecond, which was sufficient for rates well over 1 gb/s. Therefore, clock-related round-off errors were not a major factor in the performance of RBPP in our datasets, but they may be in end-systems with low-resolution clocks. Furthermore, the scheduling resolution of the OS kernel may also be an issue affecting the performance of any bandwidth sampling method (e.g., the scheduling clock used in Solaris has a default resolution of 10 ms). In our case, we were unable to determine the exact scheduling resolution of Windows2000, which appeared to deliver packets with resolution no worse than 100 microseconds. Using our packet sizes, this resolution is sufficient for rates up to 18 mb/s, which is also much higher than the speed of the bottleneck link in our case.

Examining inter-packet arrival delays ΔT in RBPP, we found that due to random scheduling delays in Windows 2000, delay ΔT deviated both ways from the ideal value τ (recall that τ is the transmission delay of the second packet of each pair over the bottleneck link). Therefore, if a packet pair was compressed by the OS kernel (i.e., $\Delta T = \tau - \varepsilon$, where ε is some positive error), the next pair was likely to be expanded by the same amount (i.e., $\Delta T \approx \tau + \varepsilon$). This is conceptually illustrated in Figure 49.

---

[44] Even though the physical speed of a T1 line is 1,544 kb/s (1,536 kb/s without the framing bits), the IP level bandwidth given our packet sizes should be between 1,500 and 1,520 kb/s depending on the actual packet size.

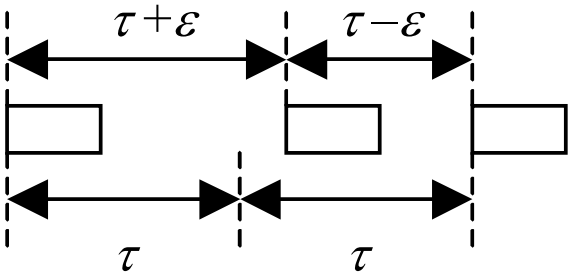

Figure 49. Compression and expansion in the OS kernel.

Since virtually none of the samples in Φ_2 were located at the correct value, an estimation technique that extracts the mode of the PDF of the samples is destined to fail (as we will further see in section 6.5). However, assuming that deviation ε is white Gaussian noise and using Figure 49, it is possible to arrive at the correct value of the bottleneck capacity by inverting each sample (which converts bandwidth units into delay units), averaging all the inverted samples, and then inverting the result (to arrive back into bandwidth units). In other words, if b_i is the i-th estimate of RBPP bandwidth in a dataset and n is the total number of samples, the *inverted average* of a dataset is given by:

$$b_{INV} = \frac{n}{\sum_{i=1}^{n} \frac{1}{b_i}}. \tag{70}$$

Applying this methodology to the entire dataset Φ_2, we find that the inverted average estimate of C was b_{INV} = 1,569 kb/s, which is very reasonable given the scrambled PDF in Figure 48 (left). The performance of this method on partial datasets (i.e., in real-

time) will be studied in section 6.5. We should also mention that the application of the median to the entire dataset, produces an estimate of C equal to 1,715 kb/s and the application of the mode produces an estimate equal to 1,740 kb/s (i.e., both are incorrect values).

Our second observation using Figure 48 is that the performance of RBPP in Φ_{32} gets significantly worse, with individual samples reaching well over 10 mb/s (the figure stops at 5 mb/s). In addition to a large number of over-estimates (42% of the samples above 2 mb/s and 22% above 5 mb/s), the three strong peaks in Φ_{32} are located at incorrect values of 840 kb/s, 1,100 kb/s, and 2,280 kb/s. The right half of Figure 48 shows the effects of both OS scheduling delays, as well as interfering traffic at the bottleneck link. In addition to a larger number of concurrent flows and hence, more interfering traffic, set Φ_{32} is more difficult than Φ_2 for any sampling method due to the lower average sending rate of each flow (i.e., 42 kb/s in Φ_{32} compared to 538 kb/s in Φ_2).

Clearly, computing the mode of the entire dataset Φ_{32} produces an incorrect estimate, which is equal to 1,100 kb/s (i.e., the value of the tallest peak). The use of the median gives a very accurate value of C = 1,537 kb/s and the use of the inverted average described above produces a strong under-estimate equal to 1,206 kb/s. Therefore, it appears that the inverted average is capable of effectively dealing with random OS-related delays, however, it is not suitable for canceling out additional inter-packet *queuing* delays introduced at the bottleneck router (possibly because these delays are not white Gaussian noise and do not follow a symmetric distribution centered at $\varepsilon = 0$).

Estimation results based on RBPP samples are summarized in Table IV below. Assuming that under-estimation is safer than over-estimation (especially for use in congestion control), we note that in RBPP, the inverted average was the safest performer out of the three estimation methods and the median was the most accurate performer.

Dataset	Mode, kb/s	Median, kb/s	Inverted average, kb/s
Φ_2	1,740	1,715	1,569
Φ_{32}	1,100	1,537	1,206

Table IV. Estimation based on RBPP samples and entire datasets.

6.4.3 ERBPP Samples

In the previous section, we discovered that RBPP suffered a significant performance degradation from kernel-related scheduling delays (as was most evident in Φ_2). The central idea behind ERBPP is to allow the OS to balance positive and negative values of delay error ε within each burst and let the application measure the resulting delay of the entire burst. In other words, burst delay ΔT equals the sum of inter-packet delays within the burst, and a single ERBPP bandwidth estimate derived from ΔT simply equals b_{INV} applied to the entire burst (assuming all packets in the burst have the same size). In addition, each ERBPP sample included some form of averaging of the queuing delays of individual packets at the bottleneck link, which was unavailable to RBPP in the previous section.

These two types of averaging explain much better performance of ERBPP in both datasets Φ_2 and Φ_{32} (shown in Figure 50). Φ_2 showed the strongest peak at 1,520 kb/s

and Φ_{32} showed one at 1,500 kb/s. A vast majority of samples (99.9% in Φ_2 and 97% in Φ_{32}) were contained between 1 and 2 mb/s.

The results of applying the three estimation techniques to entire datasets using ERBPP samples are shown in Table V. Since both datasets exhibited a single peak, the mode worked very well producing both estimates in the ballpark of the correct value. The median and the inverted average also worked well, achieving approximately the same result as the mode. Consequently, we conclude that all three estimation methods performed equally well and that the ERBPP sampling method produced a very large number of good samples regardless of the utilization of the bottleneck link or the average sending rate of the flow that measured the bandwidth (i.e., in both Φ_2 and Φ_{32}, ERBPP showed excellent performance).

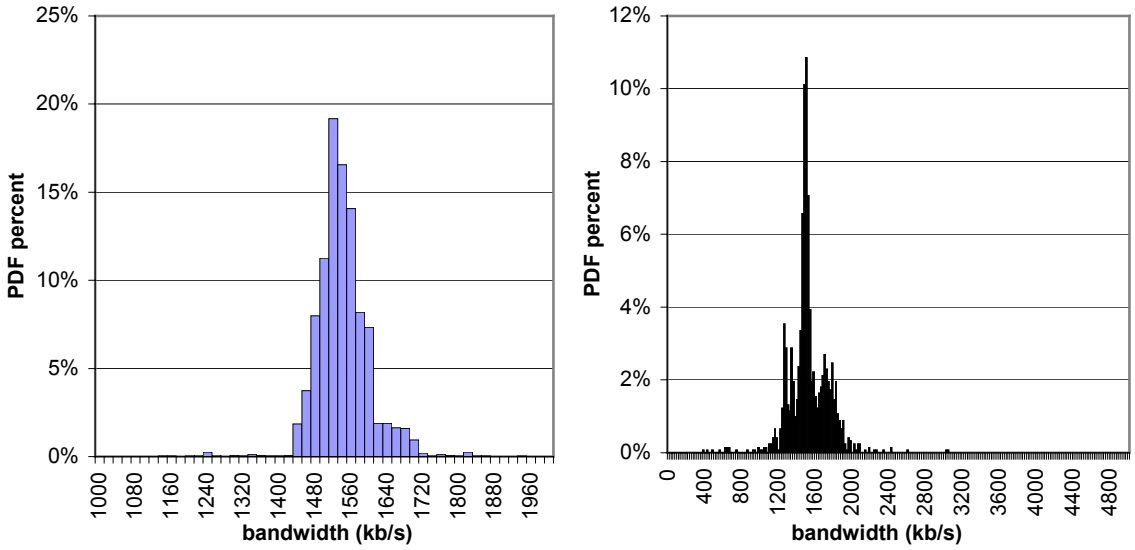

Figure 50. PDF of ERBPP samples with 2 (left) and 32 AIMD (right) flows.

It is suggested in [66] that longer bursts shift the peak of the distribution to values somewhat lower than capacity C and that shorter bursts tend to produce more-accurate samples in a close vicinity of C. We did not observe this phenomenon in our datasets and, in fact, found that larger bursts in ERBPP were typically more reliable, because spacing ΔT was less affected by the scheduling delays.

Dataset	Mode, kb/s	Median, kb/s	Inverted average, kb/s
Φ_2	1,520	1,535	1,538
Φ_{32}	1,500	1,499	1,484

Table V. Estimation based on ERBPP samples and entire datasets.

6.4.4 ERBPP+ Samples

It was expected that ERBPP+ would produce a larger number of under-estimates than ERBPP, because for each burst, ERBPP+ cannot compute a sample larger than the one computed by ERBPP. The PDF of ERBPP+ samples in both datasets is shown in Figure 51. Both datasets exhibited the tallest peak in a close vicinity of the correct value – 1,500 kb/s in Φ_2 and 1,460 kb/s in Φ_{32}; however, there were several additional smaller peaks, as low as 1,160 kb/s in Φ_{32}. Furthermore, ERBPP+ showed a visibly asymmetric distribution in both datasets, with a heavier tail on the left side due to under-estimates. At the same time, the clustering around 1.5 mb/s was still very good and the number of samples between 1 and 2 mb/s was the same as in ERBPP – 99.9% in Φ_2 and 96.5% in Φ_{32}.

We should note that even though our goal is to measure the capacity of the bottleneck link, ERBPP+ samples (which consist of many under-estimates) are much safer to

use for the purposes of congestion control, because they provide a better upper limit on the current sending rate than ERBPP and rarely exceed the actual capacity *C*. Hence, for example, we find that 44% of ERBPP samples in Φ_{32} were over 1,500 kb/s. At the same time, only 13% of ERBPP+ samples in the same dataset were above 1,500 kb/s.

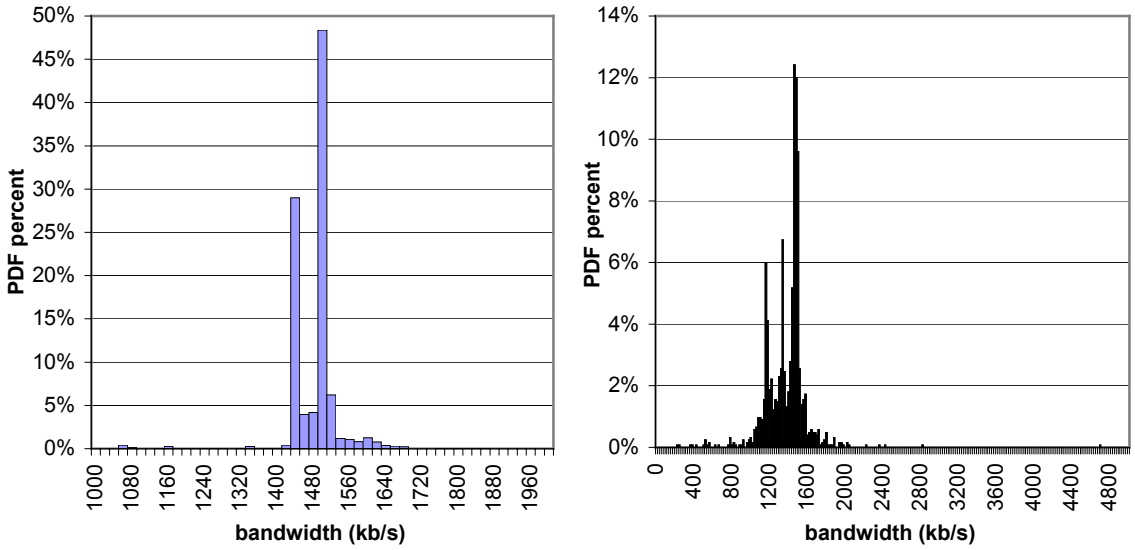

Figure 51. PDF of ERBPP+ samples with 2 (left) and 32 AIMD (right) flows.

Both the mode and the median estimation methods performed reasonably well under the circumstances, keeping the estimation error under 60 kb/s (see Table VI). However, the inverted average was unable to recover from widely varying queuing delays in Φ_{32} and a large number of under-estimates produced by ERBPP+, providing an estimate almost 200 kb/s lower than the actual value of *C* in Φ_{32}.

Dataset	Mode, kb/s	Median, kb/s	Inverted average, kb/s
Φ_2	1,500	1,502	1,480
Φ_{32}	1,460	1,442	1,331

Table VI. Estimation based on ERBPP+ samples and entire datasets.

6.5 Estimation

6.5.1 Introduction

Suppose that set $B(t,k)$ contains k latest bandwidth samples at time t collected by one of the three measurement methods discussed above. The estimation problem that we address in this section is the extraction of a single estimate $b_{EST}(t,k)$ based on set $B(t,k)$, where $b_{EST}(t,k)$ provides the *most recent* estimate of capacity C. This estimation problem is different from the one studied in the last section, because set $B(t,k)$ at any time t contains a *subset* of all samples in the corresponding dataset, and hence, estimates at different times t may exhibit different levels of accuracy (e.g., estimates at the beginning of a session are often less accurate due to the small number of samples in the set). For example, even though an estimate based on the entire dataset may be exact (i.e., 1,500 kb/s), the actual real-time estimates may be equal to 1,100 kb/s for 40% of the session and equal to 1,500 kb/s for 60% of the session. This section will deal with this *real-time* performance of each estimation method.

The use of only k latest samples allows the estimation technique to adapt to changing bottlenecks, as well as discount the history of (possibly) invalid samples (e.g., in cases when the sending rate suddenly goes up, the new estimates generally will be more reliable, and hence, should be allowed to force the old ones out of $B(t,k)$). Note that

in order to maintain a fair comparison between packet-based methods (i.e., RBPP) and burst-based methods (i.e., ERBPP and ERBPP+), we use the following notation when referring to k. In the latter two methods, $B(t,k)$ in fact contains k latest samples, while in the former method, it contains all samples from the last k bursts (i.e., this way, $B(t,k)$ contains all samples collected in the last Δ time units, for some Δ, regardless of the sampling method).

To illustrate the real-time estimation concept, Figure 52 shows a small fragment of a 600-second session of RBPP samples in Φ_2 as a function of time t. The left chart is fitted with the median estimator $b_{EST}(t,k)$ and the right chart is fitted with the inverted average estimator $b_{EST}(t,k)$, for $k = 64$ in both cases. Even though the median of the entire set produced an estimate of 1,715 kb/s, the real-time estimate $b_{EST}(t,k)$ exhibited a certain degree of fluctuation between 1,500 and 1,720 kb/s in Figure 52. At the same time, the inverted average stayed fairly constant at approximately 1,570 kb/s throughout the same segment. However, using the figure, it is not clear how well both methods performed elsewhere within the session, which is the topic of the remainder of the chapter.

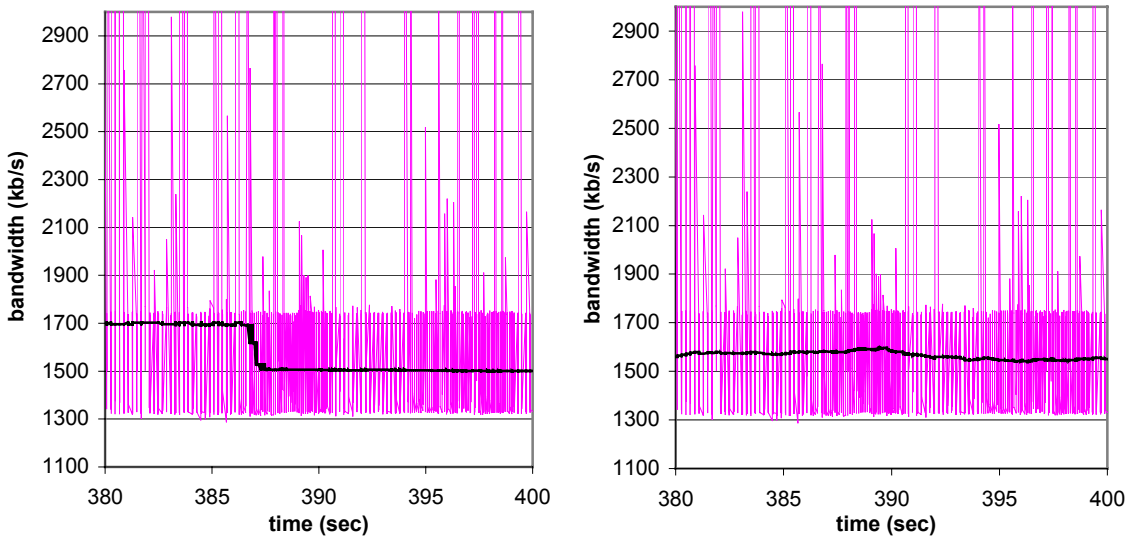

Figure 52. Timeline diagram of RBPP samples in Φ_2. The bold curves are median (left) and inverted average (right) estimates $b_{EST}(t,k)$ for $k = 64$.

6.5.2 Selection of *k*

Clearly, the best value of k (which is a tradeoff between quickly adapting to new bottlenecks and accuracy of estimation) depends on the end-to-end path and transmission rates of the application (i.e., the faster the rate, the more accurate are the samples, and hence, lower values of k are more tolerable). This chapter does not deal with selecting the value of k in real-time, but rather shows the performance of real-time estimation methods for one value of k that was found to be a reasonable compromise between speed of adaptation and accuracy in our datasets. In this section, we briefly explain how we selected such value of k.

Recall that $b_{EST}(t,k)$ is the estimate of capacity C at time t. We elected to use the absolute error criteria for assessing the quality of each estimator. Therefore, the amount

of estimation error at time t is equal to $|b_{EST}(t,k)-C|$ and the *average estimation error* during a session of duration T time units is given by:

$$e(k) = \frac{1}{T}\int_0^T |b_{EST}(t,k) - C| dt \ . \tag{71}$$

By analyzing the values of $e(k)$ for different k between 16 and 512 in our datasets, we found that the average error curve became almost flat at $k = 64$ and the use of values higher than $k = 64$ did not significantly improve the performance of the three estimation methods (i.e., the median, mode, and inverted average). Hence, in the rest of the chapter, we use a fixed value of $k = 64$, which was equivalent to 6-20 seconds worth of samples in Φ_2 and 25-30 seconds in Φ_{32}.

6.5.3 RBPP

The PDF of RBPP estimates based on the *median* estimator of set $B(t,k)$, $k = 64$, is shown in Figure 53. In Φ_2, the median did not perform very well and maintained an in-correct estimate close to 1,720 kb/s for over 70% of the session. At the same time, the vicinity (±40 kb/s) of the peak at 1,500 kb/s contains only 20% of the estimates, and the additional 10% of the estimates are spread out in between the two peaks.

However, in Φ_{32}, the median estimate was equal to the correct value of 1,500 kb/s for 16% of the session (the strongest peak), and the clustering around the peak was very good, with over 50% of the estimates between 1,460 and 1,540 kb/s. We should also note that the average estimation error $e(k)$ was 171 kb/s in Φ_2 and 105 kb/s in Φ_{32}.

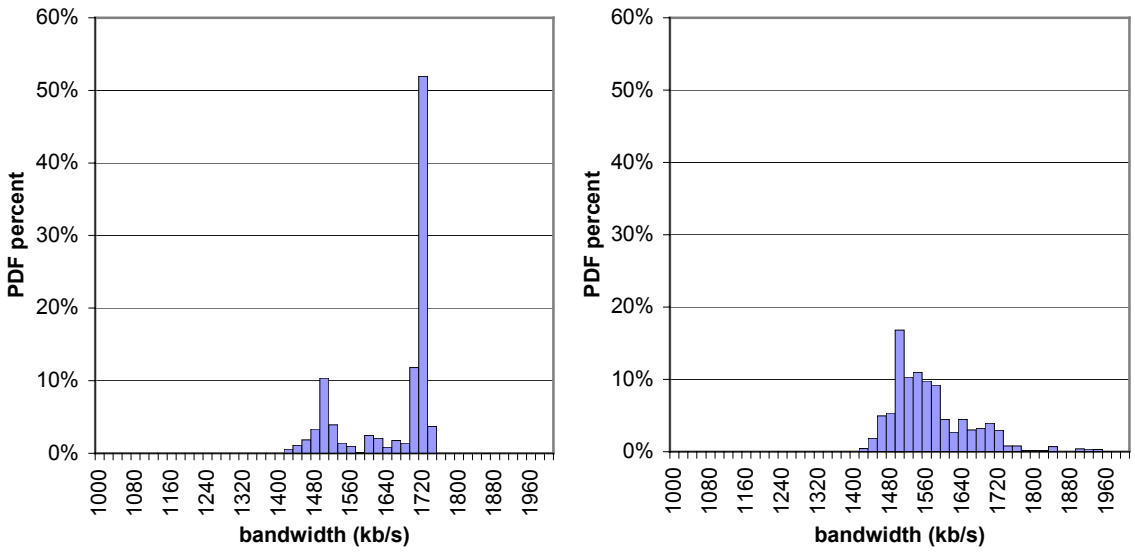

Figure 53. PDF of RBPP *median* estimates in Φ_2 (left) and Φ_{32} (right).

Figure 54 shows the PDF of estimates based on the *mode* estimator in both data-sets. In computing the mode of set $B(t, k)$, we used the bin size equal to 20 kb/s. We found that bin size approximately equal to 1-2% of capacity C was sufficient for identifying the strongest peaks of the distribution of samples and that larger bins did not improve the accuracy of estimation (similar observations were made in [66]).

As expected, the mode did not perform well, because none of the peaks in the distribution of the RBPP samples were located at the correct values. The average estimation error was 250 kb/s in Φ_2 and 523 kb/s in Φ_{32}.

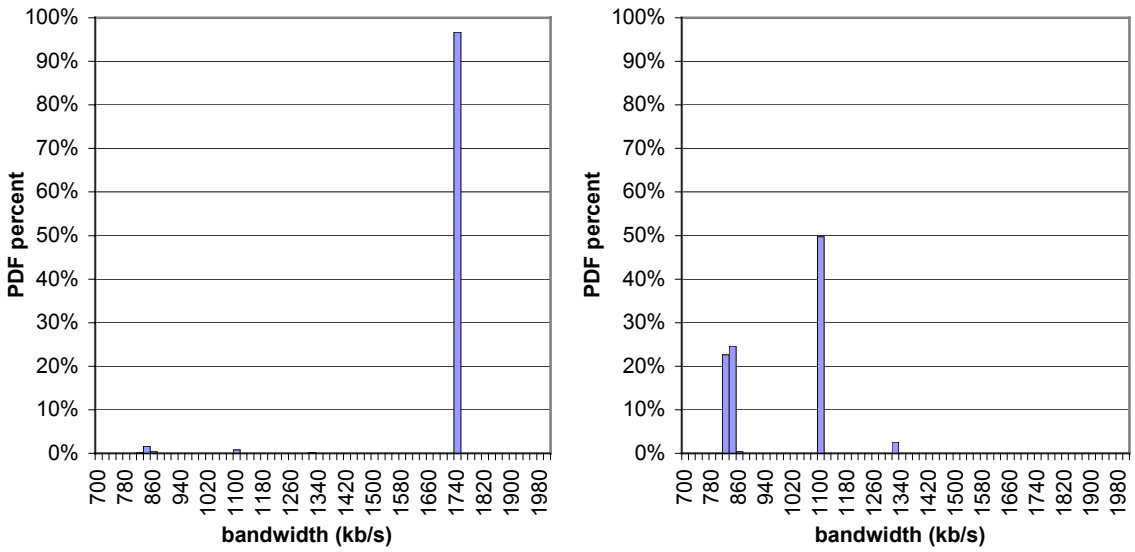

Figure 54. PDF of RBPP *mode* estimates in Φ_2 (left) and Φ_{32} (right).

Figure 55 shows the PDF of estimates produced by the *inverted average* estimator. Note the excellent performance of the method in dataset Φ_2. The method maintained an estimate between 1,540 and 1,620 kb/s for over 93% of the session, with only a few under-estimates as low as 1,300 kb/s. The inverted average performed best out of all three methods in dataset Φ_2; however, it was unable to achieve similar performance in Φ_{32}, where it produced a substantial number of under-estimates (see the figure). In Φ_{32}, only 10% of the estimates were located within 40 kb/s of capacity C, 20% within 100 kb/s, and 40% within 200 kb/s. Thus, the average estimation error was only 87 kb/s in Φ_2, while it remained quite high (i.e., 260 kb/s) in Φ_{32}.

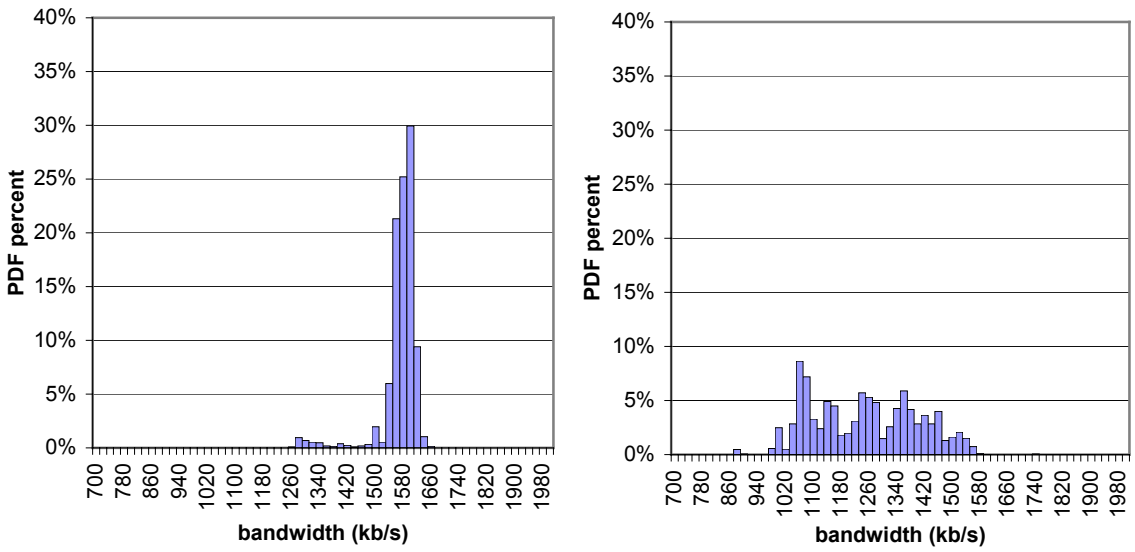

Figure 55. PDF of RBPP *inverted average* estimates in Φ_2 (left) and Φ_{32} (right).

In summary, we find that the median performed best among all three estimation methods, if we rely on the average estimation error $e(k)$ and take both datasets into account (i.e., using the sum of average errors in each dataset). However, if we use estimates $b_{EST}(t,k)$ for the purpose of congestion control, the situation is different. Recall that the median maintained an estimate equal to 1,720 kb/s for over 70% of the session in Φ_2. This is very unsafe, since this estimate is 200 kb/s over the actual capacity of the link. Therefore, assuming that the safest estimator is the best, we find that the inverted average produced estimates that are more desirable from the point of view of a congestion control scheme.

6.5.4 ERBPP

The performance of all three estimation methods based on ERBPP samples was much better, because each sample was more accurate (see the PDF of ERBPP samples in

section 6.4). The PDF of median estimates is shown in Figure 56. In Φ_2, the clustering of estimates was very good – 100% of them were contained between 1,460 and 1,560 kb/s (the average estimation error was 38 kb/s). In Φ_{32}, the clustering was even better – 100% of the estimates between 1,480 and 1,540 kb/s (the average error was 14 kb/s).

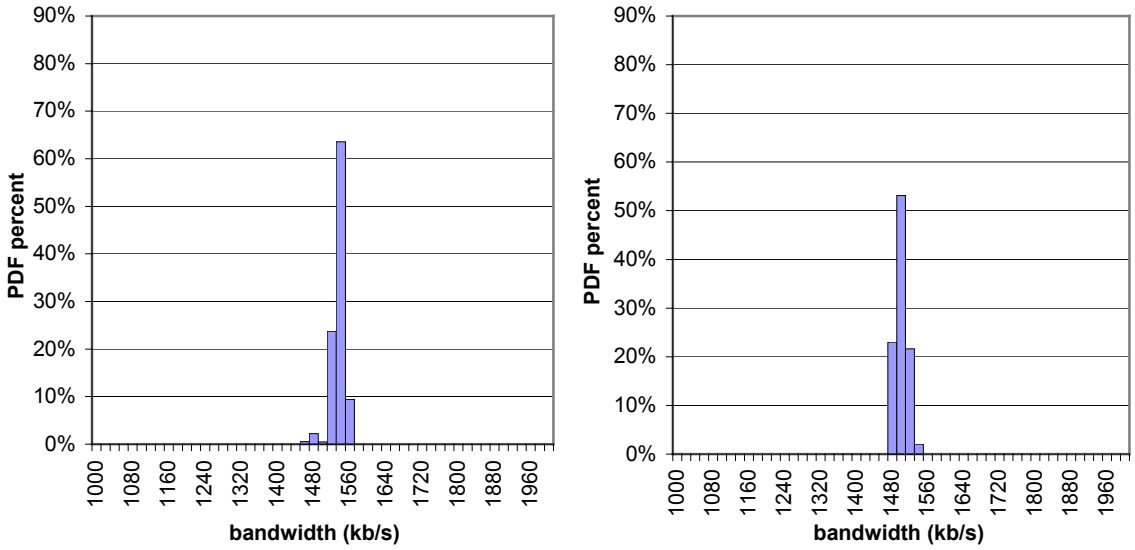

Figure 56. PDF of ERBPP *median* estimates in Φ_2 (left) and Φ_{32} (right).

Figure 57 shows the performance of the mode estimator in the same datasets. The performance of the mode is similar to that of the median, with the exception of a small peak at 1,280 kb/s in Φ_{32}. Virtually all of the remaining estimates were between 1,480 and 1,560 kb/s in both datasets. The average estimation error was 31 kb/s in Φ_2 and 19 kb/s in Φ_{32}.

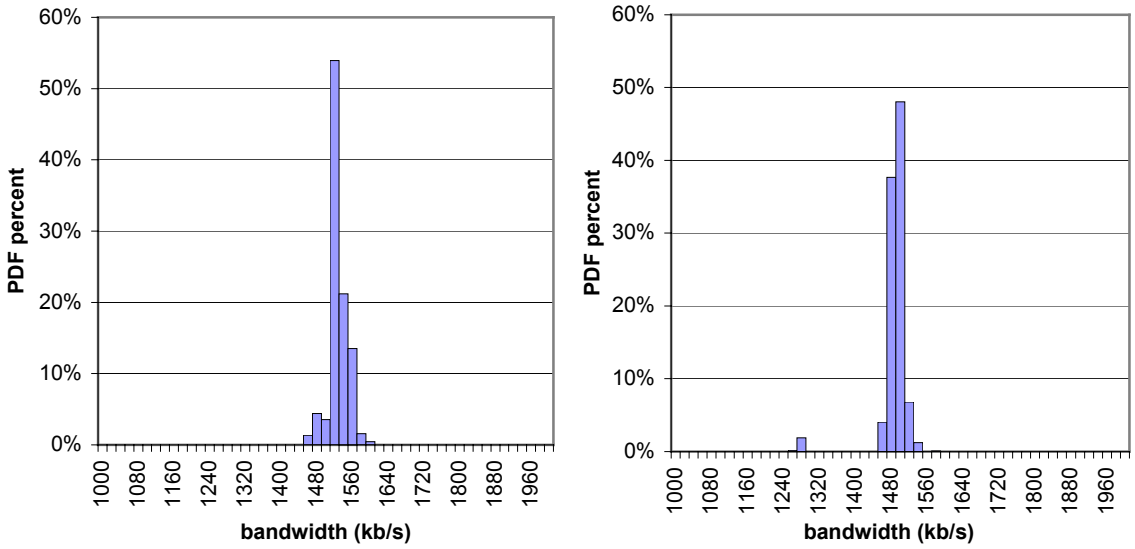

Figure 57. PDF of ERBPP *mode* estimates in Φ_2 (left) and Φ_{32} (right).

Finally, Figure 58 shows the performance of the inverted average in ERBPP. In Φ_2, the clustering of estimates was very good (with the peak at 1,540 kb/s); however, a small number of under-estimates as low as 1,420 kb/s inflated the average error to 42 kb/s (which is still comparable to that produced by the median). In Φ_{32}, a relatively large number of over-estimates skewed the average error to 83 kb/s, even though the majority (i.e., 87%) of samples were contained between 1,440 and 1,520 kb/s.

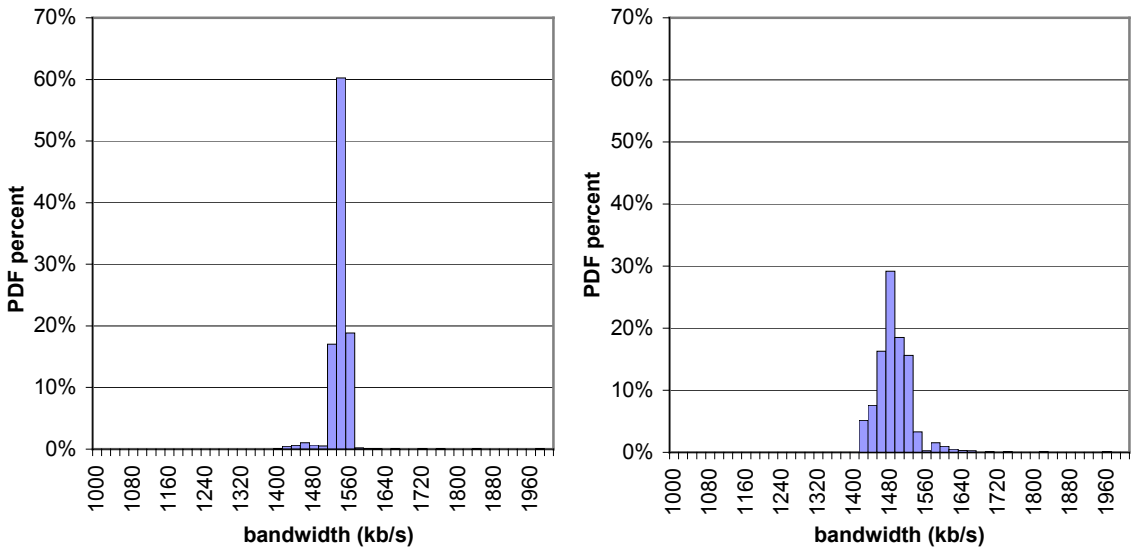

Figure 58. PDF of ERBPP *inverted average* estimates in Φ_2 (left) and Φ_{32} (right).

It appears that the median and the mode both performed very well in ERBPP, and that even the inverted average was able to achieve very good accuracy (with the maximum average error of only 83 kb/s in Φ_{32}). Furthermore, the median was more robust to sporadic outliers than the mode, the latter of which occasionally produced estimates as low as 1,280 kb/s. It is possible to completely rid the mode of these invalid estimates by increasing the number of sample k in set $B(t,k)$; however, given a limited number of samples k, we find that the median performed somewhat better than the mode in ERBPP.

6.5.5 ERBPP+

Remarkably, the median based on ERBPP+ samples performed very well in set Φ_2 as shown in Figure 59. Over 80% of the estimates in Φ_2 were approximately equal to 1,500 kb/s, and the average error was only 8.6 kb/s. However, in Φ_{32}, the situation was

different – *all* estimates were below 1,500 kb/s, 60% of which were above 1,440 kb/s and only 80% above 1,400 kb/s. The resulting average error was 76 kb/s.

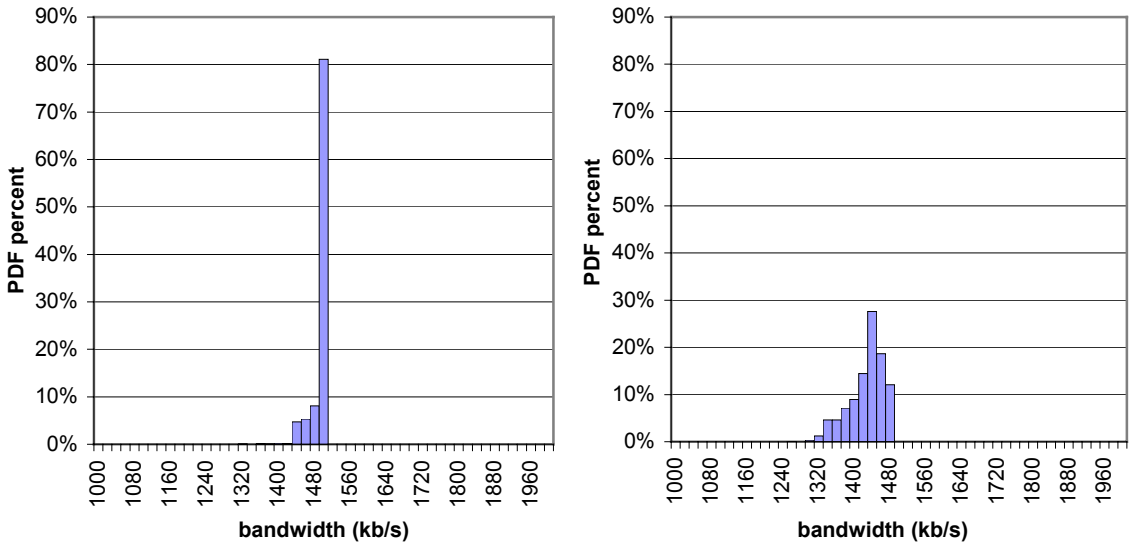

Figure 59. PDF of ERBPP+ *median* estimates in Φ_2 (left) and Φ_{32} (right).

The performance of the mode in ERBPP+ is shown in Figure 60. In Φ_2, the mode maintained very accurate estimates throughout 80% of the session (i.e., the peak at 1,500 kb/s in the figure). However, the remaining 20% of the session, the estimate was quite low (i.e., 1,440 kb/s), and the average error was twice that produced by the median in the same dataset (i.e., $e(k)$ = 15 kb/s). In Φ_{32}, the mode again showed susceptibility to random outliers, producing noticeable peaks at 1,160, 1,180, and 1,340 kb/s. The rest of the estimates were still 20-40 kb/s lower than desired value (which was due to the inherent nature of ERBPP+ to under-estimate capacity C), and the average error was 100 kb/s.

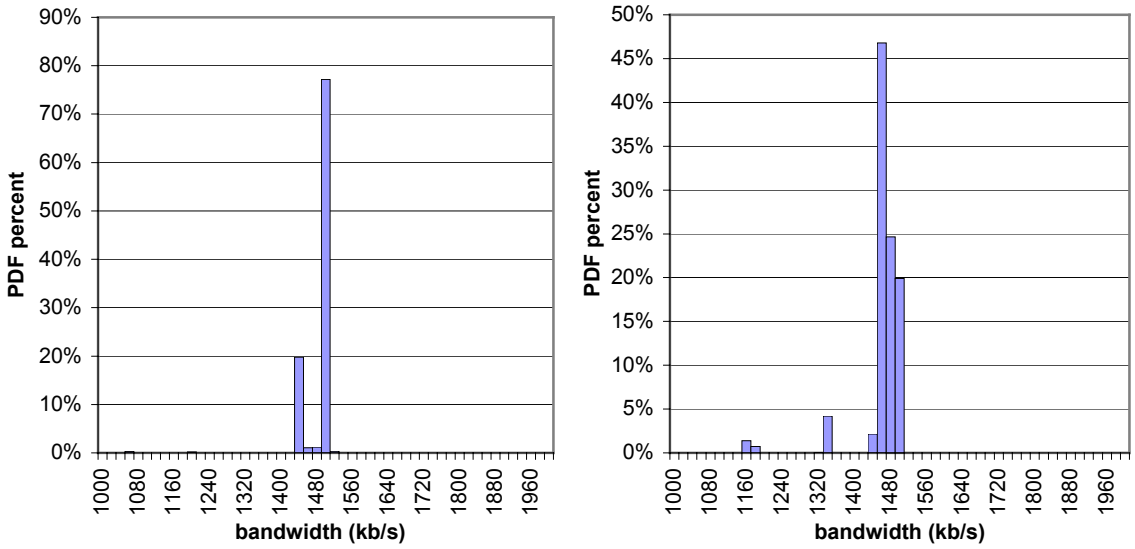

Figure 60. PDF of ERBPP+ *mode* estimates in Φ_2 (left) and Φ_{32} (right).

The inverted average did not do very well in ERBPP+ as can be seen in Figure 61. Even thought its performance is very reasonable in Φ_2 (average error 18 kb/s), it was completely confused by ERBPP+ in Φ_{32} (average error 406 kb/s). One reason for this could be the fact that the samples in ERBPP+ were based on *partial* bursts, often of varying length (where the length refers to both the number of packets that were used to derive partial samples b_m and the corresponding delay ΔT). Hence, the averaging of the corresponding partial delays (i.e., delays that produce the smallest sample b_m within each burst) may not be as meaningful as previously shown on the example of RBPP.

Additionally, the PDF of ERBPP+ samples (as shown in Figure 51, right) was already seriously skewed to the left, and the inverted averaging of such widely varying and often far-from-accurate samples scrambled the result even further as can be seen in Figure 61. Clearly, this effect was more noticeable in Φ_{32}, because random queuing de-

lays played a much bigger role in distorting both inter-packet and inter-burst spacing in ERBPP+ (as can be seen from the PDF of ERBPP+ samples in Figure 51).

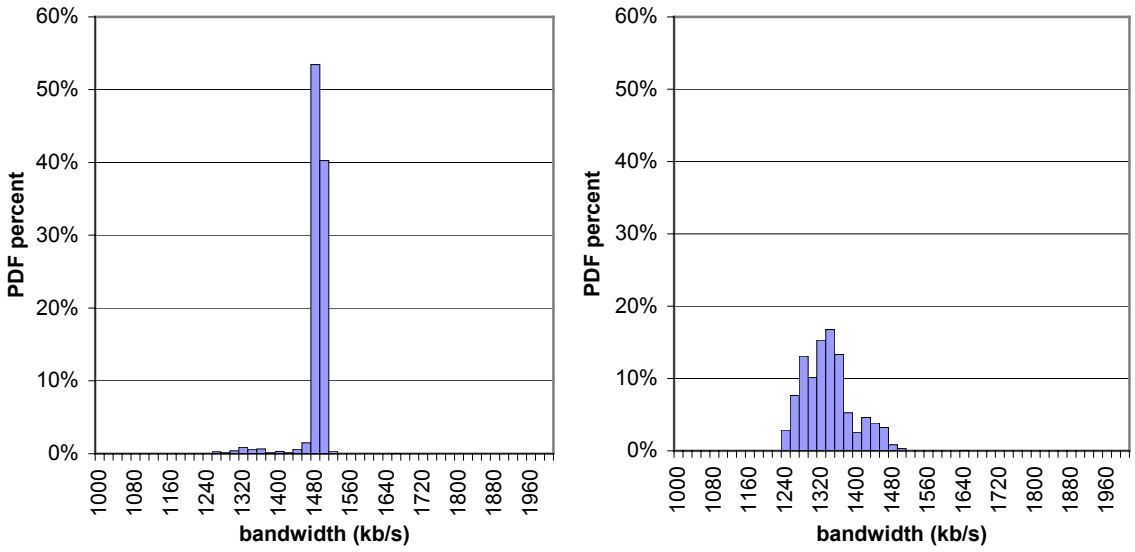

Figure 61. PDF of ERBPP+ *inverted average* estimates in Φ_2 (left) and Φ_{32} (right).

Note again that ERBPP+ rarely produced samples that exceeded 1,500 kb/s, which in turn meant that its estimates were mostly below capacity *C*. Thus, for example, the median ERBPP+ estimator did not produce any estimates above 1,500 kb/s, the mode produced only 0.3%, and the inverted average only 0.4%.

Chapter Seven

7 Conclusion and Future Work

In this chapter, we first summarize the major results of this thesis and then discuss some of the directions of our future work.

7.1 Conclusion

7.1.1 Internet Performance

In Chapter 3, we conducted a large-scale performance study of real-time streaming in the dialup Internet. Our study showed that approximately half of all phone calls to ISPs' access points were unsuccessful and that typical US users needed to *re-dial* connections on average once within the same state in order to sustain the minimum quality of service (QoS) that allowed them to receive low-bitrate Internet video. Our study found that an overwhelming majority of failures experienced by an end-user were caused by modem-to-modem pathologies, which included physical-layer connectivity problems (in-

cluding failed negotiations and 14.4-19.2 kb/s negotiated bitrates), modems that did not pick up the phone, busy phone lines, failed PPP negotiation (including failed user authentication, PPP timeouts, port disconnects by the remote modem, and various other PPP errors), failed traceroute to the server, and various IP-level problems that prevented UDP packets from bypassing the very first router in the ISP network.

Interestingly, we found that large end-to-end delays did not pose much impediment to real-time retransmission during the experiment. The majority (94%) of "recoverable" lost packets returned to the client before their decoding deadlines. We also found that approximately 95% of all recovered packets were recovered using a single retransmission attempt (i.e., a single NACK). Nevertheless, we find that the current end-to-end delays in the dialup Internet are prohibitively high to support interactive real-time applications (such as video conferencing or telephony). For non-interactive applications, startup delays in the order of 10-15 seconds are recommended given a random access point used in streaming; however, startup delays as low as one second were found to be acceptable over certain paths during the night. Even using forward error correction (FEC) coding instead of retransmission to overcome packet loss does not allow us to substantially lower the startup delay, because one-way delay jitter in the dialup Internet is often very large.

We speculate that end-to-end delays under 100 ms will be required to support interactive streaming, which seems to be currently possible with DSL and certain cable modems. Furthermore, we believe that with broadband access at home, the performance of real-time streaming will largely depend on the end-to-end congestion control em-

ployed in the streaming protocol, rather than on the backbone Internet packet-loss rates, a particular retransmission scheme, or delay jitter (all of which are significantly less relevant given low end-to-end delays). Hence, in the future, it is very important to develop congestion control suitable for real-time multimedia flows that scales to a large number of concurrent users and that can be deployed incrementally with the existing TCP flows (i.e., it should possess some form of TCP-friendliness).

Even though ACK-based congestion control [110] for TCP-like flows is understood fairly well, it is still not clear whether ACK-based flow control is suitable for real-time flows. On the other hand, even though NACK-based flow control is great for rate-based applications, its "open-loop" operation makes it much more unstable and less scalable [157]. We believe that future research should first address these congestion control issues before real-time streaming becomes widely available in the Internet.

7.1.2 Real-time Retransmission

In Chapter 4, we studied the suitability of NACK-based retransmission for recovering lost packets in real-time streaming applications and examined the performance of several classes of retransmission timeout (RTO) estimators based on the traces of our large-scale Internet experiment. Current real-time streaming applications [232] rely on NACK-based retransmission and often do not implement congestion control. The nature of NACK-based retransmission and the lack of congestion control in these applications suggest that the current RTO estimation methods implemented in TCP may not be adequate for NACK-based streaming protocols. Furthermore, even the existing RTO estima-

tion methods in TCP do not have a rigorous performance evaluation model, and their performance over diverse Internet paths remains unknown.

Our study introduced a novel performance measure (suitable for both NACK and ACK-based protocols), which captures the accuracy of hypothetical RTO estimators based on packet data traces of real Internet connections. This performance measure shows the inherent tradeoff between the number of duplicate packets generated by an estimator and the amount of unnecessary waiting for timeouts given the data traces.

Based on our performance measure, we found that in the dialup Internet (which is often accompanied by low packet loss [156]), TCP-like estimators were not optimal in traditional NACK-based protocols due to the large distance between RTT samples. We further established that in the dialup Internet, the latest RTT had the most relevance to the future RTTs and that EWMA smoothing of either RTT samples or RTT variance did not increase the accuracy of estimation. This result suggests that large distances between RTT samples (in the order of 15 seconds) allow end-to-end network conditions in the Internet to change significantly, which leads us to conclude that sampling rates of higher frequency may be required to adequately sample the RTT in the current Internet.

Furthermore, we found that frequent delay jitter samples were very useful in fine-tuning the RTO estimation between the measurements of the RTT. Delay-jitter estimators were found to perform much better in the modem traces; however, their benefits were virtually nullified when the RTT was sampled at a higher frequency.

As our results show, the performance of TCP-like estimators depends on the sampling frequency of the RTT. Consequently, we conclude that there is enough evidence to

suggest that the paradigm in which NACK-based applications sample the RTT only at times of packet loss may not be very useful. We find that higher-frequency sampling of the RTT may be necessary for accurate RTO estimation and could be additionally used for other purposes (such as equation-based congestion control [86]). Our experiments with NACK-based congestion control show that RTT sampling rates of once-per-RTT can be achieved with very little overhead (i.e., the measurement of the RTT can be incorporated into the congestion control feedback loop). In such scenarios, our study found that a scaled *SRTT* estimator was optimal and even TCP's RTO was sufficiently accurate.

7.1.3 Scalable Rate-based Congestion Control

In Chapter 5, we analyzed the problem of scaling rate-based (or NACK-based) congestion control methods in streaming applications to a large number of flows. The difficulty of rate-based congestion control stems from the fact that the sender in such protocols is not governed by "self-clocking" of acknowledgements and typically continues to stress the network at the same rate even in the presence of severe packet loss and congestion. In such situations of aggravated packet loss, the main problem of NACK-based congestion control can be narrowed down to cases when the client either does not receive any server packets at all (which by default prohibits it from changing the server's rate), or takes multiple retransmissions of control messages to notify the server about the new reduced rate.

Interestingly, these problems are only noticeable when the congestion is severe enough to require multiple retransmissions of the client's control messages, or when the

network encounters periods of *heavily-bursty* loss. Our experiments with traditional (i.e., non-scalable) NACK-based congestion control methods found that packet loss rates increased very rapidly as the number of flows on the shared link increased.

To investigate this observation further, we analyzed the class of binomial algorithms and derived the formulas of packet loss increase factor s_n as a function of the number of flows: $s_n = O(n^{l+2k+1})$. Using our derivations we found that among all proposed binomial schemes, AIMD had the best scalability $O(n^2)$ and the lowest packet loss. Furthermore, we showed that unless the schemes had the knowledge of bottleneck capacity C, the scalability of AIMD could not be improved, and even the performance of AIMD was inadequate for actual use in NACK-based applications. Even though all the derivations in the chapter assumed synchronized and immediate feedback, our final formulas were found to hold in a number of streaming experiments over a real Cisco network with random packet loss and delayed feedback.

Given the knowledge of the bottleneck bandwidth, we showed that ideal scalability was both theoretically and practically possible; however, the ISCC schemes were found to be slower in their convergence to fairness when the number of flows n was large. Even though ISCC schemes are "more careful" in probing for new bandwidth, the average efficiency of these schemes was no worse than that of AIMD or IIAD.

Regardless of whether ISCC is a viable protocol for the current or future (i.e., DiffServ) Internet, Chapter 5 not only answered the question of why NACK-based congestion control is "difficult," but it also measured the exact magnitude of this "difficulty"

and provided one solution that overcomes a rapid packet-loss increase typical to "open-loop" congestion control.

7.1.4 Real-time Estimation of the Bottleneck Bandwidth

In Chapter 6, we introduced the problem of estimating the capacity of the bottleneck link of an end-to-end path in real time and suggested that these estimates be used in rate-based congestion control. We found that the sampling methods that rely on multiple back-to-back packets (i.e., ERBPP and ERBPP+) performed significantly better than the ones that rely on the packet-pair concept (i.e., RBPP) in a wide range of scenarios. We tracked the problem with RBPP to random OS scheduling delays that often distorted the spacing between arriving packets. We should further note that at higher sending rates (i.e., T3 and up), significantly more packets will be required in each burst to reliably estimate transmission delay τ of the burst over the bottleneck link, and the performance of RBPP will get progressively worse with the increase in the speed of the bottleneck capacity.

In addition, we found that the ERBPP+ sampling method had clear advantages when used over multi-channel links compared to ERBPP, but did in fact exhibit lower accuracy over single-channel links. We also found that the majority of ERBPP+ samples were below the actual capacity of the link and were safer to be employed in congestion control than ERBPP samples even though they were less accurate.

Among real-time estimation methods, it appears that the median was more robust against random outliers than the mode when used in all three sampling methods (i.e.,

RBPP, ERBPP, and ERBPP+). In addition, we found that the inverted average performed very poorly in ERBPP+ and that its advantages were limited to RBPP in sessions with a small number of concurrent flows (i.e., cases similar to the one in Φ_2).

Furthermore, we noticed that the computational complexity of the median was the highest (i.e., it involved a sorting operation of complexity $O(k\log k)$), and that the mode and the inverted average worked equally fast (i.e., in both cases, the complexity was a linear function of k). However, the mode required extra memory to maintain the histogram, whereas the inverted average did not require any memory overhead. Therefore, we conclude that the performance of each estimation technique was directly related to the cost of its implementation – the inverted average had the lowest cost and lowest performance, while the median had the highest cost and highest performance. The same observation applies to the size of set $B(t,k)$ – larger sets typically resulted in more accurate estimation and, at the same time, more computational overhead (i.e., because k was larger). In our datasets, we found that the improvement in accuracy after $k = 64$ was negligible, however, this conclusion may not hold for other network paths and/or end-to-end protocols.

7.2 Future Work

In the end, we find that the fundamental question of whether ACK-based streaming with extra startup delays is a better streaming solution than NACK-based congestion control with its potential instability remains an open issue. Even though our work suc-

ceeded at designing scalable NACK-based congestion control, its performance in the presence of heavy packet loss is still not as robust as that of its ACK-based counterparts. Hence, we are interested in investigating the exact penalty (in terms of extra startup delays or reduced video quality) that a real-time application needs to pay to run with ACK-based congestion control. Thinking further in the same direction, it appears that some form of hybrid ACK-NACK congestion control may be a required compromise that can supply video applications with a rate-based flow control needed for real-time streaming and ACK-based congestion control needed for stability.

We have further interest in studying the performance of real-time streaming in multicast and wireless environments. In these cases, both congestion control and lost packet recovery are very different and require future work. Congestion control in multicast does not rely on a single-receiver feedback but rather depends on a *set* of receiver feedbacks, which makes it difficult to decide what model of "group happiness" (e.g., satisfying the slowest, or possibly the fastest, receiver) is most appropriate. Furthermore, in certain wireless scenarios, packet loss is induced by bit errors and RF interference rather than overflown router buffers and congestion. Hence, even in the presence of MAC-layer forward error correction (FEC) and retransmission, congestion control over wireless links may face additional challenges of distinguishing between congestion-related and bit-error-related packet loss and extra delays. In addition, under high bit-error rates, MAC-layer error recovery has the effect of clustering (i.e., compressing) the packets inside the NIC, which presents a very difficult problem for end-to-end bandwidth estimation methods.

Finally, we would like to study future protocols capable of automatically distributing the streaming load between a set of servers placed at various locations in the Internet and finding the nearest streaming server that matches the minimum requirements (which could be based on the available bandwidth, packet loss, and/or end-to-end delay) of the end-user. Ideally, this work will lead to designing a platform for large-scale real-time video streaming in the Internet and identifying what Quality of Service (QoS) framework is needed in the current Internet to reliably support multi-megabit per second (i.e., broadcast-quality) streaming to home users.

Bibliography

[1] A. Acharya and J. Saltz, "A Study of Internet Round-trip Delay," *Technical Report CS-TR-3736*, University of Maryland, December 1996.

[2] B. Ahlgren, M. Björkman, and B. Melander, "Network Probing Using Packet Trains," *Swedish Institute of Computer Science Technical Report*, March 1999.

[3] J.S. Ahn, P.B. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: Emulation and Experiment," *ACM SIGCOMM*, 1995.

[4] M. Allman and V. Paxson, "On Estimating End-to-End Network Parameters," *ACM SIGCOMM*, September 1999.

[5] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *IETF RFC 2581*, April 1999.

[6] P. Almquist, "Type of Service in the Internet Protocol Suite," *IETF RFC 1349*, July 1992.

[7] E. Amir, S. McCanne, and R. Katz, "Receiver-driven Bandwidth Adaptation for Light-weight Sessions," *ACM Multimedia*, 1997.

[8] E. Amir, S. McCanne, and H. Zhang, "An Application-level Video Gateway," *ACM Multimedia*, November 1995.

[9] S. Bajaj, L. Breslau, and S. Shenker, "Uniform Versus Priority Dropping for Layered Video," *ACM SIGCOMM*, September 1998.

[10] F. Baker, "Requirements for IP Version 4 Routers," *IETF RFC 1812*, June 1995.

[11] H. Balakrishnan, V.M. Padmanablah, S. Seshan, M. Stemm, and R.H. Katz, "TCP Behavior of a Busy Internet Server: Analysis and Improvements," *IEEE INFOCOM*, March 1998.

[12] H. Balakrishnan, H.S. Rahul, and S. Seshan, "An Integrated Congestion Management Architecture for Internet Hosts," *ACM SIGCOMM*, September 1999.

[13] D. Bansal and H. Balakrishnan, "Binomial Congestion Control Algorithms," *IEEE INFOCOM*, April 2001.

[14] D. Bansal, H. Balakrishnan, S. Floyd, S. Shenker, "Dynamic Behavior of Slowly-Responsive Congestion Control Algorithms," *ACM SIGCOMM*, August 2001.

[15] S. Basu, A. Mukherjee, and S. Klivansky, "The Series Models for Internet Traffic," *IEEE INFOCOM*, 1996.

[16] L. Benmohamed and S.M. Meerkov, "Feedback Control of Congestion in Packet Switching Networks: The Case of a Single Congestion Node," *IEEE/ACM Transactions on Networking*, vol. 1, no. 6, December 1993.

[17] J.C.R. Bennett and H. Zhang, "Hierarchical Packet Fair Queueing Algorithm," *ACM SIGCOMM*, 1996.

[18] J. Beran, "A goodness-of-fit test for time series with long-range dependence," *J. Royal Statistical Soc. B*, vol. 54, no. 3, 1992, 749-760.

[19] J. Bolliger, T. Gross, and U. Hengartner, "Bandwidth Modeling for Network-Aware Applications," *IEEE INFOCOM*, 1999.

[20] J. Bolliger, U. Hengartner, and T. Gross, "The Effectiveness of End-to-End Congestion Control Mechanisms," *Technical Report #313*, Department of Computer Science, ETH Zurich, February 1999.

[21] J. Bolot, "Analysis and Control of Audio Packet Loss over Packet-Switched Networks," *IEEE Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 1995.

[22] J. Bolot, "Characterizing End-to-End Packet Delay and Loss in the Internet," *Journal of High Speed Networks*, vol. 2, no. 3, September 1993, 289-298. (*also appeared as "End-to-End Packet Delay and Loss Behavior in the Internet" in ACM SIGCOMM, September 1993*).

[23] J. Bolot, "Cost-Quality Tradeoffs in the Internet," *Computer Networks and ISDN Systems*, vol. 28, 1996, 645-651.

[24] J. Bolot and A.U. Shankar, "Optimal Least-Square Approximations to the Transient Behavior of the Stable M/M/1 Queue," *IEEE Transactions on Communications*, vol. 43, no. 4, pp. 1293-1298, April 1995.

[25] J. Bolot and T. Turletti, "A Rate Control Mechanism for Packet Video in the Internet," *IEEE INFOCOM*, June 1994, 1216-1223.

[26] J. Bolot and T. Turletti, "Experience with Rate Control Mechanisms for Packet Video in the Internet," *ACM Computer Communication Review*, January 1998, 4-15.

[27] J. Bolot, S. Fosse-Parisis, and D. Towsley, "Adaptive FEC-based Error Control for Interactive Audio in the Internet," *IEEE INFOCOM*, March 1999.

[28] J. Bolot, T. Turletti, and I. Wakeman, "Scalable Feedback for Multicast Video Distribution in the Internet," *ACM SIGCOMM*, August 1994.

[29] M.S. Borella, D. Swider, S. Uludag, and G.B. Brewster, "Internet Packet Loss: Measurement and Implications for End-to-End QoS," *International Conference on Parallel Processing*, August 1998.

[30] M.S. Borella, S. Uludag, G.B. Brewster, I. Sidhu, "Self-similarity of Internet Packet Delay," *IEEE ICC*, August 1997.

[31] J.M. Boyce and R.D. Gaglianello, "Packet Loss Effects on MPEG Video Sent Over the Public Internet," *ACM Multimedia*, 1998.

[32] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet," *IETF RFC 2309*, April 1998.

[33] R. Braden, editor, "Requirements for Internet Hosts -- Communication Layers," *IETF RFC 1122*, October 1989.

[34] R. Braden and J. Postel, "Requirements for Internet Gateways," *IETF RFC 1009*, June 1987.

[35] L. Brakmo and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, October 1995, 1465-1480 (*earlier version in ACM SIGCOMM, 1994*).

[36] R.P. Brent, "Algorithms for Minimization without Derivatives," *Englewood Cliffs*, NJ, Prentice Hall, 1973.

[37] L. Breslau and S. Shenker, "Best-Effort versus Reservations: A Simple Comparative Analysis," *ACM SIGCOMM*, September 1998.

[38] R. Bruyeron, B. Hemon, and L. Zhang, "Experimentations with TCP Selective Acknowledgment," *ACM Computer Communication Review*, vol. 28, no.2, April 1998.

[39] H. Bryhni, H. Lovett, and E. Maartmann-Moe, "On-Demand Regional Television over the Internet," *ACM Multimedia*, 1996.

[40] R. Caceres, P.B. Danzig, S. Jamin, and D. Mitzel, "Characteristics of Wide-Area TCP/IP Conversations," *ACM SIGCOMM*, 1991.

[41] K.L. Calvert, M.B. Doar, and E.W. Zegura, "Modeling Internet Topology," *IEEE Communications Magazine*, June 1997.

[42] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP Latency," *IEEE INFOCOM*, March 2000.

[43] G. Carle and E.W. Biersack, "Survey of Error Recovery Techniques for IP-Based Audio-Visual Multicast Applications," *IEEE Network*, November/December 1997, 24-36.

[44] R.L. Carter, and M.E. Crovella, "Measuring Bottleneck Link Speed in Packet Switched Networks," *International Journal on Performance Evaluation* 27&28, 1996, 297-318.

[45] S. Cen, C. Pu, and J. Walpole, "Flow and Congestion Control for Internet Streaming Applications," *Multimedia Computing and Networking*, January 1998.

[46] V.G. Cerf, and R.E. Kahn, "A protocol for packet network interconnection," *IEEE Transactions on Communications*, vol. 22, no. 5, May 1974, pp. 637-648.

[47] L. Chiariglione, "MPEG and Multimedia Communications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 1, February 1997.

[48] B. Chinoy, "Dynamics of the Internet Routing Information," *ACM SIGCOMM*, September 1993.

[49] D. Chiu and R. Jain, "Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks," *Journal of Computer Networks and ISDN Systems*, vol. 17, no. 1, June 1989, 1-14.

[50] M. Christiansen, K. Jeffay, D. Ott, and F.D. Smith, "Tuning RED for Web Traffic," *ACM SIGCOMM*, August 2000.

[51] I. Cidon, A. Khamisy, and M. Sidi, "Analysis of Packet Loss Processes in High-speed Networks," *IEEE Trans. Info. Theory*, vol. 39, no. 1, January 1993, 98-108.

[52] K.C. Claffy, G. Miller, and K. Thompson, "The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone," *INET*, Internet Society, December 1998.

[53] K.C. Claffy, G.C. Polyzos, and H-W. Braun, "Measurement Considerations for Assessing Unidirectional Latencies," *Internetworking: Research and Experience*, vol. 4, 1993, 121-132.

[54] K.C. Claffy, G.C. Polyzos, and H-W. Braun, "Application of Sampling Methodologies to Network Traffic Characterization," *ACM SIGCOMM*, 1993.

[55] K.C. Claffy, G.C. Polyzos, and H-W. Braun, "Traffic Characteristics of the T1 NSFNET Backbone," *IEEE INFOCOM*, 1993.

[56] D.D. Clark, M.L. Lambert, and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol," *IETF RFC 998*, March 1987.

[57] D.D. Clark, M.L. Lambert, and L. Zhang, "NETBLT: A High Throughput Transport Protocol," *ACM SIGCOMM*, 1987, 353-359.

[58] D.D. Clark, S. Shenker, and L. Zhang, "Supporting Real-time Applications in an Integrated Services Packet Network: Architecture and Mechanisms," *ACM SIGCOMM*, August 1992.

[59] M. Claypool, "The Effects of Jitter on the Perceptual Quality of Video," *ACM Multimedia*, October 1999.

[60] M. Crovella, and A. Bestavros, "Self-similarity in World Wide Web Traffic, Evidence and Possible Causes," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, December 1997 (*earlier paper in SIGMETRICS 96*).

[61] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *ACM SIGCOMM*, September 1989, 1-12.

[62] B. Dempsey, J. Liebeherr, and A. Weaver, "A New Error Control Scheme for Packetized Voice over High-Speed Local Area Networks," *18th IEEE Local Computer Networks Conference*, September 1993.

[63] B. Dempsey, J. Liebeherr, and A. Weaver, "On retransmission-based error control for continuous media traffic in packet-switching networks," *Computer Networks and ISDN Systems*, vol. 28, no. 5, March 1996, 719-736.

[64] H.R. Deng and M.L. Lin, "A type II hybrid ARQ system with adaptive code rates," *IEEE Transactions on Communications*, COM 43(2/3/4), Feb./Mar./Apr. 1995, 733-737.

[65] W. Ding, "Joint Encoder and Channel Rate Control of VBR Video over ATM Networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 2, April 1997.

[66] C. Dovrolis, P. Ramanathan, D. Moore, "What Do Packet Dispersion Techniques Measure?" *IEEE INFOCOM*, April 2001.

[67] C. Dovrolis, D. Stiliadis, and P. Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," *ACM SIGCOMM*, September 1999.

[68] A.B. Downey, "Using PATHCHAR to Estimate Internet Link Characteristics," *ACM SIGCOMM*, September 1999.

[69] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communication Review*, vol. 26, no. 3, July 1996, 5-21.

[70] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," *ACM SIGCOMM*, September 1999.

[71] A. Feldman, A.C. Gilbert, and W. Willinger, "Data Networks as Cascades: Investigating the Multifractal Nature of Internet WAN Traffic," *ACM SIGCOMM*, September 1998, 42-55.

[72] A. Feldman, A.C. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP Traffic: A Study of the Role of Variability and the Impact of Control," *ACM SIGCOMM*, September 1999.

[73] K.W. Fendick, M.A. Rodriguez, and A. Weiss, "Analysis of a Rate-Based Control Strategy with Delayed Feedback," *ACM SIGCOMM*, 1992.

[74] W. Feng, D. Kandlur, D. Saha, and K. Shin, "A Self-configuring RED Gateway," *IEEE INFOCOM*, 1999.

[75] W. Feng, M. Liu, B. Krishnaswami, and A. Prabhudev, "A Priority-based Technique for the Best-effort Delivery of Stored Video," *Multimedia Computing and Networking*, January 1999.

[76] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Understanding and Improving TCP Performance Over Networks with Minimum Rate Guarantees," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, April 1999.

[77] N.R. Figueira and J. Pasquale, "Leave-in-time: A New Service Discipline for Real-time Communications in a Packet-switching Network," *ACM SIGCOMM*, September 1995, 207-218.

[78] S. Floyd, “Connections with Multiple Congested Gateways in Packet Switched Networks,” *ACM Computer Communication Review*, vol. 21, no. 5, October 1991, 30-47.

[79] S. Floyd, “TCP and Explicit Congestion Notification,” *ACM Computer Communication Review*, vol. 24, no. 5, October 1994, 10-23.

[80] S. Floyd and K. Fall, “Promoting the Use of End-to-End Congestion Control in the Internet,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, August 1999.

[81] S. Floyd, M. Handley, and J. Padhye, “A Comparison of Equation-Based and AIMD Congestion Control,” *ACIRI Technical Report* http://www.aciri.org/tfrc/aimd.pdf, May 2000.

[82] S. Floyd and T. Henderson, “The NewReno Modifications to TCP’s Fast Recovery Algorithm,” *IETF RFC 2582*, April 1999.

[83] S. Floyd and V. Jacobson, “On Traffic Phase Effects in Packet-Switched Gateways,” *ACM Computer Communication Review*, vol. 21, no. 2, April 1991.

[84] S. Floyd and V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance,” *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, August 1993, 397-413.

[85] S. Floyd and V. Paxson, “Why We Don't Know How To Simulate The Internet,” *Proceedings of the 1997 Winter Simulation Conference*, December 1997.

[86] S. Floyd, M. Handley, and J. Padhye, “Equation-Based Congestion Control for Unicast Applications,” *ACM SIGCOMM*, September 2000.

[87] S. Floyd, V. Jacobson, S. McCanne, C.G. Liu, and L. Zhang, “A reliable multicast framework for lightweight sessions and application level framing,” *ACM SIGCOMM*, 1995, 342-356.

[88] M.W. Garrett and M. Vetterli, “Joint Source/Channel Coding of Statistically Multiplexed Real-Time Services on Packet Networks,” *IEEE/ACM Transactions on Networking*, vol. 1, no. 1, February 1993.

[89] M.W. Garrett and W. Willinger, “Analysis, Modeling, and Generation of Self-Similar VBR Video Traffic,” *ACM SIGCOMM*, 1994.

[90] S.J. Golestani and S. Bhattacharyya, “A Class of End-to-End Congestion Control Algorithms for the Internet,” *IEEE International Conference on Network Protocols*, 1998.

[91] F. Gong and G.M. Parulkar, "An Application-Oriented Error Control Scheme for High-Speed Networks," *IEEE/ACM Transactions on Networking*, vol. 4, no. 5, October 1996.

[92] R. Govindan and A. Reddy, "An Analysis of the Internet Inter-Domain Topology and Route Stability," *IEEE INFOCOM*, April 1997.

[93] M. Grossglauser and J-C. Bolot, "On the Relevance of Long-Range Dependence in Network Traffic," *ACM SIGCOMM*, 1996.

[94] P.K. Gummadi, S. Saroiu, S.D. Gribble, "A Measurement Study of Napster and Gnutella as Examples of Peer-to-Peer File Sharing Systems," *Poster at ACM SIGCOMM*, August 2001.

[95] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," *ACM SIGCOMM*, September 1999.

[96] R. Gupta, M. Chen, S. McCanne, and J. Walrand, "WebTP: A Receiver-Driven Web Transport," *University of California at Berkeley Technical Report*, 1998.

[97] G. Hasegawa, M. Murata, and H. Miyahara, "Fairness and Stability of Congestion Control Mechanisms of TCP," *IEEE INFOCOM*, March 1999.

[98] E. Hashem, "Analysis of Random Drop for Gateway Congestion Control," *Technical Report LCS TR-465*, Lab for Computer Science, MIT, 1989.

[99] J. Heidemann, K. Obraczka, and J. Touch, "Modeling the Performance of HTTP Over Several Transport Protocols," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, October 1997.

[100] M. Hemy, U. Hengartner, P. Steenkiste, and T. Gross, "MPEG System Streams in Best-Effort Networks," *PacketVideo '99*, April 1999.

[101] U. Hengartner, J. Bolliger, and T. Gross, "TCP Vegas Revisited," *IEEE INFOCOM*, 2000.

[102] J.C. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP," *ACM SIGCOMM*, August 1996.

[103] H.W. Holbrook and D.R. Cheriton, "IP Multicast Channels: EXPRESS Support for Large-Scale Single Source," *ACM SIGCOMM*, September 1999.

[104] C-Y. Hsu, A. Ortega, and A.R. Reibman, "Joint Selection of Source and Channel Rate for VBR Video Transmission Under ATM Policing Constraints," *IEEE*

Journal on Selected Areas in Communications, vol. 15, no. 6, August 1997, 1016-1028.

[105] C. Huang, M. Devetsikiotis, I. Lambadaris, and A.R. Kaye, "Modeling and Simulation of Self-Similar Variable Bit Rate Compressed Video: A Unified Approach," *ACM SIGCOMM*, August 1995.

[106] P. Humblet, A. Bhargava, and M.G. Hluchyj, "Ballot Theorems Applied to the Transient Analysis of nD/D/1 Queues," *IEEE/ACM Transactions on Networking*, vol. 1, no. 1, February 1993.

[107] ISP Planet and Cahners In-Stat Group, "Dial-Up Remains ISPs' Bread and Butter," http://isp-planet.com/research/2001/dialup_butter.html, 2001.

[108] ISP Planet and Telecommunications Research International, "U.S. Residential Internet Market Grows in Second Quarter," http://isp-planet.com/research/2001/us_q2.html, 2001.

[109] S. Jacobs and A. Eleftheriadis, "Real-time Dynamic Rate Shaping and Control for Internet Video Applications," *Workshop on Multimedia Signal Processing*, June 1997, 23-25.

[110] V. Jacobson, "Congestion Avoidance and Control," *ACM SIGCOMM*, 1988.

[111] V. Jacobson, "`pathchar` – A Tool to Infer Characteristics of Internet Paths," ftp://ftp.ee.lbl.gov/pathchar/.

[112] V. Jacobson, `traceroute`, ftp://ftp.ee.lbl.gov/traceroute.tar.gz, 1989.

[113] V. Jacobson and R. Braden, "TCP Extensions for Long-Delay Paths," *IETF RFC 1072*, October 1988.

[114] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," *IETF RFC 1323*, May 1992.

[115] V. Jacobson, R. Braden, and L. Zhang, "TCP Extensions for High-speed Paths," *IETF RFC 1185*, October 1990.

[116] R. Jain, "A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks," *ACM Computer Communication Review*, vol. 19, no. 5, October 1989, 56-71.

[117] R. Jain, "Congestion Control in Computer Networks: Issues and Trends," *IEEE Network Magazine*, May 1990, 24-30.

[118] R. Jain, "Myths about Congestion Management in High-Speed Networks," *IFIP TC6 4th Conference on Information Networks and Data Communication*, March 1992.

[119] H. Kanakia, P.P. Mishra, and A. Reibman, "An Adaptive Congestion Control Scheme for Real-time Packet Video Transport," *IEEE/ACM Transactions on Networking*, vol. 3, no. 6, December 1995. (*also in ACM SIGCOMM, September 1993, 20-31*).

[120] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable protocols," *ACM SIGCOMM*, 1987.

[121] S. Keshav, "A Control-Theoretic Approach to Flow Control," *ACM SIGCOMM*, 1991.

[122] S. Keshav, "Congestion Control in Computer Networks," *Ph.D. Thesis*, University of California, Berkeley, September 1991.

[123] S. Keshav, "Packet-pair Flow Control," *ACM/IEEE Transactions on Networking*, 1997.

[124] S. Keshav and S.P. Morgan, "SMART Retransmission: Performance with Overload and Random Loss," *IEEE INFOCOM*, 1997.

[125] T. Kim, S. Lu, and V. Bharghavan, "Loss Proportional Decrease based Congestion Control in the Future Internet," *University of Illinois Technical Report* http://timely.crhc.uiuc.edu/Drafts/tech.lipd.ps.gz, July 1999.

[126] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, "Multicast Routing for Multimedia Communications," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, June 1993.

[127] K. Kontovasilis and N. Mitrou, "Effective Bandwidths for a Class of Non Markovian Fluid Sources," *ACM SIGCOMM*, September 1997.

[128] A. Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, August 1998, 485-498.

[129] C. Labovitz, A. Ahuja, and F. Jahanian, "Experimental Study of Internet Stability and Wide-Area Network Failures," *University of Michigan Technical Report CSE-TR-382-98*, 1997.

[130] C. Labovitz, G.R. Malan, and F. Jahanian, "Internet Routing Instability," *ACM SIGCOMM*, 1997.

[131] C. Labovitz, G.R. Malan, and F. Jahanian, “Origins of Internet Routing Instability," *IEEE INFOCOM*, 1999.

[132] K. Lai and M. Baker, “Measuring Bandwidth,” *IEEE INFOCOM*, March 1999.

[133] K. Lai and M. Baker, “Measuring Link Bandwidths Using a Deterministic Model of Packet Delay,” *ACM SIGCOMM*, August 2000.

[134] T.V. Lakshman and U. Madhow, “Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss,” *IEEE/ACM Transactions on Networking*, vol. 5, no. 3, June 1997.

[135] T.V. Lakshman, U. Madhow, and B. Suter, “Window-based Error Recovery and Flow Control with a Slow Acknowledgement Channel: a Study of TCP/IP Performance,” *IEEE INFOCOM*, April 1997.

[136] T.V. Lakshman, A. Ortega, and A.R. Reibman, “VBR Video: Tradeoffs and Potentials,” *Proceedings of the IEEE*, vol. 86, no. 5, May 1998, 952-973.

[137] M. Lambert, “On Testing the NETBLT Protocol over Divers Networks,” *IETF RFC 1030*, November 1987.

[138] R. Landry and I. Stavrakakis, “Study of Delay Jitter With and Without Peak Rate Enforcement,” *IEEE/ACM Transactions on Networking*, vol. 5, no. 4, August 1997.

[139] K. Lee, “Performance Bounds in Communication Networks with Variable-Rate Links,” *ACM SIGCOMM*, 1995.

[140] K-W. Lee, T. Kim, V. Bharghavan, “A Comparison of End-to-End Congestion Control Algorithms: The Case of AIMD and AIPD,” *University of Illinois Technical Report* http://timely. crhc.uiuc.edu/~kwlee/psfiles/infocom2001.ps.gz, 2000.

[141] K-W. Lee, R. Puri, T. Kim, K. Ramchandran, V. Bharghavan, “An Integrated Source Coding and Congestion Control Framework for Video Streaming in the Internet,” *IEEE INFOCOM*, March 2000.

[142] T-H. Lee and K-C. Lai, “Characterization of Delay-Sensitive Traffic,” *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, August 1998.

[143] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson, “On the Self-Similar Nature of Ethernet Traffic,” *ACM/IEEE Transactions on Networking*, vol. 2, no. 1, February 1994, 1-15 (*earlier version in ACM SIGCOMM, 1993*).

[144] B.N. Levine, S. Paul, and J.J. Garcia-Luna-Aceves, "Organizing Multicast Receivers Deterministically by Packet-loss Correlation," *ACM Multimedia*, September 1998.

[145] Q. Li and D. Mills, "Investigating the Scaling Behavior, Crossover and Anti-Persistence of the Internet Packet Delay Dynamics," *IEEE GLOBECOM*, 1999.

[146] Q. Li and D. Mills, "Jitter-Based Delay-Boundary Prediction of Wide-Area Networks," *ACM/IEEE Transactions on Networking*, vol. 9, no. 5, October 2001.

[147] Q. Li and D. Mills, "On the Long-Range Dependence of Packet Round-Trip Delays in Internet," *IEEE ICC*, vol. 2, 1998, 1185-1191.

[148] Q. Li and D. Mills, "The Delay Characterization of Wide-Area Networks," *Under submission*, January 26, 2000.

[149] S-Q. Li and C-L. Hwang, "Queue Response to Input Correlation Functions: Continuous Spectral Analysis," *IEEE/ACM Transactions on Networking*, vol. 1, no. 6, December 1993.

[150] S-Q. Li and J.D. Pruneski, "The Linearity of Low Frequency Traffic Flow: An Intrinsic I/O Property in Queueing Systems," *IEEE/ACM Transactions on Networking*, vol. 5, no. 3, June 1997.

[151] X. Li, M. Ammar, and S. Paul, "Layered video multicast with retransmission (LVMR): Evaluation of Error Recovery Schemes," *IEEE INFOCOM*, 1998.

[152] X. Li, S. Paul, P. Pancha, and M. Ammar, "Layered Video Multicast with Retransmission (LVMR): Evaluation of Error Recovery Schemes," *IEEE Workshop on Network and Operating System Support for Digital Audio and Video* (*NOSSDAV*), May 1997.

[153] D. Lin and H.T. Kung, "TCP Fast Recovery Strategies: Analysis and Improvements," *IEEE INFOCOM*, March 1998.

[154] D. Lin and R. Morris, "Dynamics of Random Early Detection," *ACM SIGCOMM*, 1997.

[155] Y-D.J. Lin, T-C. Tsai, S-C. Huang, and M. Gerla, "HAP: A New Model for Packet Arrivals," *ACM SIGCOMM*, 1993.

[156] D. Loguinov and H. Radha, "End-to-End Internet Video Traffic Dynamics: Statistics Study and Analysis," *IEEE INFOCOM*, July 2002.

[157] D. Loguinov and H. Radha, “Increase-Decrease Congestion Control for Real-time Streaming: Scalability,” *IEEE INFOCOM*, July 2002.

[158] D. Loguinov and H. Radha, "Large-Scale Experimental Study of Internet Performance Using Video Traffic," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 32, no. 1, January 2002.

[159] D. Loguinov and H. Radha, “On Retransmission Schemes for Real-time Streaming in the Internet,” *IEEE INFOCOM*, 2001.

[160] D. Loguinov and H. Radha, "Retransmission Schemes for Streaming Internet Multimedia: Evaluation Model and Performance Analysis," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 32, no. 2, April 2002.

[161] G.R. Malan and F. Jahanian, “An Extensible Probe Architecture for Network Protocol Performance Measurement,” *ACM SIGCOMM*, 1998.

[162] A. Mankin and K. Ramakrishnan, “Gateway Congestion Control Survey,” *RFC 1254*, August 1991.

[163] R. Marasli, P.D. Amer, and P.T. Conrad, “Retransmission-based Partially Reliable Transport Service: An Analytic Model,” *IEEE INFOCOM*, 1996.

[164] M. Mathis and J. Mahdavi, “Forward Acknowledgement: Refining TCP Congestion Control,” *ACM SIGCOMM*, August 1996.

[165] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP Selective Acknowledgment Options,” *IETF RFC 2018*, October 1996.

[166] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, “Macroscopic Behavior of the TCP Congestion Avoidance Algorithm,” *ACM Computer Communication Review*, vol. 27, no. 3, July 1997, 67-82.

[167] M. May, J. Bolot, C. Diot, and B. Lyles, “Reasons not to Deploy RED,” *IFIP/IEEE International Workshop on Quality of Service (IWQoS)*, 1999.

[168] S. McCanne and V. Jacobson, “Receiver-driven Layered Multicast,” *ACM SIGCOMM*, August 1996.

[169] S. McCanne and V. Jacobson, “`vic`: A Flexible Framework for Packet Video,” *ACM Multimedia*, November 1995.

[170] S. McCanne, M. Vetterli, and V. Jacobson, “Low Complexity Video Coding for Received-Driven Layered Multicast,” *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 6, August 1997.

[171] P. McKenney, “Stochastic Fairness Queueing,” *IEEE INFOCOM*, 1990.

[172] B. Melander, M. Björkman, P. Gunningberg, “A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks,” *IEEE GLOBECOM*, November 2000.

[173] A. Mena and J. Heidemann, “An Empirical Study of Real Audio Traffic,” *IEEE INFOCOM*, March 2000.

[174] Microsoft Media Player. *Microsoft Corporation*, http://www.microsoft.com/windows/mediaplayer/en/default.asp?RLD=58.

[175] P.P. Mishra and H. Kanakia, “A Hop by Hop Rate-Based Congestion Control Scheme,” *ACM SIGCOMM*, August 1992.

[176] A. Misra and T. Ott, “The Window Distribution of Idealized TCP Congestion Avoidance with Variable Packet Loss,” *IEEE INFOCOM*, 1999.

[177] J. Mo, R.J. La, V. Anantharam, and J. Walrand, “Analysis and Comparison of TCP Reno and Vegas,” *IEEE INFOCOM*, March 1999.

[178] J.C. Mogul, “Observing TCP Dynamics in Real Networks,” *ACM SIGCOMM*, 1992.

[179] R. Morris, “TCP Behavior with Many Flows,” *IEEE International Conference on Network Protocols (ICNP)*, October 1997.

[180] D. Mosberger, L.L. Peterson, P.G. Bridges, and S. O’Malley, “Analysis of Techniques to Improve Protocol Processing Latency,” *ACM SIGCOMM*, 1996.

[181] MPEG-4 International Standard, Part 2: Visual. *ISO/IEC FDIS 14496-2*, October 1998.

[182] A. Mukherjee, “On the Dynamics and Significance of Low Frequency Components of Internet Load,” *Internetworking: Research and Experience*, vol. 5, 163-205, 1994.

[183] A. Mukherjee and J.C. Strikwerda, “Analysis of Dynamic Congestion Control Protocols – A Fokker-Planck Approximation,“ *ACM SIGCOMM*, 1991.

[184] J. Nagle, “Congestion Control in IP/TCP Internetworks,” *IETF RFC 896*, January 1984.

[185] J. Nagle, “On Packet Switches With Infinite Storage,” *IETF RFC 970*, December 1985.

[186] T. Nandagopal, K-W. Lee, J.R. Li, V. Bharghavan, "Scalable Service Differentiation Using Purely End-to-End Mechanisms: Features and Limitations," *IFIP/IEEE International Workshop on Quality of Service (IWQoS)*, June 2000.

[187] Napster, http://www.napster.com.

[188] J.A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, 1965, pp. 308-313.

[189] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," *IETF RFC 2474*, December 1998.

[190] A. Orda, R. Rom, and M. Sidi, "Minimum Delay Routing in Stochastic Networks," *IEEE/ACM Transactions on Networking*, vol. 1, no. 2, April 1993.

[191] A. Ortega, "Optimal Bit Allocation under Multiple Rate Constraints," *Data Compression Conference*, April 1996.

[192] A. Ortega, M.W. Garrett, and M. Vetterli, "Toward Joint Optimization of VBR Video Coding and Packet Network Traffic Control," *Packet Video Workshop*, March 1993.

[193] A. Ortega and K. Ramchandran, "Rate Distortion Methods for Image and Video Compression," *IEEE Signal Processing Magazine*, November 1998, 23-50.

[194] T. Ott, J.H.B Kemperman, and M. Mathis, "The Stationary Behavior of Ideal TCP Congestion Avoidance," http://www.psc.edu/_networking/tcp_friendly.html, August 1996.

[195] J. Padhye, V. Firoiu, and D. Towsley, "A Stochastic Model of TCP Reno Congestion Avoidance and Control," *CMPSCI Technical Report 99-02*, 1999.

[196] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *ACM SIGCOMM*, September 1998.

[197] J. Padhye, J. Kurose, and D. Towsley, "A TCP-Friendly Rate Adjustment Protocol for Continuous Media Flows over Best Effort Networks," *ACM SIGMETRICS*, 1999.

[198] J. Padhye, J. Kurose, D. Towsley, and R. Koodli, "A Model Based TCP-Friendly Rate Control Protocol," *IEEE Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 1999.

[199] C. Papadopoulos and G.M. Parulkar, "Retransmission-based Error Control for Continuous Media Applications," *IEEE Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 1996.

[200] A. Parekh and R.G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, June 1993, 344-357.

[201] K. Park, "Warp Control: A Dynamically Stable Congestion Protocol and its Analysis," *ACM SIGCOMM*, 1993.

[202] S. Paul, S.K. Sabnani, J.C. Lin, S. Bhattacharyya, "Reliable Multicast Transport Protocol (RMTP)," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 3, April 1997 (*earlier version in IEEE INFOCOM*, 1996).

[203] V. Paxson, "Automated Packet Trace Analysis of TCP Implementations," *ACM SIGCOMM*, September 1997.

[204] V. Paxson, "Empirically Derived Analytic Models of Wide-Area TCP Connections," *IEEE/ACM Transactions on Networking*, vol. 2, no. 4, August 1994.

[205] V. Paxson, "End-to-End Internet Packet Dynamics," *ACM SIGCOMM*, September 1997.

[206] V. Paxson, "End-to-End Routing Behavior in the Internet," *ACM SIGCOMM*, August 1996.

[207] V. Paxson, "Fast, Approximate Synthesis of Fractional Gaussian Noise for Generating Self-Similar Network Traffic," *ACM Computer Communication Review*, vol. 27, no. 5, October 1997, 5-18.

[208] V. Paxson, "Growth Trends in Wide-Area TCP Connections," *IEEE Network*, vol. 8, no. 4, July/August 1994.

[209] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," *Ph.D. dissertation*, Computer Science Department, University of California at Berkeley, 1997.

[210] V. PAXSON, "On Calibrating Measurements of Packet Transit Times," *ACM SIGMETRICS*, 1998.

[211] V. Paxson, "Towards a Framework for Defining Internet Performance Metrics," *Proceedings of the INET*, 1996.

[212] V. Paxson and M. Allman, “Computing TCP's retransmission timer,” *IETF RFC 2988*, November 2000.

[213] V. Paxson and S. Floyd, “Difficulties in Simulating the Internet,” *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, August 2001.

[214] V. Paxson and S. Floyd, “Wide-Area Traffic: The Failure of Poisson Modeling,” *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, 1994, 226-244 (*earlier version in ACM SIGCOMM, 1994*).

[215] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis, “An Architecture for Large-Scale Internet Measurement,” *IEEE Communications*, August 1998.

[216] S. Pejhan, M. Schwartz, and D. Anastassiou, “Error control using retransmission schemes in multicast transport protocols for real-time media,” *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, June 1996, 413-427.

[217] S.P. Pizzo, “Why Is Broadband So Narrow?” *Forbes ASAP*, September 2001, p. 50.

[218] M. Podolsky, C. Romer and S. McCanne, “Simulation of FEC-based error control for packet audio on the Internet,” *IEEE INFOCOM*, 1998.

[219] J. Postel, “Internet Control Message Protocol,” *IETF RFC 792*, September 1981.

[220] J. Postel, “Internet Protocol,” *IETF RFC 791*, September 1981.

[221] J. Postel, “Transmission Control Protocol – DARPA Internet Program Protocol Specification,” *IETF RFC 793*, September 1981.

[222] J. Postel, “User Datagram Protocol,” *IETF RFC 768*, August 1980.

[223] Quality of Service Configuration Guide. *Cisco IOS 12.0 User's Guide*, Cisco Systems, October 1999.

[224] H. Radha and Y. Chen, “Fine-Granular Scalable Video for Packet Networks,” *IEEE Packet Video*, May 1999.

[225] H. Radha, Y. Chen, K. Parthasarathy, and R. Cohen, “Scalable Internet Video Using MPEG-4,” *Signal Processing: Image Communications Journal*, 1999.

[226] H. Radha *et al*. "The MPEG-4 Fine-Grained Scalable Video Coding Method for Multimedia Streaming Over IP", *IEEE Transactions on Multimedia*, March 2001, vol. 3, no. 1, pp. 53-68.

[227] S. Raman and S. McCanne, "A Model, Analysis, and Protocol Framework for Soft State-based Communication," *ACM SIGCOMM*, September 1999.

[228] K. Ramakrishnan and S. Floyd, "A Proposal to Add Explicit Congestion Notification (ECN) to IP," *IETF RFC 2481*, January 1999.

[229] K. Ramakrishnan and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with Connectionless Network Layer," *ACM SIGCOMM*, August 1988, 303-313.

[230] S. Ramanathan and P.V. Rangan, "Adaptive Feedback Techniques for Synchronized Multimedia Retrieval over Integrated Networks," *IEEE/ACM Transactions on Networking*, vol. 1, no. 2, April 1993.

[231] S. Ratnasamy and S. McCanne, "Inference of Multicast Routing Trees and Bottleneck Bandwidths using End-to-End Measurements," *IEEE INFOCOM*, 1999.

[232] Real Player G2, *Real Networks*, http://www.real.com.

[233] I.S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *Journal of the Society of Industrial and Applied Mathematics*, vol. 8, no. 2, June 1960, 300-304.

[234] A.R. Reibman and B.G. Haskell, "Constrains on Variable Bit-rate Video for ATM Networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 4, December 1992.

[235] R. Rejaie, "An End-to-End Architecture for Quality Adaptive Streaming Applications in the Internet," *Ph.D. Thesis*, Computer Science Department, University of Southern California, December 1999.

[236] R. Rejaie and M. Handley, "Quality Adaptation for Congestion Controlled Video Playback over the Internet," *ACM SIGCOMM*, September 1999.

[237] R. Rejaie, M. Handley, and D. Estrin, "Architectural Considerations for Playback of Quality Adaptive Video over the Internet," *Technical report 98-686*, Computer Science Department, University of Southern California.

[238] R. Rejaie, M. Handley, and D. Estrin, "Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet," *IEEE INFOCOM*, March 2000.

[239] R. Rejaie, M. Handley, and D. Estrin, "RAP: An End-to-End Rate-based Congestion Control Mechanism for Real-time Streams in the Internet," *IEEE INFOCOM*, March 1999.

[240] S.R. Resnick, “Heavy Tail Modeling and Teletraffic Data,” *Annals of Statistics*, vol. 25, no. 5, 1997, 1805-1869.

[241] I. Rhee, “Error Control Techniques for Interactive Low Bitrate Video Transmission over the Internet,” *ACM SIGCOMM*, September 1998.

[242] V.J. Ribeiro, R.H. Reidi, M.S. Crouse, and R. Baraniuk, “Simulation of nonGaussian Long-Range-Dependent Traffic using Wavelets,” *ACM SIGMETRICS*, 1999.

[243] A. Romanow and S. Floyd, “Dynamics of TCP Traffic over ATM Networks,” *IEEE Journal on Selected Areas in Communications (JSAC)*, May 1995.

[244] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, “TCP Congestion Control with a Misbehaving Receiver,” *ACM Computer Communication Review*, vol. 29, no. 5, October 1999, 71-78.

[245] S. Savage, A. Collins, and E. Hoffman, “The End-to-End Effects of Internet Path Selection,” *ACM SIGCOMM*, September 1999.

[246] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications,” *IETF RFC 1889*, December 1996.

[247] S. Servetto and K. Nahrstedt, “Broadcast Quality Video over IP,” *IEEE Transactions on Multimedia*, vol. 3, no. 1, March 2001.

[248] A. Shaikh, J. Rexford, and K.G. Shin, “Load-Sensitive Routing of Long-Lived IP Flows,” *ACM SIGCOMM*, September 1999.

[249] J.M. Shapiro, “Embedded Image Coding Using Zerotrees of Wavelet Coefficients,” *IEEE Transactions on Signal Processing*, vol. 41, no. 12, December 1993, 3445-3462.

[250] H-D. Sheng and S-Q. Li, “Spectral Analysis of Packet Loss Rate at a Statistical Multiplexer for Multimedia Services,” *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, February 1994.

[251] S. Shenker, “A Theoretical Analysis of Feedback Flow Control,” *ACM SIGCOMM*, September 1990, 156-165.

[252] S. Shenker, “Fundamental Design Issues for the Future Internet,” *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 13, no. 7, 1995, 1176-1188.

[253] A. Shionozaki and M. Tokoro, “Control Handling in Real-Time Communication Protocols,” *ACM SIGCOMM*, 1993.

[254] W. Simpson, editor, "The Point-to-Point Protocol (PPP)," *IETF RFC 1661*, July 1994.

[255] D. Sisalem and H. Schulzrinne, "The Loss-delay Based Adjustment Algorithm: A TCP-friendly adaptation scheme," *IEEE Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 1998.

[256] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti, "The PPP Multilink Protocol (MP)," *IETF RFC 1990*, August 1996.

[257] V. Srinivasan, S. Suri, and G. Verghese, "Packet Classification using Tuple Space Search," *ACM SIGCOMM*, September 1999.

[258] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," *IETF RFC 2001*, January 1997.

[259] I. Stoica and H. Zhang, "Providing Guaranteed Services without Per Flow Management," *ACM SIGCOMM*, September 1999.

[260] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High-speed Networks," *ACM SIGCOMM*, September 1998, 118-130.

[261] G. Sullivan and T. Wiegand, "Rate Distortion Optimization for Video Compression," *IEEE Signal Processing Magazine*, November 1998, 74-89.

[262] R. Talluri, "Error-Resilient Video Coding in the ISO MPEG-4 Standard," *IEEE Communications Magazine*, June 1998, 112-119.

[263] W. Tan and A. Zakhor, "Real-Time Internet Video Using Error Resilient Scalable Compression and TCP-Friendly Transport Protocol," *IEEE Transactions on Multimedia*, vol. 1, no. 2, June 1999 (*earlier paper in IEEE ICIP, 1998*).

[264] T. Turletti and G. Huitema, "Videoconferencing on the Internet," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, June 1996.

[265] UUnet Latency Statistics. http://uunet.com/network/latency.

[266] B. Vandalore, W-C. Feng, R. Jain, and S. Fahmy, "A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia," *Ohio State University Techincal Report*, OSU-CISRC-5/99-TR14, April 1999.

[267] N. Venkitaraman, T-E. Kim, and K-W. Lee, "Design and Evaluation of Congestion Control Algorithms in the Future Internet," *ACM SIGMETRICS*, 1999.

[268] S. Vutukury and J.J. Garcia-Luna-Aceves, "A Simple Approximation to Minimum-Delay Routing," *ACM SIGCOMM*, September 1999.

[269] M. Wada, "Selective recovery of video packet loss using error concealment," *IEEE Journal on Selected Areas in Communications*, vol. 7, June 1989, 807-814.

[270] R. Wade, M. Kara, and P.M. Dew, "Study of a Transport Protocol Employing Bottleneck Probing and Token Bucket Flow Control," *5th IEEE Symposium on Computers and Communications*, July 2000.

[271] Y. Wang and B. Sengupta, "Performance Analysis of a Feedback Congestion Control Policy Under Non-Negligible Propagation Delay," *ACM SIGCOMM*, 1991.

[272] Y. Wang and Q-F. Zhu, "Error Control and Concealment for Video Communication: Review," *Proceedings of the IEEE*, vol. 86, no. 5, May 1998.

[273] Z. Wang and J. Crowcroft, "Analysis of Burstiness and Jitter in Real-Time Communications," *ACM SIGCOMM*, 1993.

[274] J. Widmer, "Equation-based Congestion Control," *Diploma Thesis*, Department of Mathematics and Computer Science, University of Mannheim, February 2000.

[275] C.L. Williamson, "Optimizing File Transfer Response Time Using the Loss-Load Curve Congestion Control Mechanism," *ACM SIGCOMM*, 1993.

[276] C.L. Williamson and D.R. Cheriton, "Loss-Load Curves: Support for Rate-Based Congestion Control in High-Speed Datagram Networks," *ACM SIGCOMM*, September 1991, 17-28.

[277] W. Willinger and V. Paxson, "Where Mathematics Meets the Internet," *Notices of the American Mathematical Society*, vol. 45, no. 8, August 1998, 961-970.

[278] W. Willinger, V. Paxson, and M.S. Taqqu, "Self-Similarity and Heavy Tails: Structural Modeling of Network Traffic," Appears in "A Practical Guide to Heavy Tails: Statistical Techniques and Applications," book by Adler, R., *et al.*, Birkhauser, Boston, 1998, 27-53.

[279] W. Willinger, M.S. Taqqu, R. Sherman, and D.V. Wilson, "Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," *ACM SIGCOMM*, 1995.

[280] D.E. Wrege, E.W. Knightly, H. Zhang, and J. Liebeherr, "Deterministic Delay Bounds for VBR Video in Packet-Switching Networks: Fundamental Limits and

Practical Trade-offs," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, June 1996.

[281] L. Wu, R. Sharma, and B. Smith, "Thin Streams: An Architecture for Multicasting Layered Video," *Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, May 1997.

[282] M. Yajnik, J. Kurose, and D. Towsley, "Packet Loss Correlation in the MBone Multicast Network," *IEEE GLOBECOM*, November 1996.

[283] M. Yajnik, S. Moon, J. Kurose, and D. Townsley, "Measurement and Modelling of the Temporal Dependence in Packet Loss," *IEEE INFOCOM*, 1999.

[284] C-Q. Yang and A. Reddy, "A Taxonomy for Congestion Control Algorithms in Packet Switching Networks," *IEEE Network Magazine*, vol. 9, no. 5, July/August 1995.

[285] Y.R. Yang, M.S. Kim, and S.S. Lam, "Transient Behaviors of TCP-friendly Congestion Control Protocols," *IEEE INFOCOM*, April 2001.

[286] Y.R. Yang and S.S. Lam, "General AIMD Congestion Control," *University of Texas at Austin Technical Report* ftp://ftp.cs.utexas.edu/pub/lam/gaimd.ps.gz, May 2000.

[287] D. Yates, J. Kurose, D. Towsley, and M.G. Hluchyj, "On Per-Session End-to-End delay distributions and the call admission problem for real-time applications with QoS requirements," *ACM SIGCOMM*, 1993.

[288] D.K.Y. Yau and S. Lam, "Adaptive Rate-Controlled Scheduling for Multimedia Applications," *IEEE/ACM Transactions on Networking*, vol. 5, no. 4, August 1997.

[289] E.W. Zegura, K.L. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," *IEEE INFOCOM*, 1996.

[290] L. Zhang, "A New Architecture for Packet Switching Network Protocols," *MIT/LCS/TR-455*, Lab for Computer Science, MIT, August 1989.

[291] L. Zhang, S. Shenker, and D.D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The effect of two-way traffic," *ACM SIGCOMM*, September 1991.

[292] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the Constancy of Internet Path Properties," *ACM SIGCOMM Internet Measurement Workshop (IMW)*, November 2001.