MODEL-BASED APPROACH TO TCP-FRIENDLY CONGESTION CONTROL

A Dissertation Presented

by

JITENDRA D. PADHYE

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2000

Department of Computer Science

MODEL-BASED APPROACH TO TCP-FRIENDLY CONGESTION CONTROL

A Dissertation Presented

by

JITENDRA D. PADHYE

Approved as to style and content by:

James F. Kurose, Co-chair

Don Towsley, Co-chair

Prashant Shenoy, Member

Weibo Gong, Member

Sally Floyd, Member

James F. Kurose, Department Chair
Department of Computer Science

To my parents and my sisters

ACKNOWLEDGMENTS

It is with deep respect and appreciation that I acknowledge my advisors, Professors Jim Kurose and Don Towsley for their guidance in my professional growth. They have not only taught me how to do quality research but also how to communicate it effectively to others. Moreover, they have been caring mentors, ready to offer advice and encouragement whenever I have needed it. Without their broad vision and continual encouragement, this dissertation would not have been possible.

I have had the honor of working with several outstanding researchers during the course of this research. Thanks to Victor Firoiu for collaborating with me on the TCP modeling work. His keen mathematical insights were invaluable to our work. I would like to thank Mark Handley and Sally Floyd for their guidance and collaboration during my visit to ACIRI. Thanks to Joerg Widmer for providing some of the data used in Chapter 5.

Thanks to Professors Prashant Shenoy and Weibo Gong for their guidance as members of my dissertation committee.

I would like to thank all the colleagues and friends who provided me with access to their machines to conduct the experiments necessary for this research.

Many thanks are due to my fellow graduate students. They have contributed greatly to my professional as well as personal development. I sincerely thank H. Shrikumar, Supratik Bhattacharya, Michael Bradshaw, Tian Bu, Timur Friedman, Sneha Kasera, Vishal Misra, Sue Moon, Dan Rubenstein, Sambit Sahu, Shubhabrata Sen and Maya Yajnik for their support and for many stimulating discussions.

Gary Wallace and the CSCF staff provided wonderful system administration support during the course of this research. I can not overstate the importance of their help.

I would like to thank Betty Hardy and Sharon Mallory for their help in navigating through the graduate school.

I am indebted to Professor Larry Dowdy, Evgenia Smirni and P. Krishnan for their help and support during my stay at the Vanderbilt University.

It would have been difficult to last through the years in graduate school without the love and company of my numerous friends. They have all been there to celebrate the good times, and help me through the bad ones. I will not attempt to list their names, for the fear that any such list will be incomplete. I express my thanks to them all.

My deepest gratitude goes to my parents and my sisters. Their love, support and care have always been the greatest inspiration in my life.

Finally, I would like to thank Tanika for bringing so much joy and beauty in my life. She is clearly the best discovery of my graduate school years.

ABSTRACT

MODEL-BASED APPROACH TO TCP-FRIENDLY CONGESTION CONTROL

SEPTEMBER 2000

JITENDRA D. PADHYE

B.E., VICTORIA JUBILEE TECHNICAL INSTITUTE

M.S., VANDERBILT UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor James F. Kurose and Professor Don Towsley

Continuous media (CM) applications such as streaming audio and video are being rapidly deployed throughout the Internet. These applications must be able to co-exist with each other and with traditional TCP-based applications.

One requirement for such co-existence is the implementation of congestion control mechanism by CM applications. TCP-based applications reduce their sending rate in face of network congestion. Many CM applications run on top of UDP, as TCP's error control mechanisms are ill suited for CM applications. UDP-based applications must provide their own congestion control.

In this dissertation, we work towards development of a congestion control protocol for CM applications that will facilitate co-existence of CM applications with each other, and with TCP flows. We address this problem in two steps.

First, we develop an analytical model of TCP. Such a model is essential to understand TCP's congestion control behavior under a variety of network conditions. Our model provides a closed form equation that can be used to calculate the send rate of a TCP connection, given the packet loss rate, the round trip time and the base timeout value. We verify the accuracy of our model against measurements taken in the Internet.

The next step is to develop a congestion control protocol that is suitable for the CM applications and also shares the bandwidth fairly with TCP connections that travel over the same network path. We develop a protocol that uses our TCP model as the control function of a feedback loop. The protocol design is deliberately kept simple to allow us to evaluate the feasibility of the model-based approach to congestion control. The simple protocol is evaluated using simulations and Internet experiments. Our results show that the protocol performs well under a variety of network conditions.

We then refine the simple protocol, trading off design simplicity to achieve better performance. We introduce sophisticated mechanisms to track packet loss rate, control rate increase and handle startup behavior. We evaluate the protocol using simulations and Internet experiments. Our results show that the more complex protocol provides a smooth sending rate and is TCP-friendly under a wide variety of networking conditions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In the last few years the Internet has seen explosive growth. A wide variety of applications such as the Web, email, streaming audio and video have been deployed, to cater to the growing ranks and demands of Internet users. One of the most remarkable thing about the current Internet, and perhaps the reason for its demonstrated scalability is that it provides a simple, best-effort model of service. This means that the routers do not give priority to traffic belonging to any application or group of applications. The network does not provide any guarantees regarding the delivery of packets, including whether or not they will be delivered at all. Most routers in the current Internet employ FIFO queue management coupled with a drop-tail buffer management policy. The packets received are sent out in the order in which they are received, and any packet that arrives at a full buffer is dropped. The routers make no attempt to notify the sender either about incipient congestion or about the dropped packets. We note that solutions such as RED [14], ECN [54], IntServ [7] and DiffServ [8] have been proposed to address some of these issues. None of these have yet seen widespread deployment.

The onus of maintaining stability and "fairness" (we shall define this term in more detail in later part of the thesis) in the Internet is clearly on the applications, and not on the network itself. That is, the applications must be able to take suitable actions to maintain stability and fairness without expecting too much feedback from the network.

The most important action that must be taken by applications in any best-effort networks is that they must reduce their sending rate when faced with congestion. Otherwise, congestion collapse may result [12]. The way in which sending rate is reduced when faced with congestion, and increased when congestion is no longer present can collectively be called the *congestion control* aspect of the protocol.

The majority of the traffic on the Internet is controlled by the TCP protocol. TCP implements several congestion control mechanisms that are based on the guidelines first proposed in [6]. TCP has been instrumental in maintaining the stability of the Internet, leading to its explosive growth in the last few years.

The principle driving force behind this growth has been the development of the Web, with its inherently multi-media nature. With increasing availability of high bandwidth access lines, continuous media (CM) applications such as streaming audio and video are being rapidly deployed throughout the Internet. These applications run on top of UDP [51] but do not implement any form of congestion control [36]. And yet, these applications share the Internet bandwidth with traditional TCP-based applications such as FTP and SMTP.

Since TCP applications implement congestion control, and CM applications do not, their co-existence has lead to fears that the widespread use of CM applications will harm the performance of traditional TCP applications and ultimately lead to the congestion collapse of the Internet.

This problem can be addressed in two steps. First, the behavior of TCP under a wide variety of networking conditions must be well understood. This requires that we create an accurate mathematical model of TCP. Second, congestion control protocols must be designed for CM applications that take into account their special needs, and can yet co-exist with TCP applications.

The aim of the research presented in this dissertation is to accurately model the performance of TCP, and to design congestion control protocol for CM applications that permit the co-existence of TCP and CM applications.

In the next section, we discuss these two problems in more details.

1.1 Problem definition and motivation

1.1.1 Modeling of TCP congestion control

A large portion of the Internet traffic is controlled by the TCP protocol [70]. Thus, understanding the behavior of TCP is critical to any Internet engineering or research efforts. One way to better understand the TCP behavior is to create a mathematical model of TCP. Such a model can be useful in many ways. TCP is a complex protocol and provides many services, including packet sequencing, buffering, reliable delivery, fair sharing of bandwidth among many data flows, as well as congestion control. A mathematical model of TCP allows us to study specific aspects of the protocol in isolation from others. A TCP model is useful for a fast "what-if" analysis of various networking scenarios. Without a tractable mathematical model, such a study will have to depend on time-consuming simulations. Such an analyses are routinely used for network provisioning studies, to study the impact of new technologies such as active queue management that are being deployed in the Internet, to study interactions between TCP and new transport layer protocols, and to ensure that new transport layer protocols are "friendly" to existing TCP applications.

We now briefly review the TCP protocol. More details can be found in Chapter 2, as well as in [21, 33, 52, 64–66]. Many versions of TCP are currently deployed in the Internet. There is evidence [48, 49] to suggest that most dominant version at the time of writing this thesis is the "Reno" [64] version.

TCP is a transport layer protocol that provides two main services: reliable delivery over the best-effort IP [1] layer, and congestion control. In TCP-Reno, these two services are tightly coupled. However, as mentioned earlier, our focus is on the congestion control aspect. The congestion control algorithms implemented in TCP have two goals. The first is to ensure that the bandwidth is fairly shared, and the second is to ensure that the network remains "stable" and congestion collapse is avoided [12]. To achieve these goals, TCP does not get much help from the network layer. Most Internet routers do not send back explicit congestion notifications to the sender, but instead simply drop packets when there are too many packets to fit in the router buffer. The TCP sender finds out about lost packets based on acknowledgments received from the receiver. Packet losses are considered to be an indication of congestion, and trigger TCP's congestion control mechanisms.

A TCP sender sends packets that carry unique packet numbers. The receiver responds to each packet by sending a cumulative acknowledgment [52]. The sequence number of the acknowledgment indicates that all packets upto, but not including, that number have been received. In absence of packet losses, TCP increases its sending rate rate slowly (i.e. by one packet per round trip time, which is an *additive* increase). When an incoming ACK indicates a lost packet, the sender assumes that the packet loss was caused by congestion in the network, and reduces its sending rate by 50%, (i.e. *multiplicative* decrease). It has been shown in [6] that this additive-increase, multiplicative-decrease mechanism results in "fair" sharing of bandwidth among multiple flows under certain ideal conditions. If a large number of packets are dropped by the network at once, the TCP sender may not receive any acknowledgments at all, and is forced to time out. In this mode a TCP sender repeatedly attempts to send the first unacknowledged packet, while waiting for exponentially increasing time periods between successive sends. This timeout phase ends when the receiver successfully acknowledges the packet. A more detailed description is provided in Chapter 2.

Several TCP models have been previously proposed in the literature [32, 34, 41]. It has been shown in [34] that these models do not always match observed data well.

The main reason for this is that these models do not account for the impact of the timeout periods. As a result, these models tends to significantly overestimate send rate for loss rates greater than 5%. In addition, these models do not account for other limiting factors such as the receiver's declared window size.

In this thesis, we develop a simple analytical model of TCP congestion control that is valid for a wide variety of network conditions, and verify the model via analysis and direct measurements in the Internet.

1.1.2 Model based TCP friendly congestion control

As discussed earlier, with widespread deployment of CM applications, it is becoming increasingly important to ensure that these applications can coexist with each other and with current TCP based applications. TCP-based applications reduce their sending rate in face of network congestion. Many current CM applications use UDP as their transport layer protocol, and do not change their sending rate regardless of the congestion state of the network. This can cause severe degradation in performance of TCP-based applications and may lead to congestion collapse of the Internet.

Two different approaches can be taken to avoid performance degradation in the Internet due to widespread deployment of CM applications. One is to keep CM applications separate from each other and from the TCP applications; the second approach is to incorporate some form of congestion control in CM applications and make them "TCP-friendly".

The first approach [7,8,28] advocates models of Internet service that provide multiple service classes. End-to-end reservations may be used to provide delay and loss guarantees to the so-called "premium service" classes. The CM applications would likely fall in the premium service class, while the TCP applications might occupy their own best-effort service class. If this dichotomy is achieved completely, the CM applications running in the premium service classes may not need to incorporate any congestion control mechanisms. However, we believe that even when such premium services become available, many users of CM applications will want to use the best effort service, due to cost considerations. In that case, the CM applications that run in the best-effort class will need to incorporate some form of congestion control.

Thus, for either approach, it is important to design a congestion control protocol for CM applications running over best effort networks. In this thesis, we work towards the development of such a protocol.

Recall that TCP congestion control reduces the sending rate by one half in response to a single congestion indication. This type of congestion control is appropriate for bulk data transfer applications such as FTP, which do not have delay constraints. CM applications play out data in real-time. Reducing the sending rate by 50% in response to a single congestion indication noticeably degrades the user-perceived quality [68]. Abrupt changes in the sending rate have been a significant impediment to the deployment of TCP's end-to-end congestion control by emerging CM applications.

We believe that a viable congestion control mechanism for CM traffic should not respond to every loss event, but only to the recently measured *loss event rate*. Abrupt changes in the sending rate are minimized by using a response function that depends on the loss event rate rather than individual loss events. For traffic that competes in the best-effort Internet with TCP, the appropriate response function is a model of TCP that characterizes the steady-state sending rate of TCP as a function of the round-trip time and steady-state loss event rate. Using a TCP like response function makes the protocol "TCP-friendly" - i.e., the steady state send rate of the protocol will be roughly equal to that of a TCP connection experiencing similar network conditions. This definition of TCP-friendliness was first proposed in [32]. We refine this definition further in later chapters.

Note that a complete CM congestion control protocol will also need to account for the timeliness requirement of CM data delivery, besides minimizing abrupt changes in the send rate and ensuring TCP-friendliness. We will not discuss this aspect of CM congestion control in this thesis. Timeliness requirements are different for different applications and different CM data types. In our opinion, they are best handled at the application layer, by individual applications. We thus focus our attention on designing a TCP-friendly rate control protocol and evaluate its performance using metrics that measure variability in the send rate and compatibility with TCP.

Several TCP-friendly protocols have recently been reported in the literature [19, 42, 56, 63, 68, 71, 72]. However, many of these protocols have significant shortcomings. Many are not usable over a wide range of loss rates. Some respond to individual loss events like TCP, instead of responding to aggregate loss rates. We discuss these shortcomings in more detail in Chapter 4.

Thus the problem we solve is the following: design and evaluate a TCP-friendly rate control protocol that is useful over a wide variety of network conditions.

We design a TCP-friendly rate control protocol that makes use of a TCP model, such as the one proposed in Chapter 2. This model is used in form of an equation that gives the TCP send rate for given the packet loss rate, the round trip time and the base timeout value. The send rate of the protocol is controlled by a feedback loop that uses this equation as the control function. We call such protocols "equation-based" or "model-based" protocols.

To design and evaluate a TCP-friendly protocol that is useful over a wide range of network conditions, we must consider several issues. First, we must use a TCP model that is valid over a wide range of loss rates. We develop such a model in Chapter 2. We must design robust techniques for loss rate estimation. To evaluate the protocol, we must design suitable metrics that measure the performance of the protocol over wide range of time scales. Finally, we must also test and validate the protocol by measurements over the Internet.

We work on the problem in two steps. We first develop and evaluate a simple protocol (Chapter 4). We keep the design of the protocol as simple as possible. This allows us to evaluate the feasibility and robustness of the equation-based congestion control approach. Our results show that the protocol performs well under a variety of network conditions. These results establish the feasibility of the model-based approach to congestion control.

In Chapter 5, we refine the simple protocol, trading off simplicity to achieve better performance. We introduce sophisticated mechanisms to track packet loss rate, control rate increase and handle slowstart. As before, we evaluate the protocol using simulations and Internet experiments. Simulation studies and experimental evaluations show that this more complex version of the protocol provides a smooth sending rate and is TCP-friendly under a wide variety of networking conditions, and over a wide range of timescales.

1.2 Contributions of the dissertation

The contributions of this dissertation can be divided into two broad areas:

- Modeling and characterization of TCP congestion control and avoidance mechanisms

- Design and development of equation-based TCP-friendly congestion control.

In the area of modeling of TCP congestion control, the principle contributions can be summarized as follows: development of an accurate stochastic model of TCP congestion control, validation of this model by Internet measurements, development of a simple formula to predict TCP throughput based on this model, and development of an even more accurate Markovian model of TCP congestion control.

In the area of design and development of TCP-friendly congestion control, the principle contributions can be summarized as follows: design of a simple equation-based TCP-friendly congestion control protocol, implementation and Internet studies of this protocol, development of a improved protocol to overcome the shortcomings of the simpler protocol, definition of various metrics to measure performance of TCP-friendly congestion control protocols, and simulation studies of the two protocols based on these metrics.

We now briefly discuss each of these contributions.

1.2.1 Modeling of TCP congestion control

- **An accurate stochastic model of TCP congestion control:** We present a simple and yet accurate stochastic model of TCP congestion control mechanisms. Our model has several advantages over previous work [32, 34, 41] in this area. Our work the first to account for the impact of retransmission timeouts. It also provides a careful and integrated treatment of impact of delayed acknowledgments, limitations on the send rate imposed by receiver's window size, and explicit modeling of TCP's fast retransmit mechanisms. While previously proposed models are valid only for loss rates below 5% [34], our model is valid over a almost the entire range of packet loss rates. Our own measurements, as well as those of independent studies [3, 5, 16, 53] show that our model provides a good fit to observed data under a wide variety of network conditions.

- **Validation of the model by Internet measurements:** To validate our model, we gathered data by tracing the packet-level dynamics of over several TCP connections over the Internet. In all, we studied over 1400 TCP connections lasting 100 seconds each, and 25 TCP connections lasting 1 hour each. For our study, we used over 20 computers located across United States and Europe, running a variety of hardware and operating systems. Our measurements conclusively demonstrate that impact of retransmission timeouts must be taken into account while modeling TCP, and that our model provides a far better fit to observed data than previously proposed models. The data we gathered, and the software we developed to analyze it, has been used by other researchers [37, 55] in their studies.

- **A simple formula that predicts TCP send rate:** Our stochastic analysis yields a simple formula that can be used to predict the send rate of a TCP-Reno connection over a wide range of network conditions. The formula for the send rate can also be used in a feedback loop to design TCP-friendly rate control protocols, as illustrated in Chapters 4 and 5. The formula has also been used by other researchers to solve large queuing networks [11, 38].

- **A detailed Markovian model of TCP congestion control:** Our analysis of TCP behavior makes several simplifying assumptions in order to arrive at a closed-form solution. These assumptions are amply justified by the closeness between the model predictions and actual TCP behavior. However, to validate the model further, we have developed a more detailed Markovian model. This model does not seem to have a simple closed-form solution. However, the numerical solutions of this detailed model agree with the predictions of the simpler, closed-form model in Chapter 2, and the actual measurements all appear to be in excellent agreement, thereby providing additional validation for the approximate model. This detailed model has recently been extended [61] to investigate behavior of TCP under DiffServ [8] environments.

1.2.2 Model-based TCP-friendly congestion control

- **Design of a simple, model-based TCP-friendly congestion control protocol:** We design and implement a simple unicast congestion control protocol that is based on the TCP model proposed in Chapter 2. This is the first model-based TCP-friendly rate control protocol that uses the a TCP-model that accounts for timeouts, and hence promises to be TCP-friendly over a wide range of loss rates. Due to its simplicity, this protocol serves as a baseline further research in the area of TCP-friendly congestion control.

- **Implementation of a prototype version of the simple protocol:** We implement a simple prototype version of our protocol, and conduct more than 300 experiments over the Internet, using computers located in the United States and Europe. The results of these experiments clearly indicate the feasibility of deploying TCP-friendly congestion control based on TCP model proposed in Chapter 2.

- **Design of an improved TCP-friendly congestion control protocol:** We refine the simple protocol, trading off simplicity of design to achieve better performance. We develop new mechanisms to estimate the packet loss rate, to adjust the send rate based on observed loss rate and round trip time and to ensure stability during slow start. We identify the problems posed by data senders that go idle for long periods of time [17] and propose guidelines for both TCP as well as TCP-friendly protocols to deal with these problems.

- **Definition of performance metrics for evaluating TCP-friendly congestion control:** To compare the improved protocol in Chapter 5, with TCP as well as the simpler protocol in Chapter 4, we define several performance metrics for studying the performance of TCP-friendly congestion control protocols. The metrics are used to evaluate the protocol for compatibility with TCP and for compatibility with other flows that use the same congestion control mechanisms. We also define a metric to compare the variability in the send rates achieved by various protocols. Most importantly, we define these metrics in a way that allows for performance comparison at various timescales. None of the previous work [19, 42, 45, 56, 63, 68, 71, 72] in this area had considered the performance of a TCP-friendly congestion control protocol at multiple timescales.

- **Simulation and experimental studies of the improved protocol:** We use simulations to study the performance of the improved protocol and compare it with that of the simpler protocol. We find that the improved protocol is more "TCP-friendly" across a broad range of timescales. We also find that the send rate of the improved protocol is less "variable" than both the simple protocol and TCP itself, across a wide range of timescales. Other researchers [73] have implemented a prototype version of the new protocol. Experiments carried out using this prototype version confirm the results we obtain via simulations.

1.3 Structure of the dissertation

The structure of the thesis is as follows. In Chapter 2 we derive a simple model for TCP Reno. We show that the model provides a good fit to data gathered from Internet measurements. In Chapter 3, we provide a more precise stochastic model of TCP. This model does not provide us with a closed-form formula like the simpler model described in Chapter 2. However, numerical solutions of this more accurate model match well with the simpler model from Chapter 2, providing further justification for its usability. In Chapter 4, we describe a simple TCP-friendly rate control protocol. We evaluate the protocol using simulations and Internet experiments. These evaluations prompt us to develop a more refined version of the protocol, which we present in Chapter 5. This latter version of the protocol is also evaluated using simulations and Internet experiments. Finally, we present our conclusions and directions for future work in Chapter 6.

CHAPTER 2

A SIMPLE MODEL OF TCP RENO

2.1 Introduction

A significant amount of today's Internet traffic, including WWW (HTTP), file transfer (FTP), email (SMTP), and remote access (Telnet) traffic, is carried by the TCP transport protocol [70]. TCP together with UDP form the very core of today's Internet transport layer. Traditionally, simulation and implementation/measurement have been the tools of choice for examining the performance of various aspects of TCP. Recently, however, several efforts have been directed at analytically characterizing the send rate of TCP's congestion control mechanism, as a function of packet loss and round trip delay [32, 34, 41]. As mentioned in Chapter 1, a simple quantitative characterization of TCP send rate under given operating conditions offers the possibility of defining a "fair share" or "TCP-friendly" [32] send rate for a non-TCP flow that interacts with a TCP connection.

In this chapter we develop a simple analytic characterization of the steady state send rate of a bulk transfer TCP flow (i.e., a flow with a large amount of data to send, such as FTP transfers) as a function of loss rate and round trip time. Unlike the recent work of [32,34,41], our model captures not only the behavior of TCP's fast retransmit mechanism, but also the effect of TCP's timeout mechanism on send rate. The measurements we present in Section 2.3 indicate that this latter behavior is important from a modeling perspective, as we observe more timeout events than fast retransmit events in almost all of our TCP traces. Another important difference between our present work and previous work is the ability of our model to accurately model TCP send rate over a significantly wider range of loss rates than before; measurements presented in [34] as well the measurements presented in this chapter, indicate that this too is important. We also explicitly model the effects of small receiver-side windows. By comparing our model's predictions with a number of TCP measurements made between various Internet hosts, we demonstrate that our model is able to more accurately predict TCP send rate than the previous efforts, and is able to do so over a wider range of loss rates.

The remainder of the chapter is organized as follows. In Section 2.2 we describe our model of TCP congestion control in detail and derive a new analytic characterization of TCP send rate as a function of loss rate and average round trip time. In Section 2.3 we compare the predictions of our model with a set of measured TCP flows over the Internet, having as their endpoints sites in both United States and Europe. Section 2.4 discusses the assumptions underlying the model and a number of related issues in more detail. Section 2.5 concludes the chapter.

2.2 The model

In this section we develop a stochastic model of TCP congestion control that yields a relatively simple analytic expression for the send rate of a saturated TCP sender, i.e., a flow with an unlimited amount of data to send, as a function of loss rate and estimated average round trip time (RTT).

TCP is a protocol that can exhibit complex behavior, especially when considered in the context of the current Internet, where the traffic conditions themselves can be quite complicated and subtle [50]. In this chapter, we focus our attention on the congestion avoidance behavior of TCP and its impact on send rate, taking into account the dependence of congestion avoidance on ACK behavior, the manner in which packet loss is inferred (e.g., whether by duplicate ACK detection and fast retransmit, or by timeout), limited receiver window size, and average round trip time

(RTT). Our model is based on the Reno flavor of TCP, as it is by far the most popular implementation in the Internet today [48, 49]. We assume that the reader is familiar with TCP Reno congestion control (see for example [2, 21, 65]) and we adopt most of our terminology from [2, 21, 65].

Our model focuses on TCP's congestion avoidance mechanism, where TCP's congestion control window size, W, is increased by $1/W$ each time an ACK is received. Conversely, the window is decreased whenever a lost packet is detected, with the amount of the decrease depending on whether packet loss is detected by duplicate ACKs or by timeout, as discussed shortly.

We model TCP's congestion avoidance behavior in terms of rounds. A round starts with the back-to-back transmission of W packets, where W is the current size of the TCP congestion window. Once all packets falling within the congestion window have been sent in this back-to-back manner, no other packets are sent until the first ACK is received for one of these W packets. This ACK reception marks the end of the current round and the beginning of the next round. In this model, the duration of a round is equal to the round trip time and is assumed to be independent of the window size, an assumption also adopted (either implicitly or explicitly) in [32, 34, 41]. Note that we have also assumed here that the time needed to send all the packets in a window is smaller than the round trip time; this behavior can be seen in observations reported in [9, 48].

Let b be the number of packets that are acknowledged by a received ACK. Many TCP receiver implementations send one cumulative ACK for two consecutive packets received (i.e., delayed ACK, [65]), so b is typically 2. If W packets are sent in the first round and are all received and acknowledged correctly, then W/b acknowledgments will be received. Since each acknowledgment increases the window size by $1/W$, the window size at the beginning of the second round is then $W' = W + 1/b$. That is, during congestion avoidance and in the absence of loss, the window size increases linearly in time, with a slope of $1/b$ packets per round trip time.

We model TCP's behavior in the presence of packet loss. Packet loss can be detected in one of two ways, either by the reception at the TCP sender of "triple-duplicate" acknowledgments, i.e., four ACKs with the same sequence number, or via time-outs. We denote the former event as a "TD" (triple-duplicate) loss indication, and the latter as a "TO" loss indication.

We assume that a packet is lost in a round independently of any packets lost in *other* rounds. On the other hand, we assume that packet losses are correlated among the back-to-back transmissions within a round: if a packet is lost, we assume that all remaining packets transmitted until the end of that round are also lost. This bursty loss model is a simple and crude approximation to capture the loss behavior observed in studies such as [75]. We discuss this assumption further in Section 2.4.

We develop our model of TCP congestion control in several steps, corresponding to its operating regimes: when loss indications are exclusively TD, when loss indications are both TD and TO, and when the congestion window size is limited by the receiver's advertised window. We note that we do not model certain aspects of TCP's behavior (e.g., fast recovery) but believe we have captured the essential elements of TCP behavior, as indicated by the generally very good fits between model predictions and measurements made on numerous commercial TCP implementations, as discussed in Section 2.3. A more detailed discussion of model assumptions and related issues is presented in Section 2.4. Also note that in the following, we measure send rate in terms of packets per unit of time, instead of bytes per unit of time.

2.2.1 Loss indications are exclusively triple-duplicate ACKs

In this section we assume that loss indications are exclusively of type "triple-duplicate" ACK (TD), and that the window size is not limited by the receiver's

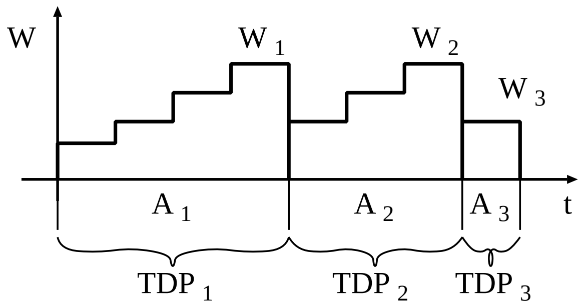


Figure 2.1. Evolution of window size over time when loss indications are triple duplicate ACKs

advertised flow control window. We consider a TCP flow starting at time $t = 0$, where the sender always has data to send. For any given time $t > 0$, we define N_t to be the number of packets transmitted in the interval $[0, t]$, and $B_t = N_t/t$, the send rate on that interval. Note that B_t is the number of packets sent per unit of time regardless of their eventual fate (i.e., whether they are received or not). Thus, B_t represents the send rate of the connection. We define the long-term steady-state TCP send rate B to be

$$B = \lim_{t\to\infty} B_t = \lim_{t\to\infty} \frac{N_t}{t} \tag{2.1}$$

We have assumed that if a packet is lost in a round, all remaining packets transmitted until the end of the round are also lost. Therefore we define p to be the probability that a packet is lost, given that either it is the first packet in its round or the preceding packet in its round is not lost. We are interested in establishing a relationship, $B(p)$, between the send rate of the TCP connection and p, the loss probability defined above.

A sample path of the evolution of congestion window size is given in Figure 2.1. Between two TD loss indications, the sender is in congestion avoidance, and the

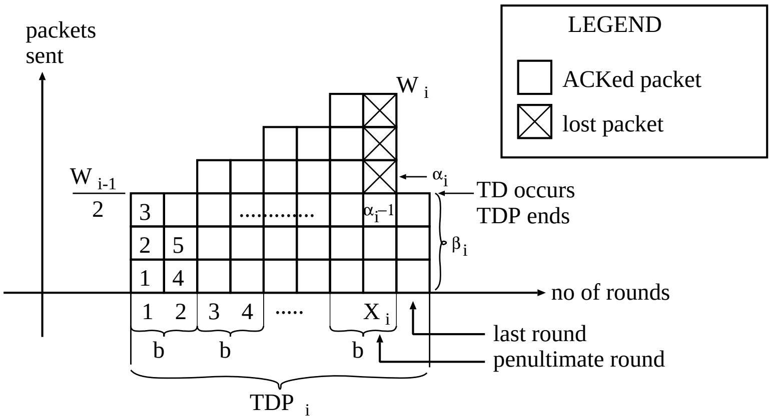


Figure 2.2. Packets sent during a TD period

window increases by $1/b$ packets per round, as discussed earlier. Immediately after the loss indication occurs, the window size is reduced by a factor of two.

We define a TD period (TDP) to be the interval of time between two TD loss indications (see Figure 2.1). For the i-th TD period we define the random variable Y_i to be the number of packets sent in the period, A_i the duration of the period, and W_i the window size at the end of the period. Considering $\{W_i\}_i$ to be a Markov regenerative process with rewards $\{Y_i\}_i$ (see for example [59]), it can be shown that

$$B = \frac{E[Y]}{E[A]} \tag{2.2}$$

In order to derive an expression for B, the long-term steady-state TCP send rate, we must next derive expressions for the mean of Y and A.

Consider a TD period as in Figure 2.2. A TD period starts immediately after a TD loss indication, and thus the congestion window size at the start of the i^{th} TD period is equal to $W_{i-1}/2$, half the size of window before the TD occurred. At each round the window is incremented by $1/b$ and the number of packets sent per round is incremented by one every b rounds. We denote by α_i the first packet lost in TDP_i, and by X_i the round where this loss occurs (see Figure 2.2). After packet α_i, $W_i - 1$

more packets are sent in an additional round before a TD loss indication occurs (and the current TD period ends), as discussed in more detail in Section 2.2.2. Thus, a total of $Y_i = \alpha_i + W_i - 1$ packets are sent in $X_i + 1$ rounds. It follows that:

$$E[Y] = E[\alpha] + E[W] - 1 \tag{2.3}$$

To derive $E[\alpha]$, consider the random process $\{\alpha_i\}_i$, where α_i is the number of packets sent in a TD period up to and including the first packet that is lost. Based on our assumption that packets are lost in a round independently of any packets lost in *other* rounds, $\{\alpha_i\}_{i=0}^{\infty}$ is a sequence of independent and identically distributed (i.i.d.) random variables. Given our loss model, the probability that $\alpha_i = k$ is equal to the probability that exactly $k-1$ packets are successfully acknowledged before a loss occurs

$$P[\alpha = k] = (1-p)^{k-1}p, \quad k = 1, 2, \ldots \tag{2.4}$$

The mean of α is thus

$$E[\alpha] = \sum_{k=1}^{\infty}(1-p)^{k-1}pk = \frac{1}{p} \tag{2.5}$$

From (2.3) and (2.5) it follows that

$$E[Y] = \frac{1-p}{p} + E[W] \tag{2.6}$$

To derive $E[W]$ and $E[A]$, consider again TDP_i. We define r_{ij} to be the duration (round trip time) of the j-th round of TDP_i. Then, the duration of TDP_i is $A_i = \sum_{j=1}^{X_i+1} r_{ij}$. We consider the round trip times r_{ij} to be random variables, that are assumed to be independent of the size of congestion window, and thus independent of the round number, j. It follows that

$$E[A] = (E[X] + 1)E[r] \tag{2.7}$$

Henceforth, we denote by $RTT = E[r]$ as the average value of round trip time.

Finally, to derive an expression for $E[X]$, we consider the evolution of W_i as a function of the number of rounds, as in Figure 2.2. To simplify our exposition, in this derivation we assume that $W_{i-1}/2$ and X_i/b are integers. First we observe that during the i-th TD period, the window size increases between $W_{i-1}/2$ and W_i. Since the increase is linear with slope $1/b$, we have:

$$W_i = \frac{W_{i-1}}{2} + \frac{X_i}{b}, \quad i = 1, 2, \ldots \tag{2.8}$$

The fact that Y_i packets are transmitted in TDP_i is expressed by

$$\begin{aligned}
Y_i &= \sum_{k=0}^{X_i/b-1} (\frac{W_{i-1}}{2} + k)b + \beta_i && (2.9)\\
&= \frac{X_i W_{i-1}}{2} + \frac{X_i}{2}(\frac{X_i}{b} - 1) + \beta_i && (2.10)\\
&= \frac{X_i}{2}(\frac{W_{i-1}}{2} + W_i - 1) + \beta_i \quad \text{using (2.8)} && (2.11)
\end{aligned}$$

where β_i is the number of packets sent in the last round (see Figure 2.2). $\{W_i\}_i$ is a Markov process for which a stationary distribution can be obtained numerically, based on (2.8) and (2.11) and on the probability density function of $\{\alpha_i\}$ given in (2.4). We can also compute the probability distribution of $\{X_i\}$. However, a simpler approximate solution can be obtained by assuming that $\{X_i\}$ and $\{W_i\}$ are mutually independent sequences of i.i.d. random variables. With this assumption, it follows from (2.8), (2.11) and (2.6) that

$$E[W] = \frac{2}{b}E[X] \tag{2.12}$$

and,

$$\frac{1-p}{p} + E[W] = \frac{E[X]}{2}\left(\frac{E[W]}{2} + E[W] - 1\right) + E[\beta] \tag{2.13}$$

We consider that β_i, the number of packets in the last round, is uniformly distributed between 1 and W_i, and thus $E[\beta] = E[W]/2$. From (2.12) and (2.13), we have

$$E[W] = \frac{2+b}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{2+b}{3b}\right)^2} \tag{2.14}$$

Observe that,

$$E[W] = \sqrt{\frac{8}{3bp}} + o(1/\sqrt{p}) \tag{2.15}$$

i.e., $E[W] \approx \sqrt{\frac{8}{3bp}}$ for small values of p. From (2.12), (2.7) and (2.14), it follows

$$E[X] = \frac{2+b}{6} + \sqrt{\frac{2b(1-p)}{3p} + \left(\frac{2+b}{6}\right)^2} \tag{2.16}$$

$$E[A] = RTT\left(\frac{2+b}{6} + \sqrt{\frac{2b(1-p)}{3p} + \left(\frac{2+b}{6}\right)^2} + 1\right) \tag{2.17}$$

Observe that,

$$E[X] = \sqrt{\frac{2b}{3p}} + o(1/\sqrt{p}) \tag{2.18}$$

From (2.2) and (2.6) we have

$$B(p) = \frac{\frac{1-p}{p} + E[W]}{E[A]} \tag{2.19}$$

$$= \frac{\frac{1-p}{p} + \frac{2+b}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{2+b}{3b}\right)^2}}{RTT\left(\frac{2+b}{6} + \sqrt{\frac{2b(1-p)}{3p} + (\frac{2+b}{6})^2} + 1\right)} \tag{2.20}$$

Which can be expressed as:

$$B(p) = \frac{1}{RTT}\sqrt{\frac{3}{2bp}} + o(1/\sqrt{p}) \tag{2.21}$$

Thus, for small values of p, (2.21) reduces to the send rate formula in [32] for $b = 1$.

We next extend our model to include TCP behaviors (such as timeouts and receiver-limited windows) not considered in previous analytic studies of TCP congestion control.

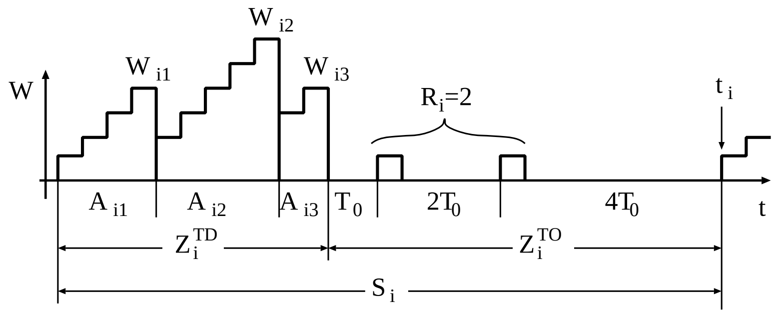


Figure 2.3. Evolution of window size when loss indications are triple-duplicate ACKs and time-outs

2.2.2 Loss indications are triple-duplicate ACKs and time-outs

So far, we have considered TCP flows where all loss indications are due to "triple-duplicate" ACKs. Our measurements show (see Table 2.2) that in many cases the majority of window decreases are due to time-outs, rather than fast retransmits. Therefore, a good model should capture time-out loss indications.

In this section we extend our model to include the case where the TCP sender times-out. This occurs when packets (or ACKs) are lost, and less than three duplicate ACKs are received. The sender waits for a period of time denoted by T_0, and then retransmits the first non-acknowledged packet. Following a time-out, the congestion window is reduced to one, and one packet is thus resent in the first round after a time out. If this retransmission is unsuccessful, the period of time out doubles to $2T_0$; this doubling is repeated for each unsuccessful retransmission until $64T_0$ is reached, after which the time out period remains constant at $64T_0$.

An example of the evolution of congestion window size is given in Figure 2.3. Let Z_i^{TO} denote the duration of a sequence of time-outs and Z_i^{TD} the time interval between two consecutive time-out sequences. Define S_i to be

$$S_i = Z_i^{TD} + Z_i^{TO}$$

Also, define M_i to be the number of packets sent during S_i. Then, $\{(S_i, M_i)\}_i$ is an i.i.d. sequence of random variables, and we have

$$B = \frac{E[M]}{E[S]}$$

We extend our definition of TD periods given in Section 2.2.1 to include periods starting after, or ending in, a TO loss indication (besides periods between two TD loss indications). Let n_i be the number of TD periods in interval Z_i^{TD}. For the j-th TD period of interval Z_i^{TD} we define Y_{ij} to be the number of packets sent in the period, A_{ij} to be the duration of the period, X_{ij} to be the number of rounds in the period, and W_{ij} to be the window size at the end of the period. Also, R_i denotes the number of packets sent during time-out sequence Z_i^{TO}. Observe here that R_i counts the total number of packet transmissions in Z_i^{TO}, and not just the number of different packets sent. This is because, as discussed in Section 2.2.1, we are interested in the send rate of a TCP flow, rather than its goodput. We have

$$M_i = \sum_{j=1}^{n_i} Y_{ij} + R_i, \quad S_i = \sum_{j=1}^{n_i} A_{ij} + Z_i^{TO}$$

and, thus,

$$E[M] = E[\sum_{j=1}^{n_i} Y_{ij}] + E[R], \quad E[S] = E[\sum_{j=1}^{n_i} A_{ij}] + E[Z^{TO}]$$

If we assume $\{n_i\}_i$ to be an i.i.d. sequence of random variables, independent of $\{Y_{ij}\}$ and $\{A_{ij}\}$, then we have

$$E[(\sum_{j=1}^{n_i} Y_{ij})_i] = E[n]E[Y], \quad E[(\sum_{j=1}^{n_i} A_{ij})_i] = E[n]E[A]$$

To derive $E[n]$ observe that, during Z_i^{TD}, the time between two consecutive time-out sequences, there are n_i TDPs, where each of the first $n_i - 1$ end in a TD, and the last

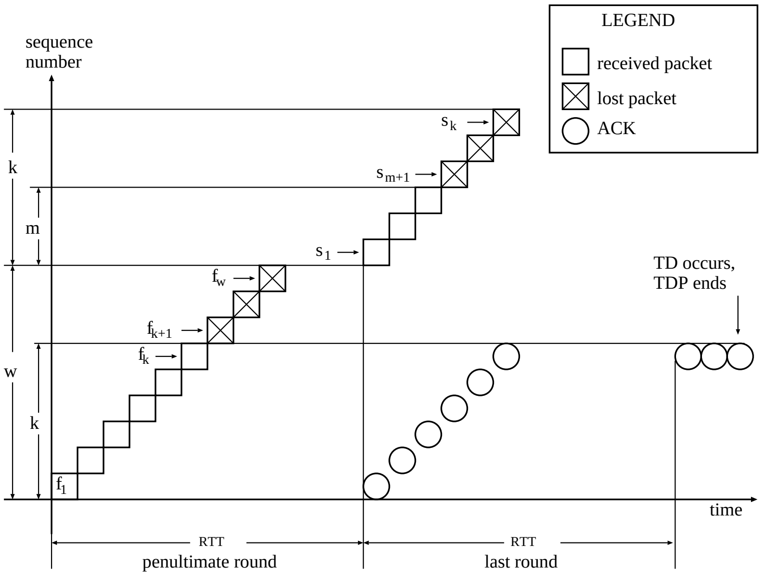


Figure 2.4. Packet and ACK transmissions preceding a loss indication

TDP ends in a TO. It follows that in Z_i^{TD} there is one TO out of n_i loss indications. Therefore, if we denote by Q the probability that a loss indication ending a TDP is a TO, we have $Q = 1/E[n]$. Consequently,

$$B = \frac{E[Y] + Q * E[R]}{E[A] + Q * E[Z^{TO}]} \tag{2.22}$$

Since Y_{ij} and A_{ij} do not depend on time-outs, their means are those derived in (2.5) and (2.17). To compute TCP send rate using (2.22) we must still determine $Q, E[R]$, and $E[Z^{TO}]$.

We begin by deriving an expression for Q. Consider the round of packets where a loss indication occurs; this round will be referred to as the "penultimate" round (see Figure 2.4) Note that in Figure 2.4 each ACK acknowledges individual packets (i.e., ACKs are not delayed). We have chosen this for simplicity of illustration. We will see that the analysis does not depend on whether ACKs are delayed or not. Let w be the current congestion window size. Thus packets $f_1, \ldots, f_w$ are sent in the penultimate round. Packets $f_1, \ldots, f_k$ are acknowledged, and packet f_{k+1} is the first one to be lost (or not ACKed). We again assume that packet losses are correlated within a round: if a packet is lost, so too are all packets that follow, until the end of the round. Thus, all packets following f_{k+1} in the penultimate round are also lost. However, since packets $f_1, \ldots, f_k$ are ACKed, another k packets, $s_1, \ldots, s_k$ are sent in the next round, which we will refer to as the "last" round. This round of packets may have another loss, say packet s_{m+1}. Again, our assumptions on packet loss correlation mandates that packets $s_{m+2}, \ldots, s_k$ are also lost in the last round. The m packets successfully sent in the last round are responded to by ACKs for packet f_k, which are counted as duplicate ACKs. These ACKs are not delayed ([65], p. 312), so the number of duplicate ACKs is equal to the number of successfully received packets in the last round. If the number of such ACKs is higher than three, then a TD indication occurs, otherwise, a TO occurs. In both cases the current period between losses, TDP, ends. We denote by $A(w, k)$ the probability that the first k packets are ACKed in a round of w packets, given there is a sequence of one or more losses in the round. Then

$$A(w, k) = \frac{(1-p)^k p}{1-(1-p)^w} \quad 1 \leq k \leq w$$

Also, we define $C(n, m)$ to be the probability that m packets are ACKed in sequence in the last round (where n packets were sent) and the rest of the packets in the round, if any, are lost. Then,

$$C(n, m) = \begin{cases} (1-p)^m p, & m \leq n-1 \\ (1-p)^n, & m = n \end{cases}$$

Then, $\hat{Q}(w)$, the probability that a loss in a window of size w is a TO, is given by

$$\hat{Q}(w) = \begin{cases} 1 & \text{if } w \leq 3 \\ \sum_{k=0}^{2} A(w,k) + \sum_{k=3}^{w} A(w,k) \sum_{m=0}^{2} C(k,m) & \text{otherwise} \end{cases} \tag{2.23}$$

since a TO occurs if the number of packets successfully transmitted in the penultimate round, k, is less than three, or otherwise if the number of packets successfully transmitted in the last round, m is less than three. Also, due to the assumption that packet s_{m+1} is lost independently of packet f_{k+1} (since they occur in different rounds), the probability that there is a loss at f_{k+1} in the penultimate round and a loss at s_{m+1} in the last round equals $A(w,k) * C(k,m)$, and (2.23) follows.

After algebraic manipulations, we have

$$\hat{Q}(w) = \min\left(1, \frac{(1-(1-p)^3)(1+(1-p)^3(1-(1-p)^{w-3}))}{1-(1-p)^w}\right) \tag{2.24}$$

Observe (for example, using L'Hopital's rule) that

$$\lim_{p\to 0} \hat{Q}(w) = \frac{3}{w}.$$

Numerically we find that a very good approximation of $\hat{Q}$ is

$$\hat{Q}(w) \approx \min(1, \frac{3}{w}) \tag{2.25}$$

Q, the probability that a loss indication is a TO, is

$$Q = \sum_{w=1}^{\infty} \hat{Q}(w) P[W=w] = E[\hat{Q}]$$

We approximate

$$Q \approx \hat{Q}(E[W]) \tag{2.26}$$

where $E[W]$ is from (2.14).

We consider next the derivation of $E[R]$ and $E[Z^{TO}]$. For this, we need the probability distribution of the number of timeouts in a TO sequence, given that there is a TO. A sequence of k TOs occurs when there are $k-1$ consecutive losses (the first loss is given) followed by a successfully transmitted packet. Consequently, the number of TOs in a TO sequence has a geometric distribution, and thus

$$P[R=k] = p^{k-1}(1-p)$$

Then we can compute the mean of R:

$$E[R] = \sum_{k=1}^{\infty} kP[R=k] = \frac{1}{1-p} \tag{2.27}$$

Next, we focus on $E[Z^{TO}]$, the average duration of a time-out sequence excluding retransmissions, which can be computed in a similar way. We know that the first six time-outs in one sequence have length $2^{i-1}T_0$, $i=1,\ldots,6$, with all immediately following timeouts having length $64T_0$. Then, the duration of a sequence with k time-outs is

$$L_k = \begin{cases} (2^k-1)T_0 & \text{for } k \leq 6 \\ (63+64(k-6))T_0 & \text{for } k \geq 7 \end{cases}$$

and the mean of Z^{TO} is

$$\begin{aligned} E[Z^{TO}] &= \sum_{k=1}^{\infty} L_k P[R=k] \\ &= T_0 \frac{1+p+2p^2+4p^3+8p^4+16p^5+32p^6}{1-p} \end{aligned}$$

Armed now with expressions for $Q, E[S], E[R]$ and $E[Z^{TO}]$ we can now substitute these expressions into equation (2.22) to obtain the following for $B(p)$:

$$B(p) = \frac{\frac{1-p}{p} + E[W] + \hat{Q}(E[W])\frac{1}{1-p}}{RTT(E[X]+1) + \hat{Q}(E[W])T_0\frac{f(p)}{1-p}} \quad 0 < p < 1 \tag{2.28}$$

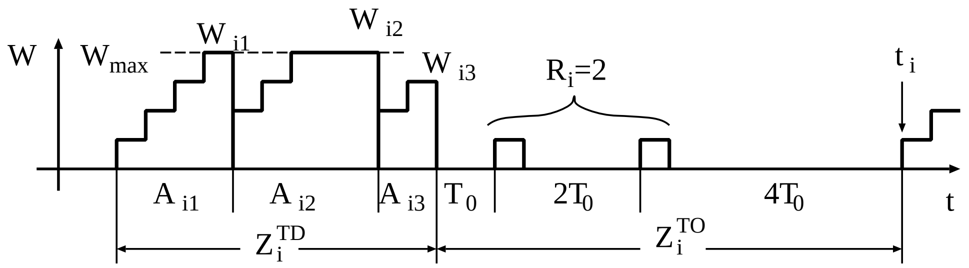


Figure 2.5. Evolution of window size when limited by W_{max}

where:

$$f(p) = 1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6 \tag{2.29}$$

$\hat{Q}$ is given in (2.24), $E[W]$ in (2.14) and $E[X]$ in (2.17). Using (2.25), (2.15) and (2.18), we have that (2.28) can be approximated by

$$B(p) \approx \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right)p(1+32p^2)} \tag{2.30}$$

2.2.3 The impact of window limitation

So far, we have not considered any limitation on the congestion window size. At the beginning of TCP flow establishment, however, the receiver advertises a maximum buffer size which determines a maximum congestion window size, W_{max}. As a consequence, during a period without loss indications, the window size can grow up to W_{max}, but will not grow further beyond this value. An example of the evolution of window size is depicted in Figure 2.5.

To simplify the analysis of the model, we make the following assumption. Let W_u denote the unconstrained window size, the mean of which is given in (2.14)

$$E[W_u] = \frac{2+b}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{2+b}{3b}\right)^2} \tag{2.31}$$

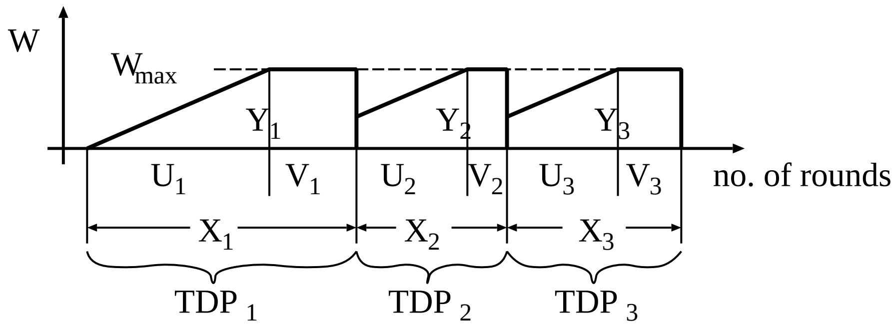


Figure 2.6. Fast retransmit with window limitation

We assume that if $E[W_u] < W_{max}$, we have the approximation $E[W] \approx E[W_u]$. In other words, if $E[W_u] < W_{max}$, the receiver-window limitation has negligible effect on the long term average of the TCP send rate, and thus the TCP send rate is given by (2.28).

On the other hand, if $W_{max} \leq E[W_u]$, we approximate $E[W] \approx W_{max}$. In this case, consider an interval Z^{TD} between two time-out sequences consisting of a series of TD periods as in Figure 2.6. During the i^{th} TDP, the window grows linearly up to W_{max} for U_i rounds, then remains constant for V_i rounds, at which point a TD indication then occurs. The window then drops to $W_{max}/2$, and the process repeats. Thus,

$$W_{max} = \frac{W_{max}}{2} + \frac{U_i}{b}, \quad \forall i \geq 2$$

which implies $E[U] = (b/2)W_{max}$. Also, considering the number of packets sent in the i-th TD period, we have

$$Y_i = \frac{U_i}{2}(\frac{W_{max}}{2} + W_{max}) + V_i W_{max}$$

and then

$$E[Y] = \frac{3}{4}W_{max}E[U] + W_{max}E[V] = \frac{3b}{8}W_{max}^2 + W_{max}E[V]$$

Since Y_i, the number of packets in the i-th TD period, does not depend on window limitation, $E[Y]$ is given by (2.6), $E[Y] = (1-p)/p + W_{max}$, and thus

$$E[V] = \frac{1-p}{pW_{max}} + 1 - \frac{3b}{8}W_{max}$$

Finally, since $X_i = U_i + V_i$, we have

$$E[X] = E[U] + E[V] = \frac{b}{8}W_{max} + \frac{1-p}{pW_{max}} + 1$$

By substituting this result in (2.28), we obtain the TCP send rate, $B(p)$, when the window is limited

$$B(p) = \frac{\frac{1-p}{p} + W_{max} + \hat{Q}(W_{max})\frac{1}{1-p}}{RTT(\frac{b}{8}W_{max} + \frac{1-p}{pW_{max}} + 2) + \hat{Q}(W_{max})T_0\frac{f(p)}{1-p}}$$

In conclusion, the complete characterization of TCP send rate, $B(p)$, is:

$$B(p) = \begin{cases} \dfrac{\frac{1-p}{p}+E[W]+\hat{Q}(E[W])\frac{1}{1-p}}{RTT(\frac{b}{2}E[W_u]+1)+\hat{Q}(E[W])T_0\frac{f(p)}{1-p}} & \text{if } E[W_u] < W_{max} \\ \\ \dfrac{\frac{1-p}{p}+W_{max}+\hat{Q}(W_{max})\frac{1}{1-p}}{RTT(\frac{b}{8}W_{max}+\frac{1-p}{pW_{max}}+2)+\hat{Q}(W_{max})T_0\frac{f(p)}{1-p}} & \text{otherwise} \end{cases} \tag{2.32}$$

where $f(p)$ is given in (2.29), $\hat{Q}$ is given in (2.24) and $E[W_u]$ in (2.14). In the following sections we will refer to (2.32) as the "full model". The following approximation of $B(p)$ follows from (2.30) and (2.32):

$$B(p) \approx \min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{2bp/3} + T_0 \min\left(1, 3\sqrt{3bp/8}\right) p(1+32p^2)}\right) \tag{2.33}$$

In Section 2.3 we verify that equation (2.33) is indeed a very good approximation of equation (2.32). Henceforth we will refer to (2.33) as the "approximate model".

Receiver	Domain	Operating System
ada	hofstra.edu	Irix 6.2
afer	cs.umn.edu	Linux
al	cs.wm.edu	Linux 2.0.31
alps	cc.gatech.edu	SunOS 4.1.3
babel	cs.umass.edu	SunOS 5.5.1
baskerville	cs.arizona.edu	SunOS 5.5.1
ganef	cs.ucla.edu	SunOS 5.5.1
imagine	cs.umass.edu	win95
manic	cs.umass.edu	Irix 6.2
mafalda	inria.fr	SunOS 5.5.1
maria	wustl.edu	SunOS 4.1.3
modi4	ncsa.uiuc.edu	Irix 6.2
pif	inria.fr	Solaris 2.5
pong	usc.edu	HP-UX
spiff	sics.se	SunOS 4.1.4
sutton	cs.columbia.edu	SunOS 5.5.1
tove	cs.umd.edu	SunOS 4.1.3
void	cs.umass.edu	Linux 2.0.30
att	att.com	Linux

Table 2.1. Domains and operating systems of hosts

2.3 Measurements and trace analysis

Equations (2.32) and (2.33) provide an analytic characterization of TCP send rate as a function of packet loss indication rate, RTT, and maximum window size. In this section we empirically validate these formulae, using measurement data from several TCP connections established between hosts scattered across United States and Europe.

Table 2.1 lists the domains and operating systems of the hosts used for the measurements[1]. All data sets are for unidirectional bulk data transfers. We gathered the measurement data by running `tcpdump` at the sender, and analyzing its output with a set of analysis programs developed by us. These programs account for various measurement- and implementation-related problems discussed in [48, 49]. For example, when we analyze traces from a Linux sender, we account for the fact that TD

[1]The hostname for machine located in the `att.com` domain has been altered due to security concerns.

events occur after getting only two duplicate acks instead of three. Our trace analysis programs were further verified by checking them against `tcptrace` [40] and `ns` [10].

We carried out two different sets of measurement experiments. Table 2.2 summarizes data from the first set. Each row in the table corresponds to a one hour long TCP connection in which the sender behaves as an "infinite source" – it always has data to send and thus the TCP send rate is only limited by TCP's congestion control. The experiments were performed at randomly selected times during 1997 and the beginning of 1998. The third and fourth column of Table 2.2 indicate the number of packets sent and the number of loss indications respectively (triple duplicate ack or timeout). Dividing the total number of loss indications by the total number of packets sent gives us an approximate value of p. This approximation is similar to the one used in [34]. The next six columns show a breakdown of the loss indications by type: the number of TD events, the number of "single" timeouts, having duration T_0, the number of "double" timeouts, $T_1 = 2T_0$, etc. Note that p depends only on the *total* number of loss indications, and not on their type. The last two columns report the average value of round trip time, and the average duration of a "single" timeout T_0. These values have been averaged over the entire trace. When calculating round trip time values, we follow Karn's algorithm, in an attempt to minimize the impact of timeouts and retransmissions on the RTT estimates. An important observation to be drawn from the data in these tables is that in all traces, timeouts constitute the majority or a significant fraction of the total number of loss indications. This underscores the importance of including the effects of timeouts in the model of TCP congestion control. In addition to "single" timeout events (column T_0), it can be seen that exponential backoff (multiple timeouts) occurs with significant frequency.

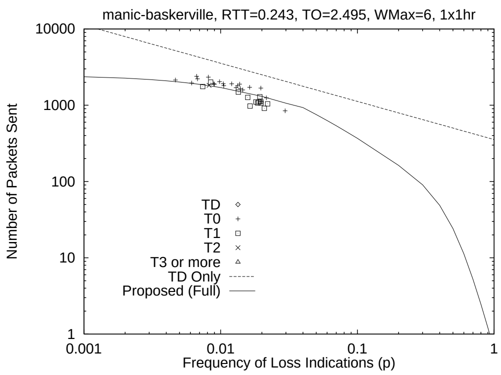
For the second set of experiments, we established 100 serially-initiated TCP connections between a given sender-receiver pair. Each connection lasted for 100 seconds, and was followed by a 50 second gap before the next connection was initiated. These

Sender	Rcvr	Pkts	Loss Indic.	TD	T_0	T_1	T_2	T_3	T_4	T_5+	RTT	Time Out
manic	alps	54402	722	19	611	67	15	6	2	2	0.207	2.505
manic	baskerville	58120	735	306	411	17	1	0	0	0	0.243	2.495
manic	ganef	58924	743	272	444	22	4	1	0	0	0.226	2.405
manic	mafalda	56283	494	2	474	17	1	0	0	0	0.233	2.146
manic	maria	68752	649	1	604	35	8	1	0	0	0.180	2.416
manic	spiff	117992	784	47	702	34	1	0	0	0	0.211	2.274
manic	sutton	81123	1638	988	597	41	7	3	1	1	0.204	2.459
manic	tove	7938	264	1	190	37	18	8	3	7	0.275	3.597
void	alps	37137	838	7	588	164	56	17	4	2	0.162	0.489
void	baskerville	32042	853	339	430	67	12	5	0	0	0.482	1.094
void	ganef	60770	1112	414	582	79	20	9	4	2	0.254	0.637
void	maria	93005	1651	33	1344	197	54	15	5	3	0.152	0.417
void	spiff	65536	671	72	539	56	4	0	0	0	0.415	0.749
void	sutton	78246	1928	840	863	152	45	18	9	1	0.211	0.601
void	tove	8265	856	5	444	209	100	51	27	12	0.272	1.356
babel	alps	13460	1466	0	1068	247	87	33	18	8	0.194	1.359
babel	baskerville	62237	1753	197	1467	76	10	3	0	0	0.253	0.429
babel	ganef	86675	2125	398	1686	38	2	1	0	0	0.201	0.306
babel	spiff	57687	1120	0	939	137	36	7	1	0	0.331	0.953
babel	sutton	83486	2320	685	1448	142	31	9	4	1	0.210	0.705
babel	tove	83944	1516	1	1364	118	17	7	5	3	0.194	0.520
pif	alps	83971	762	0	577	111	46	16	8	2	0.168	7.278
pif	imagine	44891	1346	15	1044	186	63	21	10	5	0.229	0.700
pif	manic	34251	1422	43	944	272	105	36	14	6	0.257	1.454

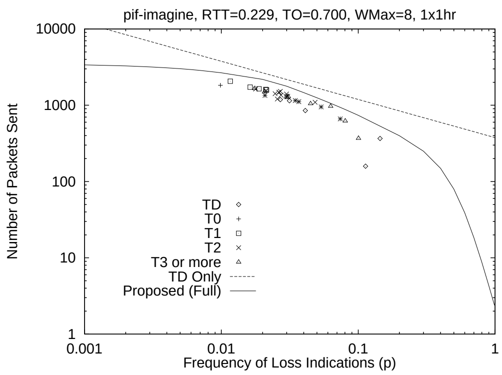
Table 2.2. Summary data from 1hr traces

experiments were performed at randomly selected times during 1998. These connections showed loss patterns similar to those observed for the 1-hour long connections.

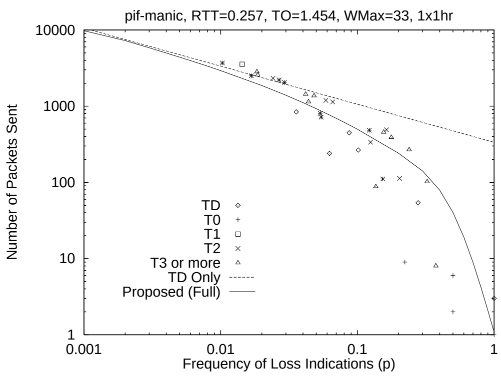
The graphs in Figure 2.7 compare the predictions of the proposed model, and the predictions of the model proposed in [34], with measurement data for one-hour long traces. The title of each graph indicates the average round trip time, the average "single" timeout duration T_0, and the maximum window size W_{max} advertised by the receiver (in number of packets). To plot the graph, each one-hour trace was divided into 36 consecutive 100 seconds intervals, and each plotted point on a graph represents the number of packets sent versus the frequency of loss indications during a 100 second interval. While dividing a continuous trace into fixed sized intervals can lead to some inaccuracies in measuring p, (e.g., the interval boundaries may occur within timeout



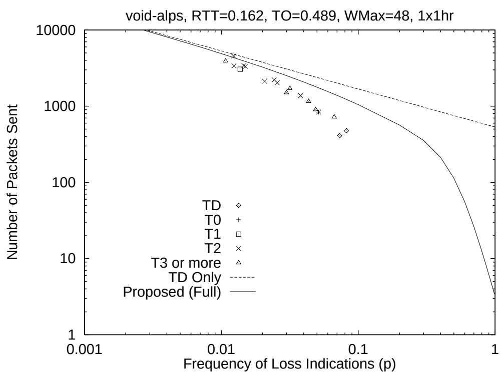
(a) manic to baskerville



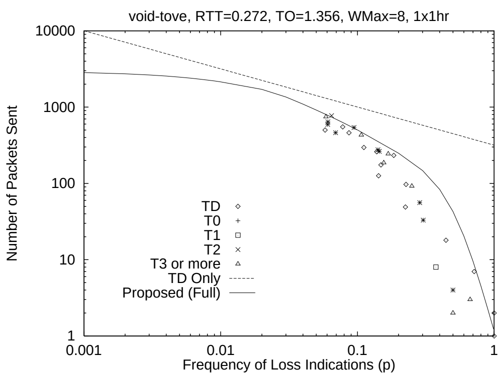
(b) pif to imagine



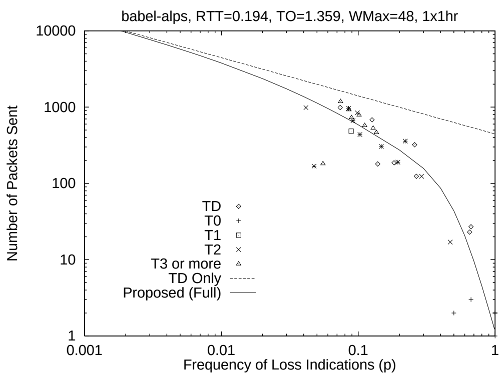
(c) pif to manic



(d) void to alps



(e) void to tove



(f) babel to alps

Figure 2.7. 1 hour traces

intervals, thus perhaps not attributing a loss event to the interval where most of its impact is felt), we believe that by using interval sizes of 100 seconds, which are longer than most timeouts, we have minimized the impact of such inaccuracies. Each 100 second interval is classified into one of several categories: intervals of type "TD" did not suffer any timeout (only triple duplicate acks), intervals of type "$T0$" suffered at least one "single" timeout but no exponential backoff, "$T1$" represents intervals that suffered a single exponential backoff at least once (i.e., a "double" timeout) etc. The line labeled "TD Only" (stands for Triple-Duplicate acks Only) plots the predictions made by the model described in [34], which is essentially the same model as described in [32], while accounting for delayed acks. The line labeled "Proposed (Full)" represents the model described by Equation (2.32). It has been pointed out in [32] that the "TD Only" model may not be accurate when the frequency of loss indications is higher than 5%. We observe that in many traces the frequency of loss indications is higher than 5% and that indeed the "TD Only" model predicts values for TCP send rate that are much higher than measured. Also, in several traces (see, for example, Figure 2.7(a)) we observe that TCP send rate is limited by the receiver's advertised window size. This is not accounted for in the "TD Only" model, and thus "TD Only" overestimates the send rate at low p values.

The graphs in Figure 2.8 compare the measured send rate with the predictions of the proposed model and the model in [34]. The title of each graph indicates the sender-receiver pair between which the measurements were carried out. As described earlier, each experiment consisted of 100 traces, each of which was 100 seconds in duration. For each trace, we measure the send rate, the loss rate, the round-trip time and T_0. We plot three points for each trace: one representing the measured send rate, a second representing the send rate predicted by the proposed model and the third representing the TD-only model in [34]. The points in each category are joined *only* *for better visual representation.* The x axis indicates the trace number and the y axis indicates the send rate, measured in terms of number of packets sent by the sender.

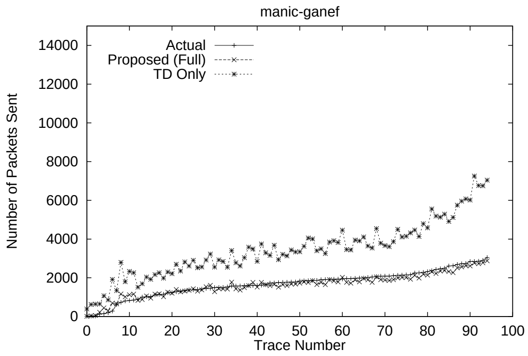
In order to accurately evaluate the models, we compute the average error as follows:

- **Hour-long traces:** We divide each trace into thirty-six 100-second intervals, and measure the number of packets sent during that interval. Let the number of packets sent in the i^{th} interval be denoted by N_i^m. We also calculate the value of the loss frequency for the i^{th} interval, and denote it by p_i. We calculate the average value of round trip time and timeout for the entire trace (these values are available in Table 2.2). Then, for each 100 second interval we calculate the number of packets predicted by our proposed model, $100 * B(p_i)$, where B is from (2.32). We denote this number by N_i^c. The average error for the trace is given by:

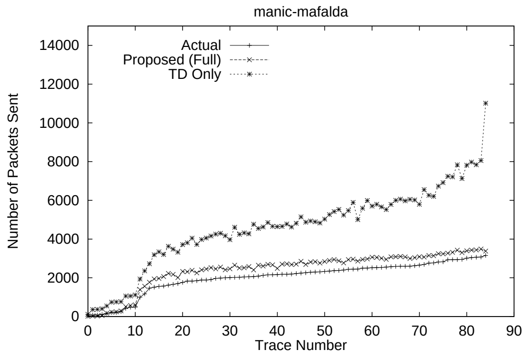
$$\frac{\sum_{i=1}^{36} \frac{|N_i^m - N_i^c|}{N_i^m}}{36}$$

  The average error of our approximate model (using B from (2.33)) and of "TD Only" are calculated in a similar manner. A smaller average error indicates better model accuracy. In Figure 2.9(a) we plot these error values to allow visual comparison. On the x-axis, the traces are identified by sender and receiver names. The order in which the traces appear is such that, from left to right, the average error for the "TD Only" model is increasing. The points corresponding to a given model are joined by line segments *only for better visual representation of the data.*

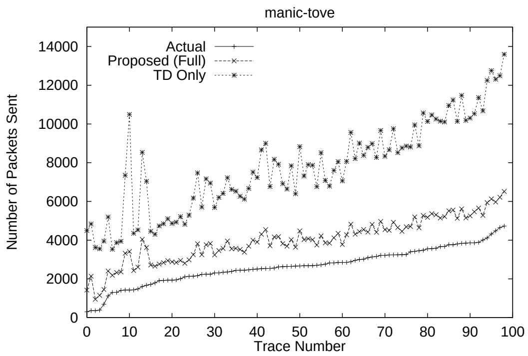
- **100-second traces:** We use the value of round trip time and timeout calculated for each 100-second trace. The error values are shown in Figure 2.9(b).



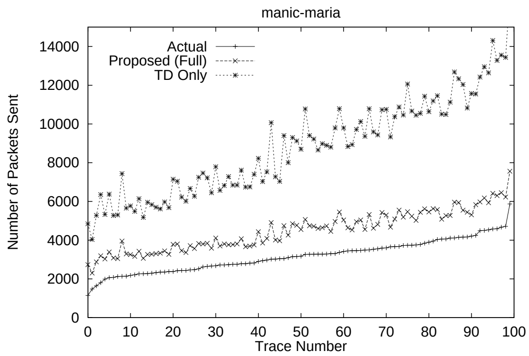
(a) manic to ganef



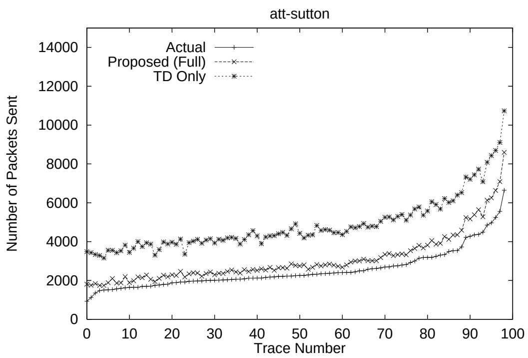
(b) manic to mafalda



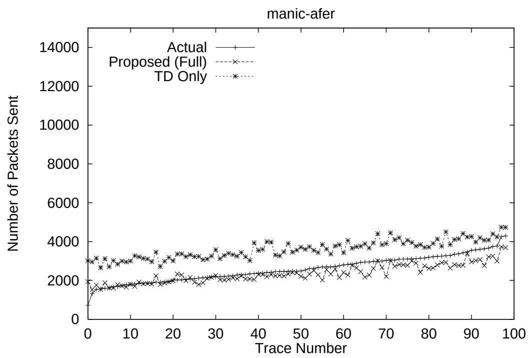
(c) manic to tove



(d) manic to maria

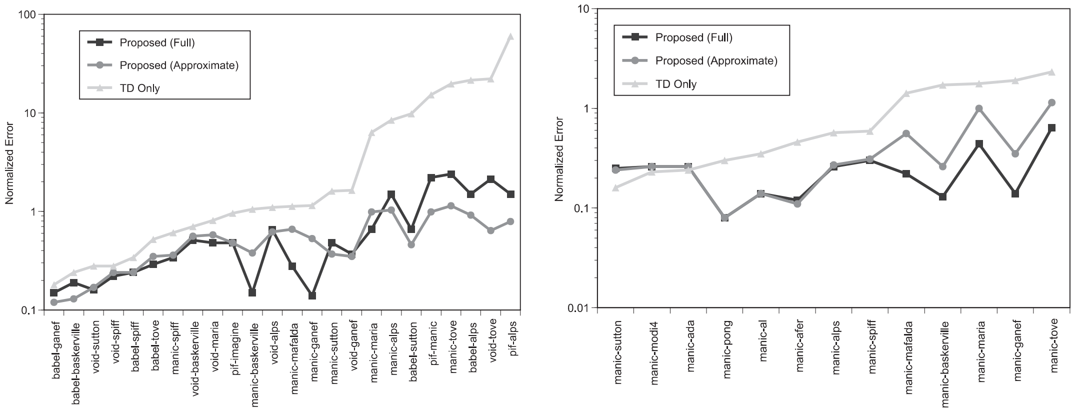


(e) att to sutton



(f) manic to afer

Figure 2.8. 100 second traces



(a) Comparison of the models for 1hr traces

(b) Comparison of the models for 100s traces

Figure 2.9. Model error

It can be seen from Figures 2.9(a) and 2.9(b) that in most cases, our proposed model is a better estimator of the observed values than the "TD Only" model. Our approximate model also generally provides more accurate predictions than the "TD Only" model, and is quite close to the predictions made by the full model. Empirical and simulation studies of the model proposed in this chapter have also been presented in [3, 5, 16, 53]. These studies have found that the model provides a good fit to the observed send rate of TCP connections under a wide variety of network conditions.

2.4 Discussion of the model and the experimental results

In this section, we discuss various simplifying assumptions made while constructing the model in Section 2.2, and their impact on the results described in Section 2.3.

Our model does not capture the subtleties of the *fast recovery* algorithm. We believe that the impact of this omission is quite small, and that the results presented in Section 2.3 validate this assumption indirectly. We have also assumed that the time spent in *slow start* is negligible compared to the length of our traces. Both these assumptions have also been made in [32, 34, 41].

We have assumed that packet losses within a round are *correlated* and losses in one round are *independent* of losses in other rounds. This assumption is supported by studies reported in [75]. The detailed loss model described in [75] also takes into account the burstiness of the packet loss process. However, the model is too complicated to allow derivation of closed-formed results. Thus a simple loss model was assumed. In our simulation studies the model was able to predict the throughput of TCP connections quite well, even with Bernoulli losses. Investigation of performance of TCP under various packet loss models is an area for future work.

Another assumption we made, that is also implicit in [32,34,41], is that the round trip time is independent of the window size. We have measured the coefficient of correlation between the duration of round samples and the number of packets in transit during each sample. For most traces summarized in Table 2.2, the coefficient of correlation is in the range of -0.1 to +0.1, thus lending credence to the statistical independence between round trip time and window size. However, when we conducted similar experiments with receivers at the end of a modem line, we found the coefficient of correlation to be as high as 0.97. We speculate that this is a combined effect of a slow link and a buffer devoted exclusively to this connection (probably at the ISP, just before the modem). As a result, our model, as well as the models described in [32, 34, 41] fail to match the observed data in the case of a receiver at the end of a modem. In Figure 2.10, we plot results from one such experiment. The receiver was a Pentium PC, running Linux 2.0.27 and was connected to the Internet via a commercial service provider using a 28.8Kbps modem. The results are for a 1 hour connection divided into 100 second intervals.

We have also assumed that all of our senders implement TCP-Reno as described in [2, 21, 65]. In [48, 49], it is observed that the implementation of the protocol stack

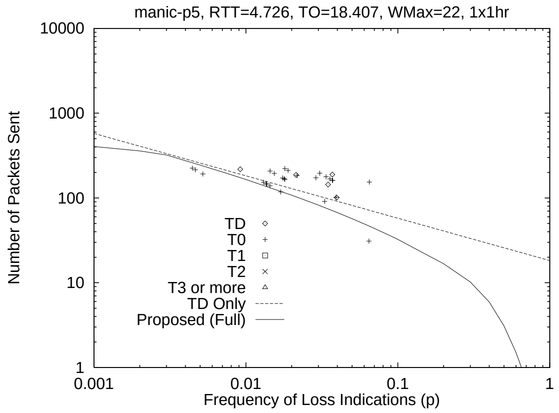


Figure 2.10. Effect of correlation between RTT and congestion window size

in each operating system is slightly different. While we have tried to account for the significant differences (for example in Linux the TD loss indications occur after two duplicate ACKs), we have not tried to customize our model for the nuances of each operating system. For example, we have observed that the Linux exponential backoff does not exactly follow the algorithm described in [2, 21, 65]. Our observations also seem to indicate that in the Irix implementation, the exponential backoff is limited to 2^5, instead of 2^6. We are also aware of the observation made in [49] that the SunOS TCP implementation is derived from Tahoe and not Reno. We have not customized our model for these cases.

During the course of the analysis presented in Section 2.2, we made several simplifying assumptions to obtain a closed form solution. A more detailed analysis, leading to a Markov model of TCP Reno is presented in Chapter 3. This Markov model does not appear to have a simple closed form solution. However, when solved numerically, the predictions of the Markov model closely match the predictions of the model proposed in this chapter.

2.5 Conclusion

In this chapter we have presented a simple model of the TCP-Reno protocol. The model captures the essence of TCP's congestion avoidance behavior and expresses send rate as a function of loss rate. The model takes into account the behavior of the protocol in the presence of timeouts, and is valid over the entire range of loss probabilities.

We have compared our model with the behavior of several real-world TCP connections. We observed that most of these connections suffered from a significant number of timeouts. We found that our model provides a very good match to the observed behavior in most cases, while models proposed in [32, 34, 41] significantly overestimate throughput. Thus, we conclude that timeouts have a significant impact on the performance of the TCP protocol, and that our model is able to account for this impact.

CHAPTER 3

A MARKOV MODEL OF TCP RENO

3.1 Introduction

In this chapter, we analyze the send rate of bulk transfer TCP flows using Markovian analysis that is more detailed than the simpler model presented in Chapter 2. We show that the predictions of the simple model in Chapter 2 closely match the predictions of the more precise model, thus validating the simple model.

Similar Markovian models of TCP have also been proposed in [26] and [41]. Our approach differs from these works in several ways. The model proposed in [41] is a fluid model, and measures the send rate of the connection. The model does not take into account the effect of TCP timeouts. As a result, the closed-form solution obtained in [41] resembles the formula derived in [32].

In [26], stochastic models for predicting the throughput of OldTahoe, Tahoe, Reno and NewReno flavors of TCP are proposed. The focus of the analysis is on LAN-like environment with wireless links. The packet-loss process is assumed to be Bernoulli and the impact of round trip time is not considered. In addition, timeouts are assumed to be limited to two successive backoffs. The actual timeout backoff mechanism [64] allows for six backoffs, and places no limit on the number of successive timeouts. The model, however, does take into account the impact of slow start and fast retransmits, which our model ignores.

The rest of this chapter is organized as follows. In Section 3.2, we present the Markovian model for TCP-Reno, and then in Section 3.3, we compare the send rate

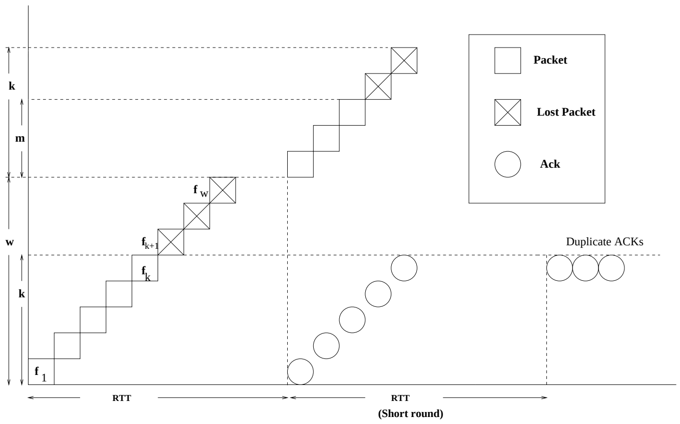


Figure 3.1. Packet and ACK transmissions preceding a loss indication

predicted by this model to the send rate predicted by the simple model proposed in Chapter 2. Section 3.4 concludes the chapter.

3.2 The model

We use the same terminology and loss model that was described in Section 2.2. To model the impact of packet losses, it is useful to consider Figure 3.1. Consider a round in which TCP's window size is w. Let f_k be the last packet in this round that makes it to the receiver. Thus, packets $f_{k+1} \dots f_w$ are lost. The sender will receive ACKs for the first k packets that made it to the receiver. This will result in a subsequent "short" round, consisting of k packets. Assume that m packets from this short round make it through. Then the sender will get m duplicate ACKs for packet f_k. If three or more duplicate ACKs are received, the sender reduces its window size by half and continues. Otherwise, it waits for a retransmit timeout and reduces its window to one. We do not model certain aspects of TCP's behavior (e.g., fast recovery) but believe we have captured the essential elements of TCP behavior, as indicated by the generally very good fits between model predictions and measurements made on numerous commercial TCP implementations, as discussed in Chapter 2.

We begin by defining several sequences of random variables that characterize the state space of the modeled system.

- $\{W_i\}_{i=0}^{\infty}, \quad 1 \leq W_i \leq W_{max}$.

  W_i is window size for round i, $i = 0, \ldots \infty$.

- $\{C_i\}_{i=0}^{\infty}, \quad C_i = 0, 1$.

  C_i allows us to model the increment of congestion window by one every two rounds during a no-loss period. $C_i = 0$ indicates the first of these two rounds and $C_i = 1$ indicates the second. We impose the following constraint on the value of C_i: If $W_i = W_{max}$, then $C_i = 0$. This captures the fact that the window size cannot grow beyond W_{max}.

- $\{L_i\}_{i=0}^{\infty}, \quad 0 \leq L_i < W_{i-1}$.

  L_i is the number of packets lost in the $(i-1)^{st}$ round, $i = 0, \ldots \infty$. $L_i = 0$ is an indication that either no packets were lost in the previous round, or the previous round was a short round and effects of packet loss have already been accounted for via a triple duplicate event or a timeout event. We define $L_0 = 0$. $L_i = 0$ indicates that this round is a short round, and is a result of L_i packets being lost in the previous round. For short rounds, C_i is always equal to 0. In other words, if $L_i > 0$, then $C_i = 0$.

- $\{T_i\}_{i=0}^{\infty}, \quad 0 \leq T_i \leq 7$.

  T_i denotes whether the connection is in a timeout state in round i, $i = 0 \ldots \infty$. $T_i = 0$ if the transmission of packets in this round was not a result of a retransmission timer expiration. In other words, $T_i = 0$ if a timeout did not occur as a result of packet loss in the previous round. Moreover, $T_i = 1, \ldots, 7$, if timeout did occur in the previous round, and the backoff value was $1, 2, 4, \ldots, 64$, respectively.

- $\{R_i\}_{i=0}^{\infty}, \quad RTT \leq R_i \leq 64 * T_0$.

  R_i denotes the duration of round i, $i = 0, \ldots \infty$. RTT denotes the round trip time, and T_0 denotes the base timeout value. During a timeout R_i can have value of $T_0, 2T_0, \ldots 64 * T_0$, as explained earlier. For all other rounds, $R_i = RTT$.

- $\{N_i\}_{i=0}^{\infty}, 1 \leq N_i \leq W_{max}$.

  N_i denotes the number of packets transmitted in round i, $i = 0, \ldots \infty$.

The sequence of random variables, $\{(W_i, C_i, L_i, T_i)\}_{i=0}^{\infty}$ is a finite state Markov chain (MC) with probability transition matrix, $\mathbf{Q} = [q_{v,a,k,s;w,c,l,t}]$, which we will soon calculate. Two states of this MC are transient, namely: $(1, 0, 0, 0)$ and $(1, 1, 0, 0)$. This is because we assume that a TCP sender starts with a window size of 1. However, once the window reaches size of 2, it is never reduced to 1 unless either a timeout occurs $(T_i > 0)$ or the previous round had size k and $k-1$ packets were lost, $(W_{i-1} = k, L_i = k-1)$. As a result, there are no transitions *to* these two states *from* the rest of the states in the MC. It can be easily seen, from the transition probability matrix defined later in this section, that the remaining states in the MC form an irreducible sub-chain, and that this subchain is aperiodic. Define set S as the set consisting of the states of this subchain. Consequently, the following limiting probabilities exist,

$$\pi_{w,c,l,t} = \lim_{i \to \infty} P(W_i = w, C_i = c, L_i = l, T_i = t), \quad (w, c, l, t) \in S$$

and they satisfy

$$\boldsymbol{\pi} = \boldsymbol{\pi} \mathbf{Q} \tag{3.1}$$

Let $\{(W_r, C_r, L_r, T_r)\}$ and $\{(W_{r'}, C_{r'}, L_{r'}, T_{r'})\}$ be two successive states in the sequence $\{(W_i, C_i, L_i, T_i)\}_{i=0}^{\infty}$.

Also, let:

$$W_r = v,\ C_r = a,\ L_r = k,\ T_r = s, \quad (v,a,k,s) \in S$$

and

$$W_{r'} = w,\ C_{r'} = c,\ L_{r'} = l,\ T_{r'} = t, \quad (w,c,l,t) \in S$$

The number of packets sent during this transition is denoted by N_r and the time taken for the transition is denoted by R_r. Of the N_r packets, M_r packets make it to the receiver. Define the conditional expectations:

$$\begin{aligned} n_{v,a,k,s;w,c,l,t} &= E[N_r | W_r = v, C_r = a, L_r = k, T_r = s, W_{r'} = w, C_{r'} = c, L_{r'} = l, T_{r'} = t] \\ r_{v,a,k,s;w,c,l,t} &= E[R_r | W_r = v, C_r = a, L_r = k, T_r = s, W_{r'} = w, C_{r'} = c, L_{r'} = l, T_{r'} = t] \end{aligned}$$

These conditional expectations, along with the transition probabilities $q_{v,a,k,s;w,c,l,t}$, capture the behavior of TCP's congestion control and avoidance algorithm. Define:

$$\begin{aligned} N &= \lim_{i \to \infty} N_i \\ R &= \lim_{i \to \infty} R_i \end{aligned}$$

For given time $t > 0$, define N_t to be the number of packets transmitted in interval $[0,t]$. Then the send rate during that interval is given by: N_t/t.

The long term, steady state send rate of a TCP connection is then defined by:

$$\begin{aligned} \text{Send rate} &= \lim_{t \to \infty} \frac{N_t}{t} \\ &= E[N]/E[R] \\ &= \frac{\sum_{(v,a,k,s) \in S} \pi_{v,a,k,s} \sum_{(w,l,c,t) \in S} q_{v,a,k,s;w,c,l,t} n_{v,a,k,s;w,c,l,t}}{\sum_{(v,a,k,s) \in S} \pi_{v,a,k,s} \sum_{(w,l,c,t) \in S} q_{v,a,k,s;w,c,l,t} r_{v,a,k,s;w,c,l,t}} \end{aligned} \tag{3.2}$$

We now calculate the transition probabilities and the conditional expectations of packets sent in each round and time required to complete the round. The loss model used to calculate these is the same as the one used in Chapter 2. We now briefly restate it. If the packet loss probability is p, then each packet in a given round may be lost with probability p, until a packet *is* lost. All the packets in the round, following the lost packet, are lost with probability 1. Losses in different rounds are independent of one another.

3.2.1 No packets are lost

When no packets are lost, the TCP window size increases by one every two round trip times. During each such round, a window's worth of packets are sent in one round trip time. Therefore:

$$
\begin{aligned}
q_{w,0,0,0;w,1,0,0} &= (1-p)^w, \quad 1 \le w < W_{max} \\
q_{w,1,0,0;w+1,0,0,0} &= (1-p)^w, \quad 1 \le w < W_{max} \\
q_{w,0,0,0;w,0,0,0} &= (1-p)^w, \quad w = W_{max} \\
\\
n_{w,0,0,0;w,1,0,0} &= w, \qquad 1 \le w < W_{max} \\
n_{w,1,0,0;w+1,0,0,0} &= w, \qquad 1 \le w < W_{max} \\
n_{w,0,0,0;w,0,0,0} &= w, \qquad w = W_{max} \\
\\
r_{w,0,0,0;w,1,0,0} &= RTT, \quad 1 \le w < W_{max} \\
r_{w,1,0,0;w+1,0,0,0} &= RTT, \quad 1 \le w < W_{max} \\
r_{w,0,0,0;w,0,0,0} &= RTT, \quad w = W_{max}
\end{aligned} \tag{3.3}
$$

3.2.2 One or more packets are lost in a round

When one or more, but not all, packets are lost in a round, the following round is a "short" round, as shown in Figure 3.2. This transition is accompanied by sending of a window worth of packets in a round trip time. When all packets in a window are

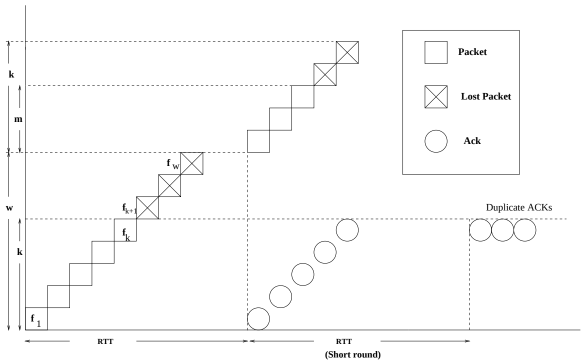


Figure 3.2. Packet and ACK transmissions preceding a loss indication

lost, there is no following "short" round, instead, the sender waits for a retransmission timeout, and the window size goes to one.

$$
\begin{aligned}
q_{w,c,0,0;w-l,0,l,0} &= p(1-p)^{w-l}, && 2 \le w \le W_{max}, \quad c=0,1, \quad 1 \le l < w \\
q_{w,c,0,0;1,0,0,1} &= p, && 1 \le w \le W_{max}, \quad c=0,1 \\
\\
n_{w,c,0,0;w-l,0,l,0} &= w, && 2 \le w \le W_{max}, \quad c=0,1, \quad 1 \le l < w \\
n_{w,c,0,0;1,0,0,1} &= w, && 1 \le w \le W_{max}, \quad c=0,1 \\
\\
r_{w,c,0,0;w-l,0,l,0} &= RTT, && 2 \le w \le W_{max}, \quad c=0,1, \quad 1 \le l < w \\
r_{w,c,0,0;1,0,0,1} &= TO, && 1 \le w \le W_{max}, \quad c=0,1
\end{aligned}
\tag{3.4}
$$

3.2.3 One or more packets are lost in a short round

The loss of one or more packets in a short round affects the number of duplicate ACKs received by the sender, as shown in Figure 3.2. If the number of packets that make it to the receiver is greater than two, the sender will get at least three duplicate ACKs and it will reduce its window size to half. Otherwise, it will wait for a timeout, and reduce its window size to one. Note that if the short round consists of less than three packets, ($w < 3$), then timeout will always occur. Note that for all the equations in (3.5), $0 < l < W_{max}$, and $1 < w + l \leq W_{max}$.

$$
\begin{array}{lcll}
q_{w,0,l,0;1,0,0,1} & = & 1, & 1 \leq w < 3 \\
q_{w,0,l,0;1,0,0,1} & = & \sum_{i=0}^{2} p(1-p)^i, & 3 \leq w < W_{max} \\
q_{w,0,l,0;\lfloor (w+l)/2 \rfloor,0,0,0} & = & \sum_{i=3}^{w-1} p(1-p)^i + (1-p)^w, & 3 \leq w < W_{max} \\
 & & & \\
n_{w,0,l,0;1,0,0,1} & = & w, & 1 \leq w < 3 \\
n_{w,0,l,0;1,0,0,1} & = & w, & 3 \leq w + l < W_{max} \\
n_{w,0,l,0;\lfloor (w+l)/2 \rfloor,0,0,0} & = & w, & 3 \leq w < W_{max} \\
 & & & \\
r_{w,0,l,0;1,0,0,1} & = & T_0 - RTT, & 1 \leq w < 3 \\
r_{w,0,l,0;1,0,0,1} & = & T_0 - RTT, & 3 \leq w < W_{max} \\
r_{w,0,l,0;\lfloor (w+l)/2 \rfloor,0,0,0} & = & RTT, & 3 \leq w < W_{max}
\end{array}
\tag{3.5}
$$

3.2.4 Exponential backoff

If a packet is retransmitted as a result of a timeout, the retransmission timer backs off exponentially. The maximum duration of the backoff timer is limited to 64 times the base timeout value. The timeout sequence ends when the retransmitted packet is successfully ACKed.

$$
\begin{array}{lcll}
q_{1,0,0,i;1,0,0,\min(i+1,7)} & = & p, & 1 \leq i \leq 7 \\
q_{1,0,0,i;2,0,0,0} & = & 1-p, & 1 \leq i \leq 7 \\
 & & & \\
n_{1,0,0,i;1,0,0,\min(i+1,7)} & = & 1, & 1 \leq i \leq 7 \\
n_{1,0,0,i;2,0,0,0} & = & 1, & 1 \leq i \leq 7 \\
 & & & \\
r_{1,0,0,i;1,0,0,\min(i+1,7)} & = & 2^{(i-1)} T_0, & 1 \leq i \leq 7 \\
r_{1,0,0,i;2,0,0,0} & = & RTT, & 1 \leq i \leq 7
\end{array}
\tag{3.6}
$$

3.2.5 Completing the transition matrix

Define Q' as the set of all elements of $\mathbf{Q}$ that have been defined in (3.3)-(3.6). Then:

$$q_{v,a,k,s;w,c,l,t} \quad = \quad 0, \quad (v,a,k,s),(w,c,l,t) \in S \quad \text{and } q_{v,a,k,s;w,c,l,t} \notin Q' \tag{3.7}$$

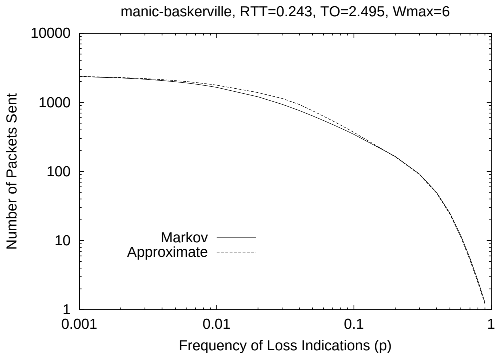
This completes the formal description of our model. In the next section, we present a comparison between this model and the model described in Chapter 2.

3.3 Comparison of the two models

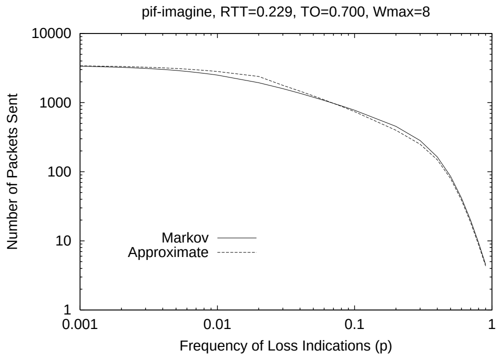
We can numerically compute the send rate predicted by the Markovian model, using (3.1), (3.2) and (3.3)-(3.7). We find that the predictions of the model match well with the simple model proposed in Chapter 2. The closeness of the match validates the approximate model. We have verified the closeness of the match over a wide range of values for RTT, p, T_0 and W_{max}. Here, we only show a representative example in Figure 3.3. These graphs correspond to the graphs in Figure 2.7. The closeness of the match between two the models is evident.

3.4 Conclusion

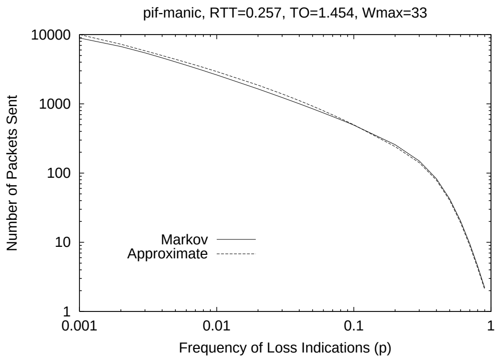
In this chapter we presented a detailed Markov model of TCP-Reno congestion control. We built this model without making many of the simplifying assumptions made while building the model described in Chapter 2. Although the Markov model does not have appear to have a closed form solution. However, numerical solutions of the model match the simple model presented in Chapter 2. This detailed model has recently been extended [61] to investigate behavior of TCP under DiffServ environments, and has been shown there to be valid over a wide range of scenarios.



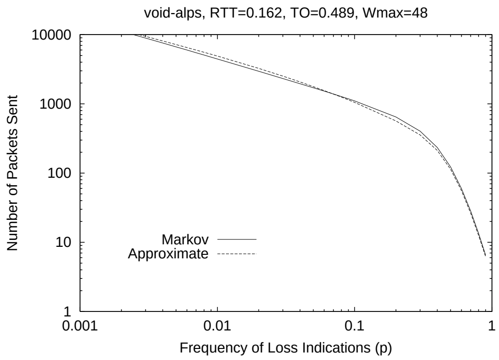
(a) manic to baskerville



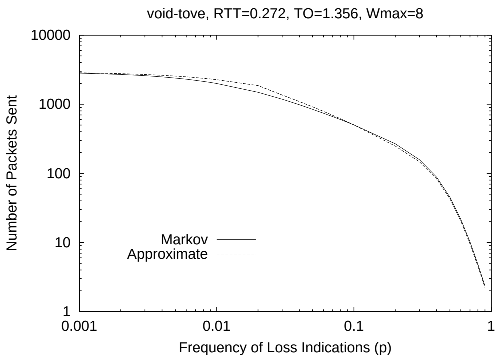
(b) pif to imagine



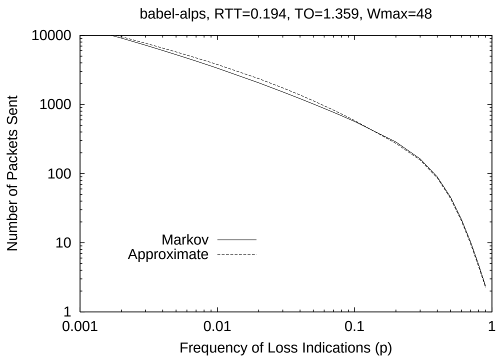
(c) pif to manic



(d) void to alps



(e) void to tove



(f) babel to alps

Figure 3.3. Comparison of the two models

CHAPTER 4

A SIMPLE TCP-FRIENDLY PROTOCOL

4.1 Introduction

Networked multimedia applications usually employ non-TCP protocols (usually UDP with some application-level control) to transmit continuous media (CM) data such as audio and video. Many current CM applications simply transmit data at the rate at which it was encoded [69]. In other words, these applications do not employ any congestion control mechanisms. It has been show in [12] that such non-responsive traffic can lead to congestion collapse of the global Internet. Thus, CM applications must implement a congestion control mechanism.

Traditional applications such as FTP run on top of TCP. As we described in Chapters 2 and 3 TCP combines reliable delivery with several congestion control mechanisms. The basic TCP congestion control mechanism is to reduce the sending rate in half in response to a congestion indication. This type of congestion control is appropriate for bulk data transfer applications such as FTP, where the average value of the throughput is the only performance metric of interest.

CM applications play out data in real-time. It has been shown in [68] that abrupt fluctuations in the throughput noticeably reduce the user-perceived quality of playout. Thus, for CM applications, fluctuations in the throughput are as important a concern as the average value. TCP congestion control responds to every congestion indication by cutting its sending rate in half. These abrupt and large changes in the sending rate have been a key impediment to the deployment of TCP's end-to-end congestion control by emerging CM applications.

CM applications must employ a congestion control algorithm to prevent congestion collapse of the Internet. This algorithm must not have an abrupt response to congestion like that of TCP. We believe that a viable congestion control mechanism for CM traffic can be designed by updating the sending rate as a function of recent loss rate. This is in contrast to TCP, which responds to every loss event. In this chapter and the next, we plan to explore "equation-based" congestion control as a candidate for such a mechanism.

Equation-based congestion control was first proposed in [32]. Whereas TCP congestion control backs off in response to a single congestion indication, equation-based congestion control uses a control equation that explicitly gives the maximum acceptable sending rate as a function of the recent loss rate. The sender adapts its sending rate, guided by this control function, in response to feedback from the receiver. For traffic that competes in the best-effort Internet with TCP, the appropriate control equation for equation-based congestion control is the TCP response function. This function characterizes the steady-state sending rate of TCP as a function of the round-trip time and steady-state loss event rate. We have presented such a characterization of TCP send rate in Chapter 2. An application wishing to send less than the TCP-compatible sending rate (e.g., because of limited demand) would still be characterized as TCP-compatible. However, if a significantly less aggressive response function were used, then the less aggressive traffic could suffer in terms of average throughput competing with TCP traffic.

For traffic that is not competing with TCP traffic, but is isolated from TCP traffic by some method (e.g., with per-flow scheduling, or in a separate differentiated-services class from TCP traffic), applications using equation-based congestion control could make a different choice for the underlying control equation. Issues about the merits or shortcomings of various control equations for equation-based congestion control are an active research area that we do not address further in this thesis.

In this chapter we propose a simple equation-based congestion control protocol based on the TCP model developed in Chapter 2. We name this protocol "Simple TCP Friendly Rate Control" (STFRC) protocol. We evaluate the protocol under various traffic conditions, using simulations and implementation. Our experimental and simulation results show that the protocol is "fair" to TCP (we make this notion precise in later sections of the chapter) and to other sessions running STFRC, and that the model-based approach to achieving TCP-friendliness is indeed practical. Our experimental results also point our various shortcomings of the proposed simple protocol. We refine our protocol further in Chapter 5.

The rest of this chapter is organized as follows. In Section 4.2, we present an overview of related work reported in the literature. Section 4.3 describes the STFRC protocol. In Section 4.4, we present simulation studies of our protocol. Section 4.5 presents results from a "real-world" implementation of the protocol. In Section 4.6 we discuss some of our design choices. Section 4.7 concludes the chapter.

4.2 Related work

Several TCP-friendly rate adjustment protocols have recently been reported in the literature [19, 42, 56, 63, 68, 71, 72]. We now briefly review each of them.

The unreliable unicast congestion control mechanisms closest to TCP maintain a congestion window which is used directly [19] or indirectly [42] to control the transmission of new packets. In [19] authors propose a protocol in which the sender manages its window size in exactly the same way as TCP does, but instead of retransmitting lost packets, it allows the user to send new data in each packet. We believe that since [19] uses TCP mechanisms directly, it is not suitable for CM applications, due to the reasons mentioned in the previous section. In the TEAR protocol (TCP Emulation at the Receivers) from [42], which can be used for either unicast or multicast sessions, the receiver emulates the congestion window modifications of a TCP sender, but then makes a translation from a window-based to a rate-based congestion control mechanism. The receiver maintains an exponentially weighted moving average of the congestion window, and divides this by the estimated round-trip time to obtain a TCP-friendly sending rate. At the present time, we do not have access to sufficient information about TEAR to allow us to comment on its comparative merits.

A class of unicast congestion control mechanisms one step removed from those of TCP are those that use additive increase, multiplicative decrease (AIMD) in some form, but do not apply AIMD to a congestion window. The Rate Adaptation Protocol (RAP) [56] uses an AIMD rate control scheme based on regular acknowledgments sent by the receiver that the sender uses to detect lost packets and estimate the RTT. The authors use the ratio of long-term and short-term averages of the RTT to fine-tune the sending rate on a per-packet basis. This translation from a window-based to a rate-based approach also includes a mechanism for the sender to stop sending in the absence of feedback from the receiver.

In [63] the authors propose an AIMD scheme that is suitable mainly for unicast applications, but may be modified for multicast applications. The scheme relies on regular RTP/RTCP reports [62] sent between the sender and the receiver to estimate the loss rate and round trip times. In addition, they propose modifications to RTP that allow the protocol to estimate the bottleneck link bandwidth using the packet-pair technique proposed in [4]. An additive increase/multiplicative decrease scheme based on these three estimates (loss rate, round trip delay, and bottleneck bandwidth) is then used to control the sending rate.

Pure AIMD protocols like RAP do not account for the impact of retransmission timeouts, and hence we believe that STFRC will coexist better with TCP in the regime where the impact of timeouts is significant. In [34] it has been shown that the impact of timeouts on TCP throughput is significant for loss rates higher than 5%.

Equation-based congestion control [32] is probably the class of TCP-compatible unicast congestion control mechanisms most removed from the AIMD mechanisms of TCP. In unicast equation-based congestion control the sender uses an equation such as those proposed in [32] or in Chapter 2, that specifies the allowed sending rate as a function of the RTT and packet drop rate, and adjusts its sending rate as a function of those measured parameters.

In [68] the authors describe a simple equation-based congestion control mechanism for unicast, unreliable video traffic. The receiver measures the RTT and the loss rate over a fixed multiple of the RTT. The sender then uses this information, along with the version of the TCP response function from [32], to control the sending rate and the output rate of the associated MPEG encoder. The main focus of [68] is not the congestion control mechanism itself, but the coupling between congestion control and error-resilient scalable video compression.

TCP-Friendly multicast protocols have been proposed in [71, 72]. In [71] the authors propose a multicast congestion control scheme in which the data is transmitted in a layered manner over different multicast groups. The more layers a receiver joins, the more data it receives. In [71] the receivers compute round trip times and estimate the packet loss rate p, and use the formula proposed in [32] to compute the TCP-friendly rate at which they should receive the data. Based on this estimate, and the knowledge of the layering schemes, each receiver can dynamically decide to join or leave certain multicast groups to adjust the rate at which it receives the data. In [72], the authors propose a similar scheme in which the layers have data rates that are fixed multiples of a base rate, and a TCP-like effect (additive increase, multiplicative decrease) is achieved by using strict time limits on when a receiver might join or leave a group. The analysis of the algorithm yields characterization for the send rate that is similar to one proposed in [32].

In the next section we propose a new protocol that uses the TCP model described in Chapter 2 and achieves TCP friendliness in a more real world scenario that includes competing TCP-Reno connections, drop-tail switches and diverse background traffic conditions.

4.3 The STFRC protocol

The simple TCP friendly rate control (STFRC) protocol is a rate-adjustment congestion control protocol that is based on the TCP characterization proposed in Chapter 2. Unlike [32,34,41], the characterization in Chapter 2 takes into account the effects of timeouts, a consideration that is particularly important when TCP-Reno (one of the most widely deployed versions of TCP) is used with drop-tail routers, which tend to produce correlated losses. If a TCP-Reno connection encounters correlated losses, it tends to experience a significant number of timeouts [9]. In Chapter 2 we quantified this phenomenon and its effects on send rate. The resulting analytic characterization of TCP send rate was given in Equation (2.33). We restate it here:

$$\text{Send rate} \approx \min\left(\frac{W_{max}}{R}, \frac{1}{R\sqrt{\frac{4p}{3}} + B\min\left(1, 3\sqrt{\frac{3p}{4}}\right)p(1+32p^2)}\right) \tag{4.1}$$

where the send rate is measured in packets per unit time, W_{max} is the receiver's declared window size, R is the round trip time experienced by the connection, p is the loss rate (or, more accurately, the frequency of loss indications per packet sent) and B is the base timeout value [64].

There are two parts to the STFRC protocol: a sender-side protocol and a receiver-side protocol. The sender-side protocol works in *rounds* of duration M time units. We call M the *recomputation interval.* At the beginning of each round, the sender computes a TCP-friendly rate (we will shortly describe this computation in detail), and sends packets at that rate, as discussed below. Each packet carries a sequence number and a timestamp indicating the time the packet was sent. The receiver acknowledges each packet, by sending an ACK that carries the sequence number and timestamp of the packet it is acknowledging. Consider an ACK for a packet whose sequence number is k. In addition to the sequence number and the timestamp, the ACK also carries a bit vector of 8 bits indicating whether or not each of the previous 8 packets $(k-7\ldots k)$ was received. The sender processes these ACKs to compute sending rate for the next round. Note that each packet is ACKed eight times, providing some protection against ACK losses.

Let us now consider the sending rate computation in detail. Consider round i. Let r_i be the sending rate for this round, R be the the current round trip time estimate, and B be the estimate of the base timeout value. The number of packets to be sent in this round is $n_i = r_i * M$. The n_i packets are clocked out uniformly during the round. As noted earlier, packets carry a sequence number and a timestamp indicating the time the packet was sent. The sender keeps a log of all packets it has sent in this round. The log contains two entries for each packet. The first entry indicates whether the packet has been *(i)* received and has been acknowledged by the receiver; *(ii)* presumed lost; *(iii)* of unknown status (neither ACKed nor yet presumed lost). We call this the "received status" of the packet. The second entry consists of a value that is equal to the time the packet was sent plus the current base timeout value. We call this the "timeout limit" for the packet.

As the sender sends packets, it also receives ACKs from the receiver. Consider an ACK carrying sequence number k that is received by the sender at time t_k. Let the timestamp carried by the ACK be s_k. The sender updates the lost/received status of packets $(k-7\ldots k)$ using the bit vector available in the ACK. The sender also updates the round trip time estimate (R) and base timeout (B) using the difference $t_k - s_k$. This update is done exactly as in TCP; see [65] for the details of the computation. At the end of the i^{th} round, the sender computes r_{i+1} as follows:

Let the current time be t_i. Let j be the packet with the smallest sequence number, whose received status was "unknown" at the end of round $i-1$, l be the last packet sent and a be the highest sequence number for which we have received an ACK. Then any packet whose sequence number lies between j and l, (both included) and whose timeout limit is less than t_i, is marked as lost. Also, any packet whose sequence number lies between j and a (both included), and whose received status is "unknown" is marked as lost. Let x_i be the number of packets marked as "received" between j and a, and let y_i be the number of packets marked as "lost" between j and a. Then:

- If $y_i = 0$, then no packets were lost and:

$$r_{i+1} = 2 * r_i$$

  Hence, when no packets are lost in a round, packets are sent twice as fast in the next round. We will discuss this feature more in Section 4.6.

- Otherwise, $y_i \neq 0$. Let $p_i = \frac{y_i}{x_i + y_i}$. In this case, the rate for round $i+1$ is

$$r_{i+1} = f(W_{max}, R, p_i, B)$$

  where f is defined in (4.1).

The starting value r_0, can be set to any reasonable value. We have found that for sufficiently long flows, and for reasonable values of M, the value of r_0 has little impact on the performance of the protocol. For all simulations and experiments described in this chapter, we set this value to 40 packets/second. The initial values of R and B are set in a manner similar to TCP [65].

STFRC has no built-in error recovery mechanisms. When a comprehensive congestion control protocol, based on STFRC is developed, the applications will be able to choose an error control strategy that is appropriate for the given media type. An important feature of any transmission control protocol is "self-limitation" [56]. This means that if the protocol starts experiencing 100% or near 100% losses, its sending rate should be reduced to almost zero. TCP achieves this via timeouts and eventual closedown of the connection. The STFRC protocol uses the model proposed in Chapter 2, which takes into account the effect of timeouts and automatically reduces the sending rate to very small values at high loss rates.

The key question is how frequently the sender should re-compute the rate, i.e., how to determine the value of M. In the following section we use simulations to explore various strategies for choosing M, and their impact on the performance of the protocol.

4.4 Simulation results

In this section we present simulation studies of the STRFC protocol. The simulations are used to study the behavior of the protocol under controlled conditions. In the following section then we present additional studies carried out over the Internet. We have used the `ns` simulator [10] for our simulations. There are two main challenges for any simulation study of this nature: first, how to select appropriate network topologies, how to effectively model the background traffic and how to define and measure appropriate performance metrics. Several difficulties in this regard are pointed out in [50]. Thus, before we present any simulation results, we discuss our simulation topology and our performance metrics.

4.4.1 Simulation topology

In our simulations, we use a simple topology to uncover and illuminate the important issues; our experiments with STRFC over the Internet test its use in "real-world" scenarios. The simulated network topology assumes a single shared bottleneck link,

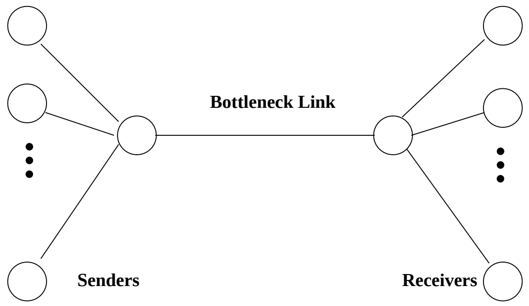


Figure 4.1. Simulation topology

as shown in Figure 4.1. The sources are arranged on one end of the link and the receivers on the other side. All links except the bottleneck link are sufficiently provisioned to ensure that any drops/delays that occur are only due to congestion at the bottleneck link. All links are drop-tail links. Many previous studies [9, 14, 56, 63] have used similar topologies.

The problem of accurately modeling background traffic is more difficult. We present results using two types of background traffic: infinite-duration FTP-like connections and self-similar UDP traffic. The infinite-duration FTP connections allow us to study the steady-state behavior of our protocol, while the self-similar UDP traffic is believed to be a good model for short TCP connections such as those resulting from web traffic [47, 74].

When multiple TCP connections are simulated over a single bottleneck link, the connections can become synchronized. We take two measures to prevent such synchronization. First, we start the connections at slightly different times. Second, before each packet is sent out, a small random delay is added to simulate processing overhead. These measures are applied to both TCP and STRFC connections.

4.4.2 Performance metrics

Recall that we view STRFC protocol as only a *first step* towards developing a comprehensive congestion control protocol for CM data flows. Thus, we are only interested in measuring the "TCP-friendliness" of the STRFC protocol. We define the "friendliness" metric as follows. Let k_c denote the total number of monitored STRFC connections and k_t denote the total number of monitored TCP connections. We denote the send rate of the k_c STRFC connections by $T_1^c, T_2^c, \ldots T_{k_c}^c$ and that of the TCP connections by $T_1^t, T_2^t, \ldots T_{k_t}^t$ respectively. Define:

$$T_C = \frac{\sum_{i=1}^{k_c} T_i^c}{k_c} \qquad \text{and} \qquad T_T = \frac{\sum_{i=1}^{k_t} T_i^t}{k_t}$$

The performance metric of interest is the "friendliness ratio", F:

$$F = T_C/T_T$$

Another metric for measuring performance is the "equivalence ratio", E:

$$E = \max(T_T/T_C, T_C/T_T)$$

Note that the value of E is always ≥ 1. E gives a better visual representation of the closeness of the send rates achieved by the two protocols. However, this metric will distort any trend that might be present in the ratio of the two send rates as we vary various parameters. For example, a decreasing value of F as a function of some system parameter will not always result in a decreasing value of E. Thus, we use F as the fairness metric whenever we are interested in *trends*, and use E otherwise. It is also important that the STRFC connections achieve fairness amongst themselves. We define the ratio:

$$FC = \frac{\max_{1\leq i\leq k_c} T_i^c}{\min_{1\leq i\leq k_c} T_i^c}$$

to characterize the fairness achieved among the STRFC connections.

4.4.3 Simulation scenarios

We now present results of performance evaluation of the STRFC protocol in various simulation scenarios.

4.4.3.1 Long duration flows

In this scenario we consider traffic made up entirely of equal numbers of infinite-duration TCP connections and infinite-duration STRFC connections. All connections always have data to send. All connections "start" at the beginning of the simulation (subject to the slight variation in starting times introduced to avoid synchronization) and last until the simulation ends. The aim here is to study steady state behavior of the STRFC protocol. If STRFC performs well (i.e., in a TCP-friendly manner), the TCP and STRFC connections should see approximately the same send rate.

We vary the total number of flows in the network between 10 and 50. Half of these connections are TCP connections and the rest are STRFC connections. The initial sending rate, r_0, for all STRFC connections was set to approximately 40 packets/second. The bottleneck bandwidth is held constant at 1.5Mbps, and the bottleneck delay is set to 50ms. This roughly simulates a situation in which a number of connections share a T1 link. As the number of flows grows, the window sizes of individual TCP connections shrink, increasing the probability of timeouts. In such circumstances, the congestion control protocols proposed in [56, 63] are not be able to guarantee fairness, as explained in Section 4.2.

We consider three different ways to determine the manner in which STRFC recomputes its rate:

- Fixed recomputation interval, i.e., we use a fixed value for M. We call this strategy S1.

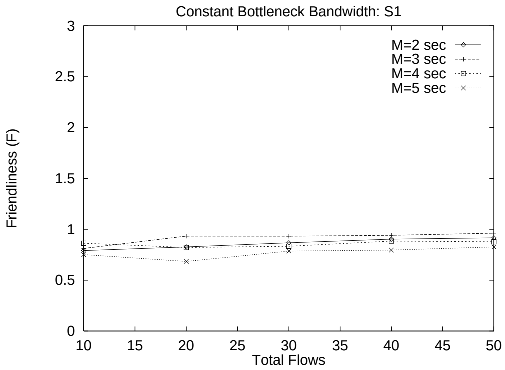
- The recomputation interval is a fixed multiple of round trip time. If at the beginning of round i the round trip time is rtt_i, then the next recomputation is performed after $K * rtt_i$ time units, where K is constant. We call this strategy S2.

- The recomputation interval is calculated at the beginning of each round, and is set to sum of two numbers, one of which is a constant while the other is chosen from a uniform random distribution. This strategy will further prevent STRFC connections from synchronizing with each other. We call this strategy S3.

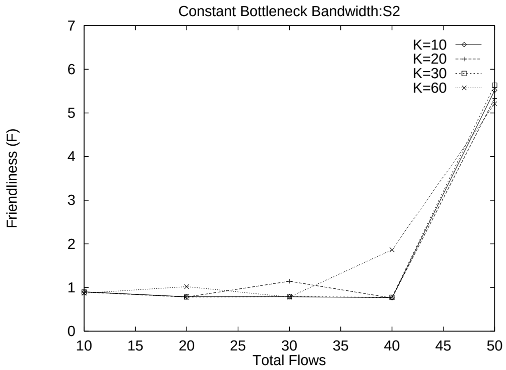
In Figure 4.2(a) we present simulation results for the case in which the STRFC protocol uses strategy S1, with five values of M between 2 and 5 seconds. The length of each simulation was 1000 seconds, and the send rate of all connections was measured at the end of the simulation. Each data point is an average of three experiments. It can be seen that with steady state background traffic, the protocol is able to maintain a friendliness ratio close to 1.

In Figure 4.2(b) we present simulation results when STRFC protocol uses strategy S2, with four values of K between 10 and 60. We notice that as the load on the network increases, the resulting STRFC behavior is more aggressive than TCP. As the load on the network increases, the round trip time experienced by each flow also increases. As a result, each STRFC flow re-computes its rate less frequently. TCP reduces its transmission rate multiplicatively every time it encounters a loss, and increases it only additively in case of no loss. The slowness of response of STRFC flows to react to losses results in it maintaining a higher send rate while the TCP connections back off their sending rate. Thus STRFC is more aggressive, and clearly S2 is not an appropriate strategy for deciding recomputation intervals.

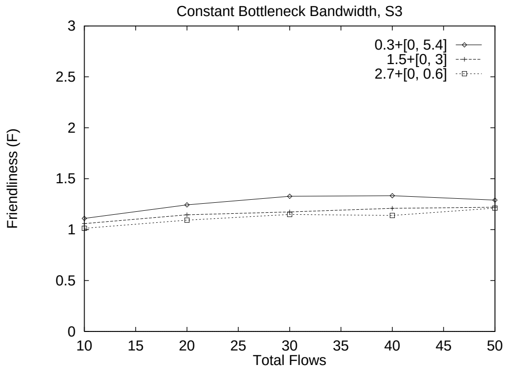
In Figure 4.2(c) we present simulation results where STRFC protocol uses strategy S3. For each line we use a different constant and a different uniform random



(a) Strategy S1



(b) Strategy S2



(c) Strategy S3

Figure 4.2. Three strategies for recomputation interval

distribution: $0.3 + [0, 5.4], 1.5 + [0, 3]$ and $2.7 + [0, 0.6]$. For this simulation study, all STRFC connections were started simultaneously. It can be seen that in this third case the protocol is able to maintain a friendliness ratio close to 1.

We have performed simulations with other bottleneck delays and observed similar results. In the rest of this section we only present results using strategy S1. We do this for two reasons. First, strategy S1 is the simplest strategy. The goal of this chapter is to present the STRFC protocol as a baseline policy; use of a simple policy to decide the recomputation interval is consistent with that goal. Second, the question of selecting the appropriate recomputation interval requires more complex answers

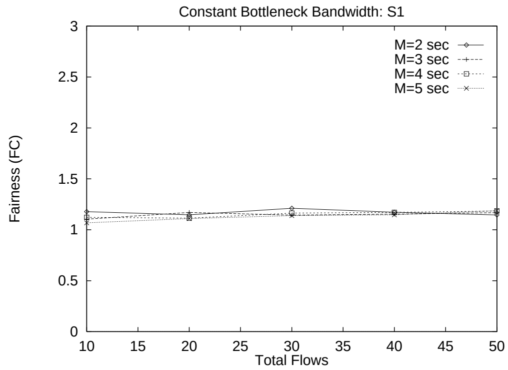


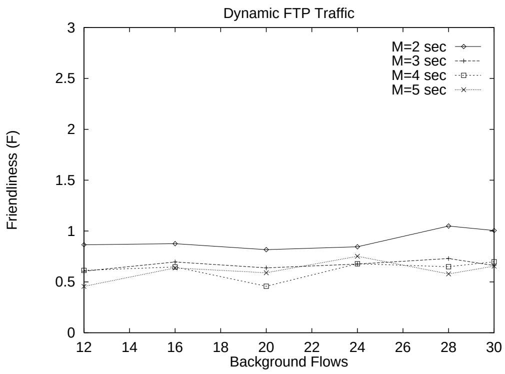
Figure 4.3. Friendliness among STRFC connections

than the three simple strategies described here. The recomputation interval must be short enough to allow STRFC to be responsive, while at the same time it must be large enough to allow the loss rate measurements to be meaningful. This question will be further addressed in Chapter 5. For our purposes it is thus appropriate to restrict the baseline protocol described here to the simplest strategy.

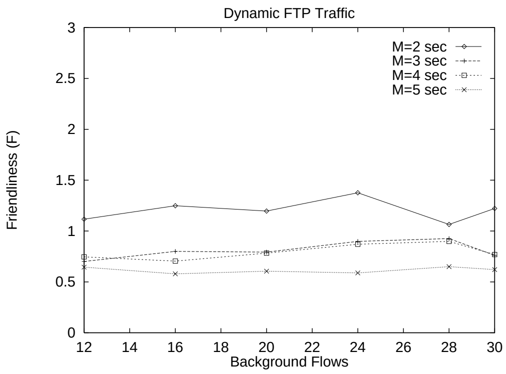
Recall that the STRFC connections should be fair to each other as well. In Figure 4.3 we plot the value of FC when the STRFC protocol uses strategy S1. It can be seen that the STRFC protocol achieves acceptable fairness among STRFC connections in most cases.

4.4.3.2 ON/OFF UDP traffic

In this simulation scenario, we model the effects of competing web-like traffic (very small TCP connections, some UDP flows). It has been reported in [47] that WWW-related traffic tends to be self-similar in nature. In [74], it is shown that self-similar traffic may be created by using several ON/OFF UDP sources whose ON/OFF times are drawn from heavy-tailed distributions such as the Pareto distribution. Figure 4.5 presents results from simulations in which the "shape" parameter of the Pareto distribution is set to 1.2. The mean ON time is 1 second and the mean OFF time is 2



(a) Average transfer 40KB



(b) Average transfer 80KB

Figure 4.4. Friendliness (F), dynamically arriving FTP connections

seconds. During ON times the sources transmit with a rate of 12Kbps. The number of simultaneous connections is varied between 20 and 80. The simulation was run for 25000 seconds. As in the previous subsection, there are two monitored connections, an infinite TCP connection and an infinite STRFC connection (i.e. $k = 2$). The bottleneck link bandwidth is set to 1.5Mbps and the bottleneck delay is set to 50ms. From the results in Figure 4.5, we can see that the STRFC protocol is still relatively fair. The fairness index again decreases as the recomputation interval, M, increases. We believe that this is due to the fact that the STRFC connection recomputes its rate only after every M time units. Hence, it can not increase its sending rate during the small periods of time in which the background traffic drops in intensity. We repeated these simulations by setting the value of the shape parameter to 1.5 and 1.9, and observed similar results.

4.4.4 Summary of simulation results

The simulations results presented in this section show that the STRFC protocol is TCP-friendly under a wide range of traffic conditions. We found that the strategy of using a fixed value for recomputation interval (M), works well for the simulation

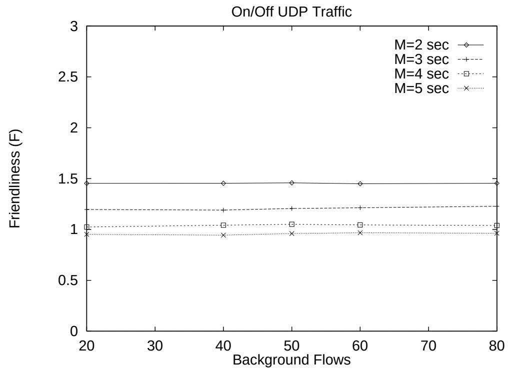


Figure 4.5. Friendliness(F), ON-OFF UDP Traffic, Shape = 1.2

scenarios we have considered. While the simulation study indicates that the protocol performs well, it is important to observe the behavior of the protocol in the real world. In the next section we discuss the implementation and present results based on experiments carried out over the Internet.

4.5 Implementation and experimental results

As noted in [50], simulating an Internet-like environment is very difficult. It is thus essential to test protocols like STFRC via implementation and experimentation in a real-world setting. Our goal here is thus to show that the approach is practical, and that protocol performance under real-world conditions is comparable to that observed in the simulations. We have implemented a prototype version of our protocol and tested it on several Unix systems. In this section, we first describe the implementation, and discuss some of the difficulties encountered. We then present the results from over 300 experiments performed using this implementation.

4.5.1 Implementation

Our implementation of STFRC is done in user space, on top of UDP. The sender side of STFRC runs as two processes, one sending the data and the other receiving

ACKs. The two processes communicate via shared memory. An earlier attempt to implement the protocol using multiple threads failed, as the *p-threads* package could not provide sufficiently accurate scheduling control to avoid starving either the sender or the receiver thread. We were able to reuse much of the `ns` simulation code for the actual implementation. However, we encountered three important problems during the implementation:

- A significant problem in any actual implementation is the accuracy of the various timers involved. For simulation purposes, we could time out packets with arbitrary precision. This is not possible in an actual implementation, as timers are neither arbitrarily accurate nor are they free of overheads. In some of our early experiments, we found that when using the `gettimeofday` and `select` system calls, we could not control inter-packet interval more accurately than within several milliseconds. While we could achieve better accuracy using busy waiting in the process that sent the packets out, this can possibly starve the process that receives ACKs. On a FreeBSD machine used in this study, busy waiting caused other problems that forced us to use `gettimeofday` and `select` system call in our FreeBSD implementation. Due to these difficulties, it is not possible to clock packets out smoothly over the duration of each round. Instead, in our implementation, we send packets out in small bursts. Consider round i. Let R be the round trip time estimate at the beginning of this round, and r_i be the sending rate. The duration of the round is M time units. We divide each round into bursts of duration R. The number of bursts is thus, $b_i = M/R$. In each burst, n_i/b_i (rounded to nearest integer) packets are sent back-to-back, followed by a silence period of R time units.

- Another important problem is the accuracy of the round trip times. The STFRC protocol begins measuring the round trip time for each packet as soon as it is handed to the kernel socket using `sendto`. Thus, our round trip times include the time each packet spent waiting in the kernel buffers (similarly for ACKs). Thus, our estimate of round trip times is higher than the estimate made by the in-kernel TCP. In addition, due to additional difficulties with timers, we had to restrict the protocol to transmit *at least* one packet per round trip time.

- In our simulation studies, it was easy to ensure that the packet sizes for TCP and STFRC connections were the same. It is more complicated to ensure this in practice. Our implementation currently does not employ any path MTU discovery algorithm, nor does it change the size of outgoing packets. For each experiment described in the next section, the packet size is held constant, determined by the MTU discovered by TCP in previous experiments between the same two hosts. While we have found that we seldom had problems with the MTU value, it is hard to quantify the effects of constant packet size on send rate.

As a result of the implementation considerations noted above, we expect the results from implementation experiments to differ somewhat from the simulation experiments. However, we can still use the implementation to corroborate the intuition gained through our simulations, to examine STFRC performance in real-world setting, and to provide a starting point for a more refined implementation.

4.5.2 Experimental results

The hostnames, domains and operating systems of the machines used for the implementation study are listed in Table 4.1. To measure the fairness of STFRC compared to TCP, we performed the following experiment. We established two connections between a pair of hosts. One of these connections was controlled by the STFRC protocol, while the other was controlled by the TCP protocol. Both connections ran simultaneously, and transferred data for 1000 seconds, as fast as possible.

Based on the results we observed in our simulation studies, we fixed the length of the recomputation interval, M, for the STFRC connection was set to 3 seconds. The receiver's declared window size, W_{max}, was set to 100 packets; and the initial sending rate, r_0, was set to approximately 40 packets/second. The send rate of the two connections was measured in terms of number of packets transferred in these 1000 seconds. Let us denote these send rates T_C and T_T respectively.

Figures 4.6(a)-4.6(c) show the results when the senders were *void*, *manic* and *bmt* respectively. Since we are not interested in trends along the x-axis, we use E as our performance metric. For each of the first five bars, the x-axis shows the receiver. To plot this graph, at least 15 experiments were performed between the sender and the receiver at random times during the day and night[1], and for each experiment the value of E was computed. It is suggested in [22] that data from such experiments should be represented by its median, and that the variation be represented by the semi-inter quartile range (SIQR), defined as half of the difference between the 25^{th} and the 75^{th} percentiles of the data set. Thus, the height of each bar is the median of that data set, while the the bar represents the SIQR, centered about the median. The last bar represents the median and the SIQR of all experiments.

It can be seen that in most cases, the STFRC protocol achieves a send rate that is within 35-50% of the TCP send rate and that the difference seldom exceeds 75%. The median of all three data sets taken together is 1.448 and the SIQR is 0.275. There are many possible reasons for the observed difference between the TCP and STFRC send rates. Some variation is unavoidable – we have found that the send rate of two simultaneous TCP connections between the same hosts can differ by as much as 10%. Additional variation results from the various implementation difficulties described

[1]Experiments with *bmt* as a sender were performed only during the day.

earlier. Finally, one must remember that the formula described in Chapter 2 is only an approximation.

Figures 4.6(a)-4.6(c) are based on send rates that have been computed over the entire duration of the experiment (i.e., 1000 seconds). It is also interesting to compare the difference in TCP and STFRC as a function of time, and over shorter intervals of time. Such a comparison illustrates how well the STFRC protocol performs at various time scales. In Figure 4.7, we plot the send rate of the STFRC and the TCP connections between *manic* and *edgar*, measured every 6, 12, 24 and 48 seconds respectively. It can be seen that STFRC tracks variations in send rate of the TCP connection quite well, at various time scales.

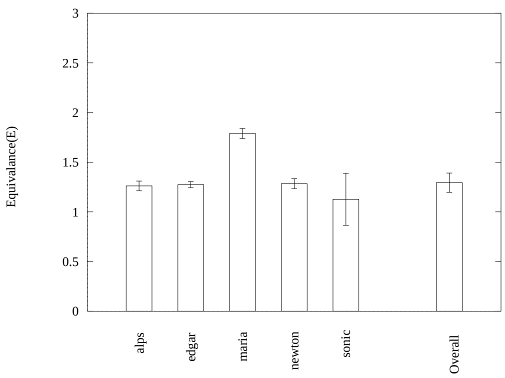
To measure the sensitivity of the protocol to the interval over which we measure the loss rate and update the sending rate (i.e. the value of M), we performed several data transfers between the same sender-receiver pair, using different measurement intervals. We now use F as our fairness metric, as we are interested in the trend in the performance metric as we vary M. In Figure 4.8 we show the results of one such study, performed between *void* and *alps*. The measurement interval was varied between between 2 and 10 seconds. For each value of measurement interval, at least 20 experiment were conducted at random times. The data points are the medians of the send rate ratios, and the error bars represent the SIQR. One can see that as the measurement interval grows larger, the STFRC protocol becomes less aggressive. From the results presented in this section, we can conclude that the protocol indeed performs well in a real world setting, despite the limitations and difficulties imposed by various implementation issues.

4.6 Discussion of protocol features

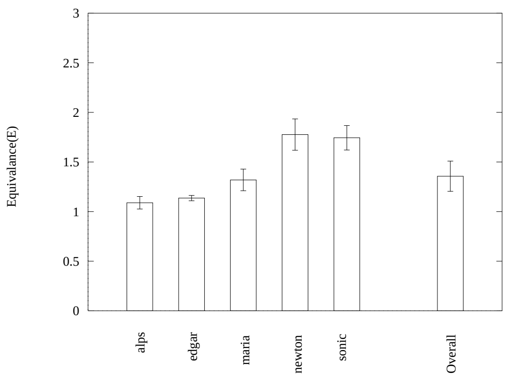
In this section we discuss the impact of some of the design choices made while simulating and implementing STFRC. Recall that STFRC doubles its sending rate

Hostname	Domain	Operating System
alps	cc.gatech.edu	SunOS 4.1.3
bmt	cs.columbia.edu	FreeBSD 2.2.7
edgar	cs.washinton.edu	OSF1 3.2
manic	cs.umass.edu	IRIX 6.2
maria	wustl.edu	SunOS 4.1.3
newton	nokia-boston.com	SunOS 5.5.1
sonic	cs.ucl.ac.uk	SunOS 5.5.1
void	cs.umass.edu	Linux 2.0.30

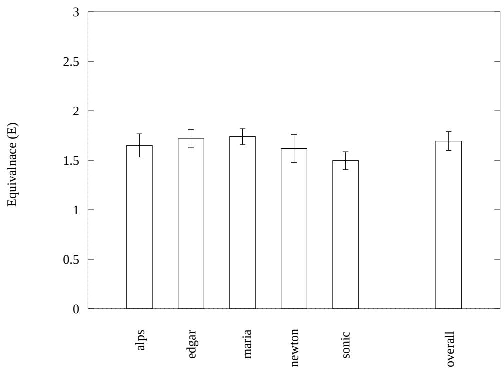
Table 4.1. Hosts used for empirical studies



(a) Sender: void

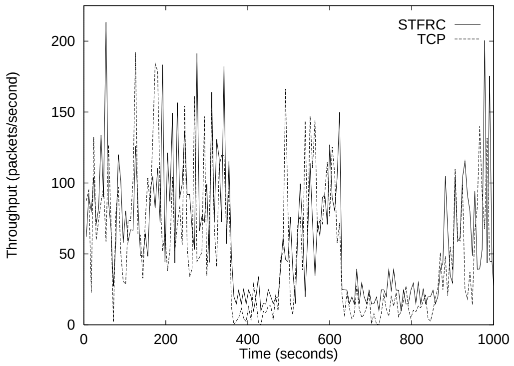


(b) Sender: manic

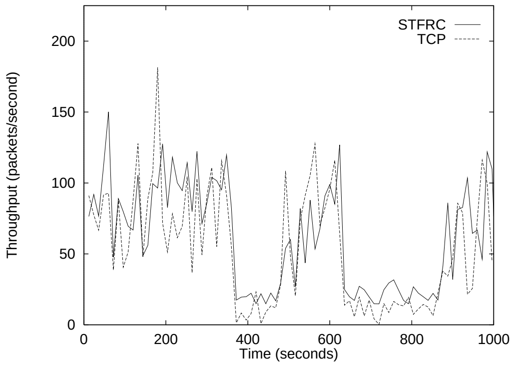


(c) Sender: bmt

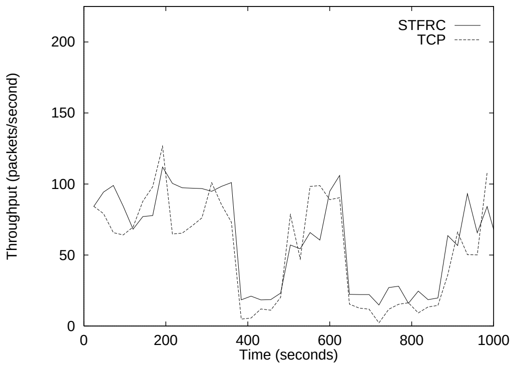
Figure 4.6. Experimental results



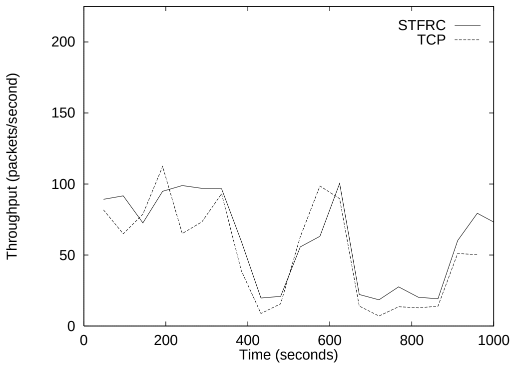
(a) Throughput every 6 seconds



(b) Throughput every 12 seconds



(c) Throughput every 24 seconds



(d) Throughput every 48 seconds

Figure 4.7. Throughput at various timescales

when no packets are lost in an entire recomputation period, since the formula in (4.1) is not valid for zero loss rate. During periods of no loss, the TCP window grows linearly (ignoring the initial slow start period), by one every RTT. Since the sending rate is proportional to the window size, one can say that the sending rate of TCP grows linearly during periods of no loss. We found that when we try to mimic this linear increase behavior in STFRC, the protocol performed poorly (i.e., the friendliness ratio was higher). This is due to the fact that in most of our simulations and Internet experiments, the fair share of the STFRC connection tended to be relatively small. If, after a recomputation interval of no loss, we increased the rate in a linear fashion, the

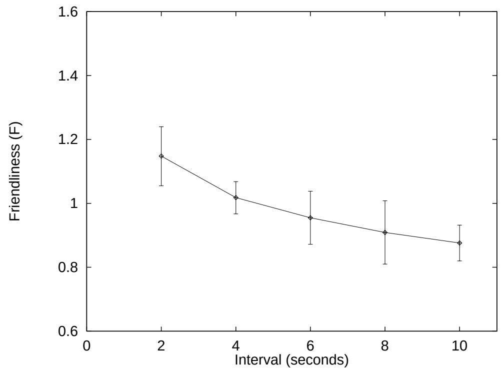


Figure 4.8. Sensitivity to measurement interval. void-alps

relative change in the rate was very high (e.g., if no loss occurred during a three-second period, and if RTT was 100ms, the rate would increase by $3/0.1 = 30$). This led to very high losses in the next round, which in turn dropped the sending rate to a very low value, leading again to a no-loss, or low-loss, period. This oscillatory behavior was detrimental to the performance of the protocol. Doubling the sending rate seems to offer a good compromise between responsiveness (ramping up the sending rate quickly) and avoiding oscillatory behavior. We will come back to this question in Chapter 5.

The value of W_{max} can significantly affect the send rate computed using the formula in (4.1) at low loss rates. While in the simulation studies it is easy to ensure that competing TCP and STFRC flows had the same value for W_{max}, this is hard to ensure in practice. We note that this problem is inevitable whenever flow control is employed: two TCP connections experiencing the same network conditions, but having different values for W_{max}, will have different send rates.

Another design issue is how to set the initial value for r_0. For the simulation and implementation results reported in this chapter, we set this value to approximately 40 packets/second. As long as the recomputation interval M was small compared to the time over which the friendliness or equivalence was being measured, the value for r_0 had little impact on the performance of STFRC.

As mentioned in Section 4.5, timer inaccuracies and overheads force us to send packets out in small bursts, instead of clocking them out evenly over the duration of each *round*. The impact of this burstiness on the performance of the STFRC protocol is hard to quantify. On one hand, one may imagine that burstiness would lead to slightly higher loss rates for the STFRC connection, forcing the send rate down. On the other hand, traffic from a TCP flow is somewhat bursty as well [9]. Thus the impact of bursty nature of STFRC flow on friendliness ratio is hard to judge.

It should be noted that the formula in (4.1) is not valid for certain network scenarios, such as TCP connections running over modem lines with large dedicated buffers, as discussed in Chapter 2. This implies that the STFRC protocol would not work well in these situations either. We are currently working on solutions to this problem. We also note that STFRC reacts to changes in network conditions only every M time units (i.e., the duration of recomputation interval). If the network traffic conditions change on a faster time scale, the difference between the send rate of a TCP connection and a STFRC connection experiencing similar network conditions may be significant. Under such dynamic conditions, obtaining accurate loss estimates and round trip times can be problematic. We note that in real-world testing, Figures 4.6(a)- 4.6(c), we have found that the protocol works well with a recomputation interval of three seconds.

4.7 Conclusions

In this chapter, we have presented a simple TCP-friendly rate adjustment protocol. The protocol achieves TCP-friendliness by changing its sending rate, based on analytic characterization of TCP send rate developed in Chapter 2. In addition to studying the protocol through simulations, we implemented a prototype version of the protocol and tested it with experiments over the Internet. The results of both simulation and implementation experiments show that the protocol is able to achieve send rate that is close to the the send rate of a TCP connection traveling over the same network path. Thus, we conclude that formula-based feedback-loop approach to congestion control and achieving TCP-friendliness is indeed practical.

One of our goals in designing this protocol, was simplicity. We chose a simple feedback mechanisms (acknowledge every packet), a simple loss estimation algorithm (measurement of packet losses over a fixed period of time), a simple algorithm for rate change (change rate at fixed intervals to the one predicted by the formula). Despite its simplicity, the protocol is shown to work under certain networking scenarios. On the other hand, the simplicity does result in several shortcomings. The protocol calculates the loss rate and updates its send rate only at fixed time intervals. Thus, the protocol can be TCP-friendly only at timescales significantly larger than the timescale over which loss rate computations are done. The protocol lacks any mechanism to handle slowstart, and begins with a fixed send rate. In the absence of packet losses, the protocol doubles it sending rate. This can lead to oscillatory behavior. A less aggressive strategy for increasing the sending rate in the absence of packet losses is needed.

Now, we ask ourselves the question: if we are willing to add to the complexity of the protocol, can we address some of the shortcomings? Several aspects of the protocol offer a tradeoff between complexity and performance. For example, we can design a more sophisticated loss rate estimation algorithm that is better able to track the loss rate. We can devise a rate change strategy that results in a smoother send rate. In the next chapter, we begin by discussing some of these tradeoffs, and design a new protocol that addresses these issues.

CHAPTER 5

AN IMPROVED TCP-FRIENDLY PROTOCOL

5.1 Introduction

In the previous chapter, we presented a simple TCP-friendly rate control protocol. While concluding the chapter, we pointed out that while the protocol is simple, and works well under certain networking scenarios, improvements are still possible. However, these improvements would add to the complexity of the protocol. Recall that simplicity of design was among the primary goals for the protocol described in the last chapter. In this chapter we design an equation-based, TCP-friendly protocol that addresses various shortcomings of the protocol presented in the previous chapter. This improvement in the performance comes at the cost of added complexity.

Let us begin by identifying the shortcomings of the STFRC protocol that we will address in this chapter:

- First and foremost, a better technique for loss rate estimation is clearly needed. The STFRC protocol measured loss rate over fixed time intervals. With fixed length intervals, it is difficult to ensure that a measurement interval contains an adequate number of samples to obtain a meaningful loss rate. For example, if the round trip time of the connection varies significantly, then this technique might not perform well.

- TCP reacts only once to multiple packet losses that occur in a single round trip time. The loss rate estimation mechanism of TFRC should be modified to account for this.

- Recall that STFRC adjusts its sending rate after a fixed time interval that is least an order of magnitude larger than the round trip time. This makes the protocol less adaptive to traffic fluctuations that may occur on smaller timescales. It also means that the sender may experience sharp changes in the sending rate at the end of each measurement interval. In [68] the authors have shown that such sharp changes are not desirable from the viewpoint of a CM application.

- In the absence of packet losses, the protocol doubles it sending rate. This can lead to oscillatory behavior. As we will see, a less aggressive strategy for increasing sending rate in absence of packet losses can avoid these oscillations.

- The performance of the protocol should be investigated over multiple time scales. The STFRC protocol updates its sending rate only at fixed time intervals. Thus, its performance can be meaningfully investigated only at time scales larger than update time scale. Note that the simulation and experimental results presented in this chapter considered send rates aggregated over very long durations (upto 1000 seconds). The performance on smaller timescales can be of significant interest to network engineers. This requires us to define metrics that can evaluate the performance of the protocol over different timescales.

- The STFRC protocol was always started with a fixed sending rate and did not include any mechanism similar to TCP's slowstart.

The protocol presented in this chapter address these shortcomings. As mentioned earlier, the price paid for added performance is the increased complexity. We call this protocol the TCP-Friendly rate control protocol (TFRC).

The rest of the chapter is organized as follows. Section 5.2 describes the new protocol. A discussion of protocol features is provided in Section 5.3. Simulation results are presented in Section 5.4, along with a discussion of performance metrics. Section 5.5 presents results from an implementation study. Finally, in Section 5.6, we present our conclusions, and directions for future work.

5.2 Description of the protocol

5.2.1 Protocol overview

The TFRC protocol is split into two parts, a sender-side protocol and a receiver-side protocol. The protocol, like STFRC, is designed to use the TCP response equation (equation (4.1)) as the control equation for rate update.

The basic TFRC protocol is as follows. The sender sends a stream of data packets to the receiver at some rate. The receiver sends a feedback packet to the sender once every round trip time. Based on the information contained in the feedback packets, the sender adjusts its sending rate in accordance with Equation (4.1), to maintain TCP-friendliness. If no feedback is received from the receiver in two round trip times, the sender cuts its sending rate in half.

The values of the round trip time (RTT), the loss event rate (p) and the base timeout value T_0 are needed to calculate the send rate using Equation (4.1). In the TFRC protocol, the sender is responsible for calculating the values of RTT and T_0, while the receiver is responsible for calculating the value of p. We now describe the protocol in detail.

The rest of the section is organized as follows. In Section 5.2.2 we describe the contents of the packets exchanged by the sender and the receiver. In Section 5.2.3, we describe the sender-side protocol. In Section 5.2.4, we define the concept of *loss event* and describe the algorithm used by the receiver to compute the *average loss event rate*. Finally, in Section 5.2.5 we describe the receiver-side protocol, using the concepts developed in Section 5.2.4. We note that in the following description, all rates are measured in packets per unit time. We also assume that packets are received in the order in which they are sent, i.e. the network may drop the packets, but it does not reorder them.

5.2.2 Packet contents

Before we describe the sender and receiver functionality, it is useful to describe the contents of the data packets sent by the sender and feedback packets sent by the receiver.

5.2.2.1 Sender packets

Each data packet sent by the sender contains the following information:

- A sequence number. This number is unique and increases serially.
- Timestamp indicating when the packet is sent. We denote by t_i^s the timestamp of the packet with sequence number i.
- Current estimates of round trip time and base timeout. The estimates reported in packet i are denoted by RTT_i and T_{0_i}, respectively.
- Current sending rate. The estimate of sending rate reported in packet i is denoted by R_i.

5.2.2.2 Receiver packets

Each feedback packet sent by the receiver contains the following information:

- Timestamp of the last packet received. We denote this by e^{ts}. If the last packet received at the receiver has sequence number i, then $e^{ts} = t_i^s$.
- Amount of time elapsed between the receipt of the last packet, and the generation of this feedback report. We denote this by t^d.
- The rate at which the receiver received data since the last feedback report was generate. We denote this by R^r.

- Current estimate of loss event rate. We denote this by p.

5.2.3 Sender protocol

The sender sends a stream of data packets to the receiver at some rate. When a feedback packet is received from the receiver, the sender changes its sending rate, based on the information contained in the feedback report. The contents of the feedback report have been described in Section 5.2.2. If the sender does not receive a feedback report for two round trip times, it cuts it sending rate in half. This is achieved by means of a timer called the *nofeedback* timer.

We describe the sender-side protocol in three steps. In Section 5.2.3.1, we describe the sender behavior when a feedback packet is received. Then, in Section 5.2.3.2, we describe the steps taken when the *nofeedback* timer expires. Finally, in Section 5.2.3.3, we describe how the sender is initialized.

5.2.3.1 Sender behavior when a feedback packet is received

Assume that the feedback packet is received by the sender at time t', the current estimate of round trip time is RTT and the current estimate of base timeout value is T_0. The sender then performs the following steps:

- Calculate a round trip sample: $RTT_s = t' - e^{ts} - t^d$.
- Update the round trip time estimate: $RTT = \alpha * RTT + (1 - \alpha) * RTT_s$. The value of α is a configurable parameter. With higher value of of α, the sender reacts quickly to the changes in round trip time.
- Update T_0 in a manner similar to TCP [2], using RTT_s.
- If $p > 0$
  - Calculate R_n, that is the allowed "TCP-friendly" sending rate, based on the values of p, RTT and T_0, using Equation (4.1).

- If $R < R_n$
  - Set $R = \min(R_n, R + 1/RTT, 2 * R^r)$
- Otherwise, (i.e. $R \geq R_n$)
  - Set $R = R_n$.

- Otherwise, (i.e. $p = 0$)
  - Set $R = \max(\min(2 * R, 2 * R^r), 1/RTT)$.

- Stop *nofeedback* timer if it was running. Reset it to expire after after $\max(2 * RTT, 2/R)$ time units.

We note that when $p = 0$, the sender doubles its sending rate for every feedback packet it receives. While this may seem similar to the STFRC protocol described in Chapter 4, the actual behavior is far less aggressive. Also note that when the sender increases the sending rate, the new rate may not exceed $2 * R^r$. More discussion is provided in Section 5.3.1.

5.2.3.2 Expiration of *nofeedback* timer

If the *nofeedback* timer expires, the sender carries out the following steps:

- R = R/2

- Set *nofeedback* to expire after $\max(2 * RTT, 2/R)$ time units.

Thus, if the receiver stops sending feedback reports, the sender will eventually reduce its sending rate to a very low value.

5.2.3.3 Sender initialization

To initialize the sender, RTT and T_0 are initialized in a manner similar to TCP [2]. The initial value of R is set to 1 packet/second. The *nofeedback* timer is initially set to expire after 2 seconds.

This completes the description of the sender side protocol. We now describe the concepts of *loss event* and *loss event rate*, before describing the receiver protocol.

5.2.4 Loss event and average loss event rate

The main function of the receiver is to estimate the average loss event rate, p. In this section, we we formalize the notion of a loss event and describe the algorithm used by the receiver to compute the average loss event rate.

5.2.4.1 Definition of a loss event

To formally defining the term *loss event*, we first develop an intuitive definition of the term in the context of a window-based protocol like TCP, and then formalize it for a rate-based protocol like TFRC.

Consider a TCP-NewReno sender with congestion window w. It is allowed to have w unacknowledged packets outstanding. These packets have left the sender, but have not yet been acknowledged by the receiver - i.e., they are "somewhere in the network". Let i be the first outstanding packet, and $i + w$ be the last one. Let us suppose that the sender now receives three duplicate acknowledgments for packet j, where $i \leq j < i+w$. The sender will thus consider packet j to be lost and it will react to this loss by reducing its congestion window size in half. Assume further that it next detects the loss of packet k, where $j < k \leq i+w$. The TCP sender will not react to this packet loss. The rationale behind this decision is that when there are multiple packets outstanding in the network, and some of them are lost, they are likely lost due to the same "congestion event". Thus it is sufficient to react only once.

Since TCP is allowed to have a window worth of packets outstanding, we may say that when multiple packets are lost within a window of data, the TCP sender treats it as a single *loss event*. This event can be associated with the first packet of the loss

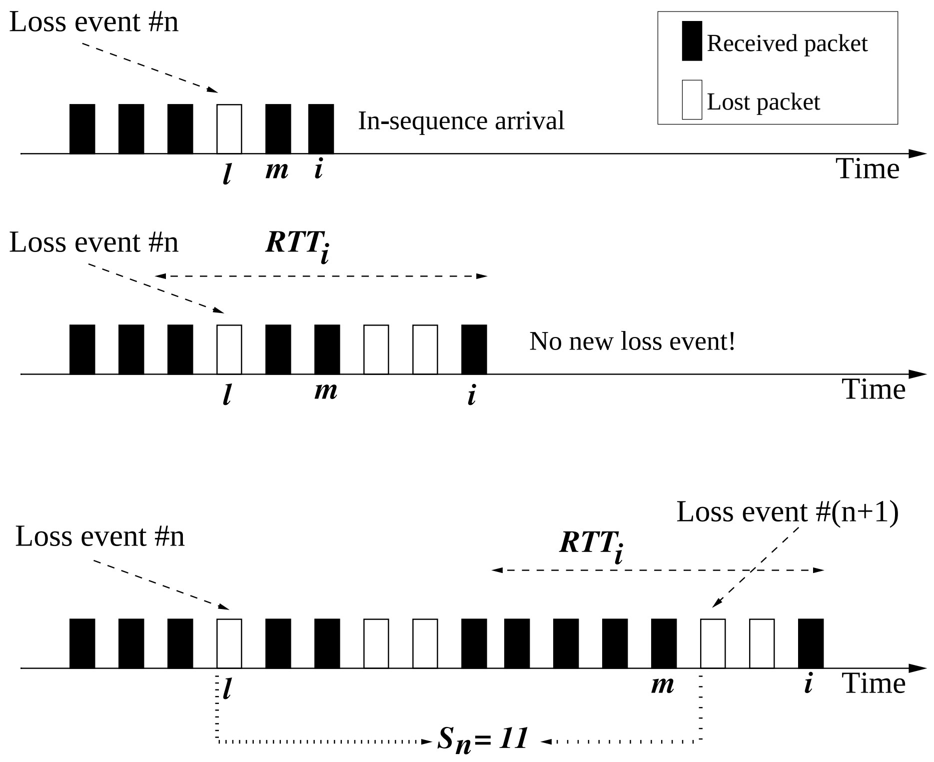


Figure 5.1. Loss event detection

event. Thus, in the above case, we can say that the TCP sender experienced a *loss event* at packet j.

A TFRC sender does not maintain a congestion window like TCP. Since the TCP sender sends one window worth of packets in a round trip time, one may approximate TCP behavior by saying that if multiple packets are lost within one round trip time, the TCP sender will react only once. In other words, multiple packet losses within a round trip time are treated as a single *loss event*.

We now describe how the receiver detects loss events. We illustrate this with help of Figure 5.1. The figure shows three timelines. Each timeline shows a number of packets. The position of a packet on the timeline depends on when the packet was sent from the sender. The packets that make it to the receiver are shown by filled rectangles, while the packets that are not received are indicated by an empty rectangle.

Assume that the n loss events have already occurred, and the n^{th} loss event occurred at packet l. Let m be the maximum sequence number seen at the receiver so far. Note that $m > l$. Now assume that a packet i arrives at the sender. There are two possibilities:

1. $i = m + 1$: Packet received in sequence. This scenario is shown in the first timeline of Figure 5.1.

2. $i > m + 1$: The break in the sequence number is taken as an indication that packets $m + 1, \ldots, i - 1$ are lost. There are two possibilities:

   (a) $t_l^s >= t_i^s - RTT_i$: These lost packets were sent within one round trip time of the last loss event, and hence are part of the same loss event. This scenario is shown in the second timeline of Figure 5.1.

   (b) $t_l^s < t_i^s - RTT_i$: This indicates a new loss event $(n+1)^{st}$ at packet $m + 1$. This is shown in the third timeline of Figure 5.1.

To initialize the algorithm, we need to assign an initial value to l. We will describe how this is done in Section 5.2.5.

The n^{th} loss interval is defined as the interval between loss events n and $(n + 1)$. The length of n^{th} loss interval is denoted by s_n, and is defined as the number of packets sent between the two successive loss events. For example, if the n^{th} loss event occurred at packet i and the $(n + 1)^{st}$ loss event occurred at packet j, then $s_n = j - i + 1$.

5.2.4.2 Loss event rate estimation

Based on the history of packets received and loss events observed, the receiver calculates the average loss event rate. The loss event rate (p) estimation algorithm used for TFRC is called Weighted average of loss intervals (WALI). In the following, we describe WALI in detail. In Section 5.3, we provide discussion of various features of WALI, and compare it with other algorithms for estimating loss event rates.

The WALI algorithm computes the average loss rate over the last n loss intervals. The receiver considers last n loss intervals, and computes an average loss interval $\hat{s}$:

$$\hat{s} = \frac{\sum_{i=1}^{n} w_i s_i}{\sum_{i=1}^{n} w_i}$$

Where w_i is the weight assigned to the i^{th} loss interval. The estimated loss event rate, is then: $p = 1/\hat{s}$.

The performance of the WALI method depends on two factors. First, the value of n, which is the number of loss intervals used to compute the average. The second is is the weight assigned to each loss interval. When selecting the value of n, one must keep in mind the tradeoff between sensitivity to noise of the calculated loss rate and responsiveness. A high value of n will dampen may provide a smoother average, but will dampen the impact of most recent loss intervals. For the simulation and implementation studies presented in the later parts of this chapter, we use the following weights: $w_8, w_7, w_6, w_5 = 1$; $w_4 = 0.8$; $w_3 = 0.6$; $w_2 = 0.4$; and $w_1 = 0.2$. The results of the simulation and implementation studies show that these values work well over a wide range networking conditions. A careful investigation of various alternatives for the values of n and the weights is an area for future work.

We also need to consider the impact of those packets that have been received since the last loss event, but are not yet part of any loss interval. Note that loss intervals are terminated by lost packets. Assume that the receiver is about to calculate the loss event rate. Assume also that the receiver has received k packets since the previous loss event, and no packets have been lost. This means that none of these packets currently belong to any loss interval. The WALI algorithm, as described above, will ignore these k packets while calculating the average loss event rate. It is clear that if k is sufficiently large, its impact should be taken in to account, otherwise the receiver will continue to report an average loss rate that is higher than the actual loss rate. To deal with this problem, we modify the WALI algorithm as follows. Define:

$$\hat{s} = \max\left(\frac{k * w_1 + \sum_{i=2}^{n} w_i s_i}{\sum_{i=1}^{n} w_i}, \frac{\sum_{i=1}^{n} w_i s_i}{\sum_{i=1}^{n} w_i}\right)$$

and:

$$p = 1/\hat{s}$$

This implies that we include k in the loss interval calculations whenever k is large enough to increase the average loss interval.

We note an important feature of the WALI algorithm. It can be seen that as soon as the receiver has seen at least one loss event, the WALI algorithm will never report a non-zero packet loss rate. Implications of this feature are discussed in Section 5.3.1.

Now that we have formally defined a *loss event*, and described how WALI computes average loss event rate based on packet history, we proceed to describe the receiver functionality.

5.2.5 Receiver protocol

The receiver receives packets from the sender and sends back feedback reports once every round trip time. The receiver uses a timer called the *feedback* timer, to trigger the feedback reports. To generate the reports, the receiver maintains a history record for each packet. The record consists of:

- The packet sequence number,
- Status of the packet, which is one of the following three: received, presumed lost and marks a loss event, presumed lost but does not mark a loss event.

- Timestamp of the packet. Timestamp of packet i is denoted by t_i^s.

- Time at which the packet was received. For packet i, this is denoted by t_i^r.

We describe the receiver-side protocol in three steps. In Section 5.2.5.1, we describe the receiver behavior when a data packet is received. Then, in Section 5.2.5.2, we describe the steps taken when the *feedback* timer expires. Finally, in Section 5.2.3.3, we describe how the receiver is initialized.

5.2.5.1 Receiver behavior when a data packet is received

Assume that a packet with sequence number i has just arrived at the receiver, the current time is t, the maximum sequence number of a packet received at the receiver so far is m and the last loss event occurred at packet l. The receiver performs the following steps.

- Add packet i to history, mark it as received.

- Set $t_i^r = t$

- If $i > m + 1$:

  - As explained in Section 5.2.4.1, the break in the sequence number is taken as an indication that packets $m + 1, \ldots, i - 1$ are lost.

  - Add packets $m + 1, \ldots, i - 1$ to history, mark each one "presumed lost but does not mark a loss event".

  - If $t_l^s < t_i^s - RTT_i$, mark packet $m + 1$ as "presumed lost and indicates a loss event". This is the loss even detection algorithm described in Section 5.2.4.1. Set $t_{m+1}^s = t_i^s$.

- Set $m = i$.

5.2.5.2 Expiration of *feedback* timer

Assume that the current time is t, the feedback timer has just expired, the maximum sequence number of a packet received at the receiver by time t is m, the last feedback report was sent at time t', and the maximum sequence number of a packet received at the receiver by time t' was m'. The receiver then performs the following steps.

- If $m > m'$:
  - Calculate the average loss event rate p using the WALI algorithm described in Section 5.2.4.2.
  - Set $e^{ts} = t^s_m$
  - Set $t^d = t - t^r_m$
  - Let n be number of packets added to the history, and marked as "received" between t' and t. Set $R_r = n/(t - t')$.
  - Send a feedback packet with values of p, e^{ts}, t^d and R_r.
- Set the feedback timer to expire at RTT_m.

Note that if no packets have been received since the generation of the last report, no new report is sent.

5.2.5.3 Receiver initialization

The receiver is initialized by the first packet that arrives at the receiver. Assume that the first packet arrives at time t and carries a sequence number of i. The receiver then performs the following steps:

- Add the packet to the history, mark it as received. Thus, all the packets lost until the first received packet are ignored.

- Set $l = i$
- Set $m = i$
- Set $t_i^r = t$
- Send a feedback packet with following values:
  - $e^{ts} = t_i^s$
  - $t^d = 0$
  - $R_r = MAXRCV$, where $MAXRCV$ is a configurable parameter. It is usually set to some very large value, such as the largest floating point number allowed by the system.
  - $p = 0$
- Set the feedback timer to expire after RTT_i

This completes the description of the TFRC protocol. In the next section, we discuss the impact of various features of the protocol.

5.3 Discussion of protocol features

5.3.1 Sender behavior when receiver reports zero loss rate

In Section 5.2.3, we pointed out that when $p = 0$, the sender doubles its sending rate for every feedback packet it receives. This seems like very aggressive behavior, similar to STFRC. However, as we pointed out in Section 5.2.4.2, the loss rate estimation algorithm at the receiver is designed in such a way that p is always greater than 0 following the first loss event. Thus, the receiver only reports $p = 0$ until the first loss event occurs. The sender may double its sending rate once per RTT only until $p = 0$.

This sender behavior is simular to the slowstart algorithm employed by TCP [2]. The rapid increase in the send rate during the slowstart phase is necessary for the sender to quickly ramp up to the available bandwidth. We also note that the minimum sending rate during the slowstart period is $1/RTT$. This is done to allow the sender to faster increase from its initial sending rate of one packet per second.

After the first loss event, the receiver will report a loss rate that is greater than 0, and the sender will come out of the slowstart phase. After the slowstart phase, sender can increase its sending rate by at most $1/RTT$ per RTT. In addition, note that when the sender increases it sending rate, it is never allowed to increase it past twice the rate at which the receiver has received data since the previous feedback report. This mechanisms further limits the increase in the send rate, especially during the slowstart phase.

Thus, the TFRC protocol is far less aggressive than STFRC. In the STFRC protocol, the receiver may report a zero packet loss rate at any time. The sender always doubles its sending rate whenever zero packet loss rate is reported.

5.3.2 Loss fraction vs. loss event fraction

The obvious way to measure loss is as a loss fraction calculated by dividing the number of packets that were lost by the number of packets transmitted. However, as described earlier, TCP responds only once to multiple packet losses within the same window. We model this behavior in TFRC by considering multiple packet losses within the same round trip time as a single loss event.

When routers use RED [14] queue management, multiple packet drops in a window of data are less common, but with drop-tail queue management it is common for several packets to be lost the same round-trip-time to when the queue overflows. This can result in multiple packets dropped from a window of data from a single flow, resulting in a significant difference between the loss fraction and the loss event fraction for that flow.

To see how the loss-event fraction differs from the regular loss fraction in the presence of random packet loss, consider a flow that sends N packets per round-trip time, and assume a Bernoulli loss model with loss probability p_{loss}. The probability that at least one packet is lost in a given round-trip time is $1-(1-p_{loss})^N$. Therefore the loss-event fraction p_{event}, calculated as number of loss events per packet sent, is given by:

$$p_{event} = \frac{1-(1-p_{loss})^N}{N}$$

Note that for a fixed loss probability, the the loss-event fraction decreases as the sender transmission rate increases. However, the transmission rate is determined by the congestion control scheme, and so itself depends on p_{event}. For a very high loss environment where the congestion window is rarely higher than one, and for a low loss environment, there will be little difference between the packet loss rate and the loss event rate for a flow. However, for a moderate loss environment where the congestion window is usually higher than one, there is some difference between the two. A more formal discussion of this problem is presented in [55].

The version of the TCP response function in Equation (4.1) uses a loss model, which, as the measurements in Chapter 2 show, may be approximated by measuring the loss event rate. Ideally, this response function would be replaced with a TCP response function based on directly on loss event rates.

5.3.3 Increasing the transmission rate

One issue to resolve is how to increase the sending rate when the rate given by the control equation is greater than the current sending rate. As the loss rate may be dependent of the transmission rate, to avoid oscillatory behavior in the sending rate, it may be necessary to provide a damping mechanism. One such mechanism is to restrict the increase to be small relative to the sending rate during the period that it takes for the effect of the change to show up in feedback that reaches the sender. As described before, the increase rate does not exceed one packet per RTT every RTT, which is the rate of increase of a TCP flow in congestion avoidance mode.

5.3.4 The response to persistent congestion

Simulations have shown that, in contrast to TCP, TFRC requires from three to eight round-trip times to reduce its sending rate by one half in response to persistent congestion. As discussed before, this slower response to congestion is coupled with a slower increase in the sending rate than that of TCP. In absence of congestion, TCP increases the sending rate by one packet/RTT for every round-trip time. In case of TFRC, the WALI algorithm will not decrease the average packet loss rate until a longer-than-average period has passed without congestion (see Section 5.2.5). Hence, TFRC will increase its send rate only after this period has passed. Thus, the gradual decrease in the send rate of TFRC in response to increase in congestion is coupled with a gradual increase in response to absence of congestion.

5.3.5 Comparison of WALI with other algorithms

In Section 5.2.4.2, we described WALI algorithm to compute the average loss event rate. Several other algorithms for loss event rate estimation are possible. In this section, we present two such algorithms, and compare their performance with WALI.

5.3.5.1 Event-based packet history (EBPH)

This algorithm calculates loss event rate over a history window of packets that includes a fixed number of loss events. Assume that the feedback timer has just expired, and the highest sequence number of a packet received by the receiver so far is m. The receiver then looks back in the packet history until it finds k loss events,

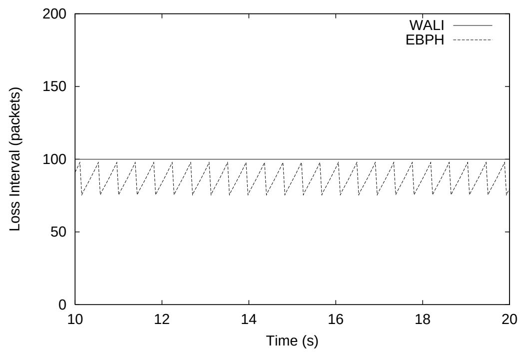


Figure 5.2. Comparison of WALI and EBPH with idealized periodic loss

where k is a configurable parameter. If k loss events are found in $n(< m)$ previous packets (i.e. $m - n + 1, \ldots, m$), then the average loss event rate is $p = k/n$. If the receiver is able to detect only $l < k$ loss events in the entire available history of m packets, then the average loss event rate is $p = l/m$.

This algorithm suffers from the effect that even with a perfectly periodic loss pattern, loss events entering and leaving the sample can cause changes to the measured loss rate, and hence add unnecessary noise to the loss signal. This is illustrated by the following simple simulation scenario. In this scenario, the network drops every 100^{th} packet. The value of k for the $EBPH$ algorithm is set to 4. The average loss intervals calculated by WALI and EBPH are shown in Figure 5.2. It can be seen that the WALI reports a steady average loss interval that corresponds to the periodic loss pattern, while the average loss interval calculated using the EBPH method shows fluctuations even with a periodic loss pattern. We note that while the simulation scenario itself is rather unrealistic because of the perfect periodicity of packet losses, it helps to illustrate the difference in the two mechanisms.

5.3.5.2 Exponentially weighted moving average (EWMA)

This algorithm computes an exponentially weighted moving average of loss intervals. The loss event rate, p, is computed as the inverse of this average. Assume that n loss events have already occurred, and the average loss interval is $\hat{s}$. Now assume that $(n+1)^{st}$ loss event occurs. This loss event allows us to calculate s_n. The average loss interval is then updated as follows:

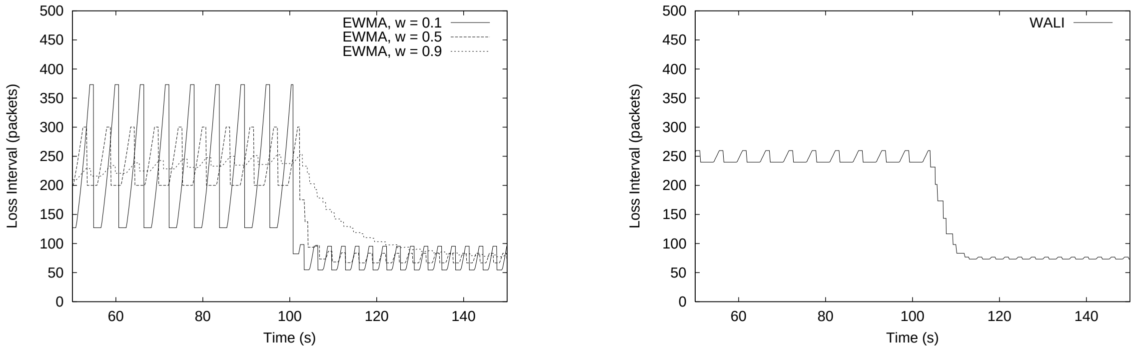
$$\hat{s} = w * \hat{s} + (1 - w) * s_n$$

and:

$$p = 1/\hat{s}$$

The value of w is a configurable parameter. Note that this algorithm takes into account *all* the past loss events. If the value of w is low, then the value of p will depend strongly on the most recent loss events, and may exhibit high variance. This will increase the variance in the sending rate. If the value of w is high, the value of p will be slow to react to any changes in the loss rate. Thus sender may be slow to respond to changes in the congestion level. The EWMA method can be modified to take into account the number of packets received since last loss event, in a manner similar to WALI. For all comparisons between EWMA and WALI in this section, we use the modified EWMA algorithm.

To compare EWMA and WALI, consider the following scenario. In this scenario, the loss intervals alternate between 100 and 400 packets for the first 100 seconds. In other words, the following packets are dropped: $100, 500, 600, \ldots$. After 100 seconds, the loss intervals alternate between 100 and 50. The scenario is artificial, and is designed to highlight the shortcomings of the EWMA method. We test the performance of EWMA with three values of $w : 0.1, 0.5$ and 0.9. When the value of w is lower, less weight is given to the history. The average loss interval calculated by the EWMA



(a) Average loss interval calculated by EWMA

(b) Average loss interval calculated by WALI

Figure 5.3. Comparison of EWMA and WALI

method is shown in Figure 5.3(a). Performance of the WALI algorithm under the same loss pattern is shown Figure 5.3(b).

To reduce the variability in the average loss event rate, the EWMA algorithm must give a large weight to the history. It can be observed that for a value of $w = 0.9$, the variability in the average loss event rate computed by the EWMA algorithm is comparable to the variability in the average loss event rate computed by the WALI algorithm. However, because of the large weight given to the history, the EWMA algorithm is much slower than WALI to respond to the change in the loss pattern.

If less weight ($w = 0.1$) is given to the history, the EWMA algorithm is quick to respond to the change in the loss pattern. However, because of the smaller weight given to the history, the average loss event rate computed by the EWMA algorithm shows high variability, when compared to the WALI algorithm.

The above comparisons of WALI with EBPH and EWMA have been done under unrealistic loss patterns. However, these simulations do indicate that WALI may be preferable over EWMA and EBPH in order to reduce variations in the sending rate, while still responding quickly to any changes in the loss rate. More detailed studies of WALI under various real-world scenarios are required before the TFRC protocol can be considered for deployment in the Internet.

5.4 Simulation results

We have tested TFRC extensively across the public Internet, in the Dummynet network emulator [58], and in the *ns* [10] network simulator. In the next section, we present a summary of simulation results, and in section 5.5 we look at behavior of the TFRC implementation over Dummynet and the Internet.

5.4.1 Performance metrics

We are primarily interested in two measures of performance of the TFRC protocol. First, we wish to compare the average send rates of a TCP flow and a TFRC flow experiencing similar network conditions. Second, we would like to compare the "smoothness" of these send rates. Ideally, we would like a TFRC flow to achieve the same average send rate as that of a TCP flow, and yet have less variability.

The timescale at which the send rates are measured affects the values of these measures. We define the send rate of a given data flow F at time t, measured at a timescale δ:

$$R_{\delta,F}(t) = \frac{s * \text{ packets sent by F between } t \text{ and } t+\delta}{\delta}, \tag{5.1}$$

where s is the packet size in bytes. We characterize the send rate of the flow between time t_0 and t_1, where $t_1 = t_0 + n\delta$, by the time series: $\{R_{\delta,F}(t_0 + i\delta)\}_{i=0}^{n}$. The coefficient of variation (CoV), which is the ratio of standard deviation to the average, of this time series can be used as a measure of variability [22] of the sending rate of the flow at timescale δ. A lower value implies a smoother flow.

To compare the send rates of two flows a and b, at a given time scale, we define the equivalence at time t:

$$e_{\delta,a,b}(t) = \min\left(\frac{R_{\delta,a}(t)}{R_{\delta,b}(t)}, \frac{R_{\delta,b}(t)}{R_{\delta,a}(t)}\right), \quad R_{\delta,a}(t) > 0 \, or \, R_{\delta,b}(t) > 0 \tag{5.2}$$

Taking the minimum of the two ratios ensures that the resulting value remains between 0 and 1. Note that the equivalence of two flows at a given time is defined only when at least one of the two flows has a non-zero send rate. The equivalence of two flows between time t_0 and t_1 can be characterized by the time series:

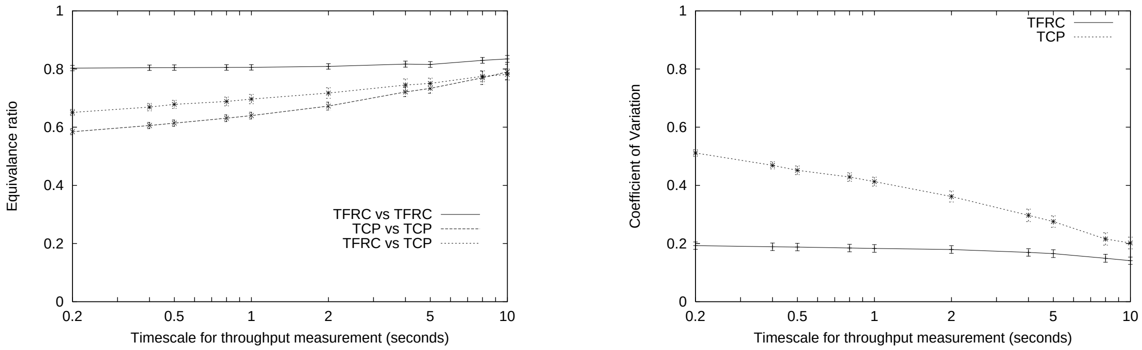
$$\{e_{\delta,a,b}(t_0 + i * \delta)\}_{i=0}^{n}$$

The average value of the defined elements of this time series is called the equivalence ratio of the two flows at timescale δ. The closer it is to 1, the more "equivalent" the two flows are. We choose to take the average instead of the median to capture the impact of any outliers in the equivalence time series. We can compute the equivalence ratio between a TCP flow and a TFRC flow, between two TCP flows or between two TFRC flows. Ideally, the ratio should be very close to 1 over a broad range of timescales between two flows of the same type experiencing the same network conditions.

5.4.2 Simulation scenarios

5.4.2.1 Long duration flows

For measuring the steady performance of the TFRC protocol, we consider the simple well-known single bottleneck simulation scenario, as described in Section 4.4.1. The access links are sufficiently provisioned to ensure that any packet drops/delays due to congestion occur only at the bottleneck link.



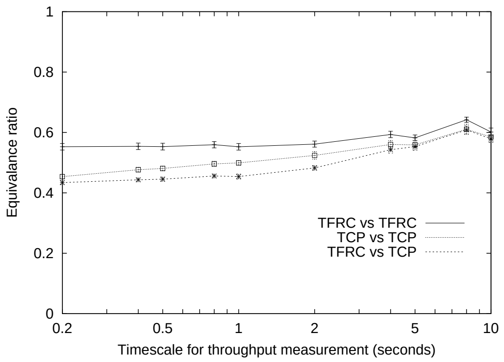
(a) TCP and TFRC equivalence

(b) Coefficient of Variation of TCP and TFRC

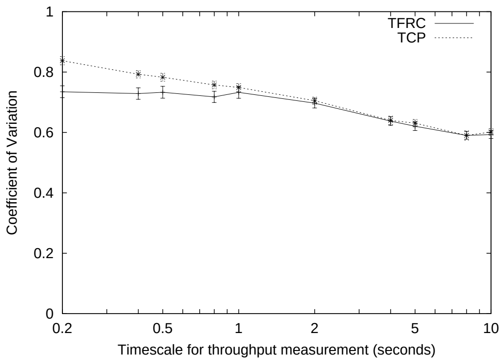
Figure 5.4. Comparison of TCP and TFRC for long-duration traffic

We considered several simulation scenarios, but illustrate here a scenario with a bottleneck bandwidth of 15Mbps and a RED queue. The bottleneck delay is 50ms and the packet size is 1000 bytes. The bottleneck queue runs RED with *gentle* enabled, a total buffer of 100 packets, a *minthresh* of 10 and a *maxthresh* of 50. There are 16 SACK TCP and 16 TFRC flows. The simulation duration is 150 seconds, and the results are from the last 100 seconds of the simulation. The round-trip time of each flow, excluding the queuing delay, is random, uniformly distributed between 80 and 120 milliseconds. The flows are started at random times, uniformly distributed between 0 and 10 seconds. To plot the graphs, we monitor the performance of one flow belonging to each protocol. The graphs are the result of averaging 14 such runs, and the 90% confidence intervals are shown. The loss rate observed at the bottleneck router was about 0.1%.

Figure 5.4(a) shows the equivalence ratios of TCP and TFRC as a function of the timescale of measurement. Plots are shown for the mean equivalence ratio between pairs of TCP flows, between pairs of TFRC flows, and between pairs of different types of flows. The equivalence ratio of TCP and TFRC is between 0.6 to 0.8 over a



(a) TCP and STFRC equivalence



(b) Coefficient of Variation of TCP and STFRC

Figure 5.5. Comparison of TCP and STFRC for long-duration traffic

broad range of timescales. This shows that TCP and TFRC flows have comparable send rates. The equivalence ratio between TFRC flows is higher compared to the equivalence ratio between TCP flows. This implies that the difference in the send rates of several TFRC flows sharing a bottleneck link is smaller than the difference in the send rates of several TCP flows sharing a bottleneck link.

Figure 5.4(b) shows the CoV for TCP and TFRC flows as a function of the timescale. We observe that CoV of TFRC is lower than that of TCP over the timescale shown in the graph. This implies that the the send rate of TFRC is smoother than that of TCP over a broad range of timescales. Both this and the better TFRC equivalence ratio are due to the fact that TFRC responds only to the aggregate loss rate, and not to individual loss events.

To compare the performance of this version of the TFRC protocol with the STFRC protocol proposed in Chapter 4, we repeat our simulations with the STFRC protocol. The recomputation interval for the STFRC protocol is set to 2 seconds. The results are shown in Figures 5.5(a) and 5.5(b).

Comparing the graphs in Figures 5.4(a) and 5.5(a), we can see that TFRC is more TCP friendly than STFRC. From the graph in Figure 5.5(b), we can see that STFRC protocol is almost as bursty as TCP, regardless of the timescale. This is in sharp contrast to the behavior seen in Figure 5.4(b). We believe that the better performance of the new version of the TFRC protocol is due to its better loss estimation technique, which is more successful in tracking the steady state loss rate produced by the steady background traffic. The less aggressive algorithm for increasing the sending rate also contributes to improved performance.

5.4.2.2 ON-OFF UDP traffic

In this simulation scenario, we model the effects of competing web-like traffic (very small TCP connections, some UDP flows). It has been reported in [47] that WWW-related traffic tends to be self-similar in nature. In [74], it is shown that self-similar traffic may be created by using several ON/OFF UDP sources whose ON/OFF times are drawn from heavy-tailed distributions such as the Pareto distribution. Figures 5.6-5.8 present results from simulations in which we simulate such background traffic. The mean ON time is 1 second and the mean OFF time is 2 seconds, and during an ON period each source sends at 500Kbps. The number of simultaneous connections is varied between 50 and 150. In addition to the ON/OFF flows, there are two long duration connections: a TCP connection and a TFRC connection. The simulation is run for 5000 seconds. The results are averages of 10 runs. The bottleneck link characteristics are the same as in the previous simulation. These simulations produce a wide range of loss rates, as shown in Figure 5.6.

We measure the send rates of the TCP and the TFRC connection on several different timescales and show the results in Figures 5.7 and 5.8. We observe in Figure 5.7 that at loss rates below 10% the equivalence ratio of TFRC and TCP connections is between 0.7 to 0.8 which is similar to the steady-state case. At loss rates of more than

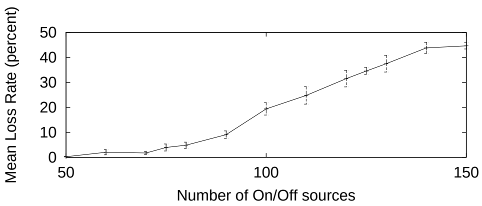


Figure 5.6. Loss rate at the bottleneck router, with ON-OFF background traffic

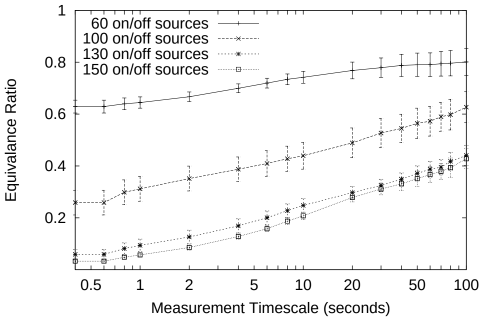


Figure 5.7. TCP equivalence with TFRC, with ON-OFF background traffic

30%, the equivalence ratio is low at all but the longest timescales because packets are sent so rarely, and any interval in which only one of the flow sends no packets gives a value of zero in the equivalence time series, while the intervals in which neither flow sends any packets are not counted. This tends to result in a lower equivalence ratio. However, on long timescales, even at 40% loss (150 ON/OFF sources), the equivalence ratio is still 0.4, meaning that one flow gets about 40% more than its fair share and one flow got 40% less. Thus TFRC is seen to be comparable to TCP over a wide range of loss rates even when the background traffic is quiet variable.

Figure 5.8 shows that the send rate of TFRC is much smoother than the send rate of TCP, especially when the loss rate is high. Note that the CoV for both flows is much higher compared to the values in Figure 5.4(b) at comparable timescales. This is due to the higher loss rates and the variable nature of background traffic in these simulations.

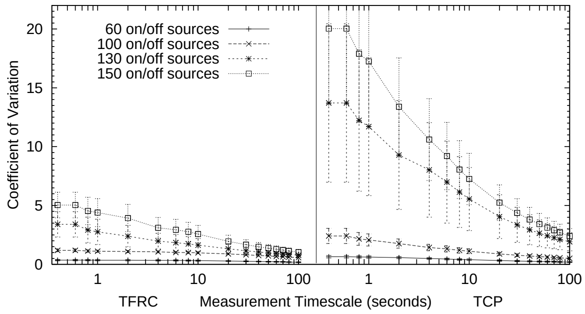


Figure 5.8. Coefficient of Variation of TFRC (left) and TCP (right), with ON-OFF background traffic

5.5 Implementation results

We have implemented the TFRC algorithm, and conducted many experiments to explore the performance of TFRC in the Internet. Our tests include two different transcontinental links, and sites connected by a microwave link, T1 link, OC3 link, cable modem, and dial-up modem. In addition, conditions unavailable to us over the Internet were tested against real TCP implementations in Dummynet.

In this section, we provide only a summary of the implementation results. The results presented in this section are reproduced from [73], with permission of the author. More detailed results, and a description of the prototype can be found in [73].

To summarize all the results, TFRC is generally fair to TCP traffic across the wide range of network types and conditions we examined. Figure 5.9 shows a typical experiment with three TCP flows and one TFRC flow running concurrently from London to Berkeley, with the bandwidth measured over one-second intervals. In this case, the transmission rate of the TFRC flow is slightly lower, on average, than that of the TCP flows. At the same time, the transmission rate of the TFRC flow is smooth, with a low variance; in contrast, the bandwidth used by each TCP flow varies strongly even over relatively short time periods, as shown in Figure 5.11. Comparing this with Figure 5.8 shows that, in the Internet, both TFRC and TCP perform very similarly to the lightly loaded (50 sources) "ON/OFF" simulation environment which had less

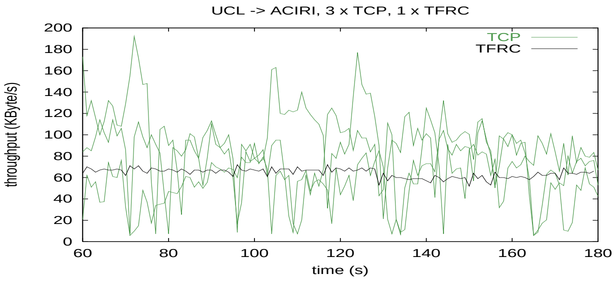


Figure 5.9. Three TCP flows and one TFRC flow over the Internet.

than 1% loss. The loss rate in these Internet experiments ranges from 0.1% to 5%. Figure 5.10 shows that fairness is also rather similar in the real world.

5.6 Conclusion

In this chapter we have outlined a proposal for equation-based unicast congestion control for unreliable, rate-adaptive applications. We have evaluated the protocol extensively using simulations and experiments in the Internet. We find that under a wide variety of network conditions, the TFRC protocol achieves an average send rate that is comparable to the average send rate of a TCP connection experiencing similar network conditions. We also find that the send rate of a TFRC connection shows less variability than the send rate of a TCP connection, under a wide variety of network conditions. The TFRC protocol is more complex than the STFRC protocol proposed in Chapter 4. On the other hand, our results show that TFRC performs better STFRC described in Chapter 4 under a wide variety of networking conditions. Thus, we have answered affirmatively the question we posed in Section 4.7.

The current implementations of the TFRC congestion control mechanisms (in *ns* and in the actual implementation) have an omission that we are planning to correct. The current congestion control mechanisms are designed for a sender that always has data available to send (until the last packet has been sent). When we began this work, our intention was to emulate the behavior of TCP as much as possible; however,

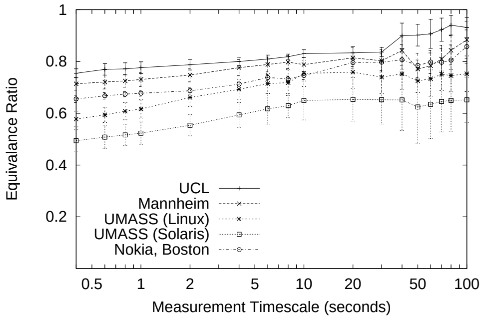


Figure 5.10. TCP equivalence with TFRC over different Internet paths

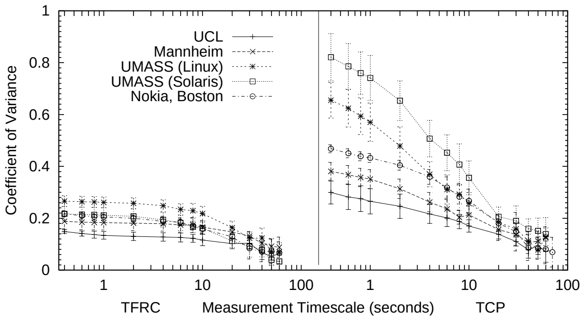


Figure 5.11. Coefficient of Variation of TFRC (left) and TCP (right) over different Internet paths

there was no consensus on the appropriate response of TCP congestion control to a quiescent or application-limited period, where the previously-authorized congestion window or sending rate was not fully used. A proposal for modification of TCP congestion control to deal with a quiescent sender has been described in [17]. Our plan is to implement a rate-based variant of this approach in TFRC.

While the current implementation of TFRC gives robust behavior in a wide range of environments, we certainly do not claim that this is the optimal set of mechanisms for unicast, equation-based congestion control. Active areas for further work include mechanisms for the receiver's update of the packet drop rate estimate after a long period with no packet drops, and the sender's adjustment of the sending rate in response to short-term changes in the round-trip time. We assume that, as with TCP's congestion control mechanisms, equation-based congestion control mechanisms will continue to evolve based both on further research and on real-world experiences.

As an example, we are interested in the potential of equation-based congestion control in an environment with Explicit Congestion Notification (ECN) [54].

There have also been proposals for increase/decrease congestion control mechanisms that reduce the sending rate in response to each loss event, but that do not use AIMD; we would like to compare TFRC with these congestion control mechanisms as well. We believe that the emergence of congestion control mechanisms for relatively-smooth congestion control for unicast traffic can play a key role in preventing the degradation of end-to-end congestion control in the public Internet, by providing a viable alternative for unicast multimedia flows that would otherwise be tempted to avoid end-to-end congestion control altogether [12].

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this chapter we summarize the research presented in this dissertation and present directions for future work.

6.1 Summary of research

We considered the problem of congestion control in the current Internet. Our focus was on understanding TCP congestion control and designing TCP-friendly congestion control protocols for continuous media flows. In Chapter 1, we motivated and introduced the problem. In Chapters 2, 3, 4 and 5, we considered these two problems in detail.

In Chapter 2 we presented a simple model of TCP-Reno congestion control. We validated the model by extensive measurements done in the Internet. The model captures the essence of TCP's congestion avoidance behavior and expresses send rate as a function of loss rate. The model takes into account the behavior of the protocol in the presence of timeouts, and is valid over the wide range of loss probabilities. We compared our model with the behavior of several real-world TCP connections. We found that our model provides a very good match to the observed behavior. Our model yielded a simple, closed-form formula for calculating the send rate of a TCP-Reno connection under given network conditions. This formula can be used for many purposes such as fast evaluation of large networks and design of TCP-Friendly protocols.

In Chapter 3 we presented a detailed Markovian model of TCP-Reno congestion control. We built this model without making many of the simplifying assumptions made while building the model described in Chapter 2. The Markov model does not have appear to have a closed form solution. However, numerical solutions of the model match the simple model presented in Chapter 2 very well, thus validating the simpler model.

In Chapter 4, we presented a simple TCP-Friendly rate control protocol (STFRC), based on the TCP model proposed in Chapter 2. In addition to studying the protocol through simulations, we implemented a prototype version of the protocol and tested it with experiments over the Internet. The results of both simulation and implementation experiments show that the protocol is able to achieve a send rate that is close to that of a TCP connection traveling over the same network path. Thus, we conclude that equation-based approach to congestion control and achieving TCP-friendliness is indeed practical.

One of our goals in designing the STFRC protocol was simplicity. Despite its simplicity, the protocol was shown to work under certain networking scenarios. On the other hand, its simplicity does result in several shortcomings. In Chapter 5, we begin by discussing these shortcomings. We investigate how the simplicity of the protocol can be traded off to address some of the shortcomings. We design a new protocol named TCP-friendly rate control protocol (TFRC) that incorporates better techniques for loss rate estimation, slowstart functionality, a strategy to deal with periods of very high loss and a rate change strategy that results in a smoother send rate. These techniques add to the complexity of the protocol, but in return provide better performance. We define performance metrics to evaluate the protocol over a wide range of timescales. We test the protocol using extensive simulations and experiments over the Internet. Our results show that TFRC performs better than

STFRC and is a good congestion control mechanism for applications that would prefer a smoother sending rate than that of TCP, while maintaining TCP-friendliness.

The focus of our work has been on congestion control. Our work helps towards better understanding of TCP congestion control. We addressed the need to design congestion control for the growing class of continuous media applications that cannot use TCP congestion control for various reasons. Our research is a combination of analysis and experimentation: we not only design models of TCP congestion control, but we also test their validity by conducting experiments over the Internet. We not only design TCP-friendly protocol and evaluate them via simulations but we build prototype implementation and test them over the Internet.

We believe that our work lays the foundation for designing comprehensive congestion control protocols for continuous media traffic in the best-effort Internet.

6.2 Future work

- **Further development and evaluation of TCP-friendly rate control protocols**

  We have presented two equation-based, TCP-friendly rate control protocols in this dissertations. Many avenues are open for future work in this area. Here we point to some of them.

  - A better understanding and evaluation of various techniques for loss rate estimation is needed.
  - A control-theoretic analysis of equation-based protocols can shed light on various performance issues such as transient response of the protocol and stability of network containing a large number of such flows.

  - A proposal for modification of TCP congestion control to deal with a quiescent sender has been described in [17]. Our plan is to implement a rate-based variant of this approach in TFRC.
  - We are interested in the potential of equation-based congestion control in an environment in which Explicit Congestion Notification (ECN) is widely deployed.
  - It is possible design a rate control protocol around a control function other than the one in Equation (4.1). A comparative study of various control functions and their respective benefits and drawbacks can shed further light on this issue.
  - A rich variety of issues come up when one seeks to extend this work in the multicast domain. A formal definition of TCP-friendliness in a multicast setting is lacking. Various algorithms for loss rate and round trip time measurements, constrained by the limitations imposed by the multicasting environment, need to be explored.

- **A model for TCP-SACK**

  When we began the research presented in this dissertation, TCP-Reno was the dominant flavor of TCP deployed in the Internet. However, TCP-Reno is being rapidly replaced by TCP-SACK [33]. We plan to develop a model of TCP-SACK, similar to the model of TCP-Reno presented in this dissertation. We plan to use more accurate loss models for this analysis, compared to the one we used in Chapter 2.

- **Understanding the impact of widespread deployment of TCP-SACK**

  TCP-SACK uses selective acknowledgments to minimize retransmission timeouts. We expect that under most networking conditions, a sender using TCP-

SACK will suffer fewer retransmission timeouts than a sender using TCP-Reno. As a result, we believe that the average send rate of a TCP-SACK connection will be higher than that of a TCP-Reno connection. In other words, TCP-SACK is more aggressive than TCP-Reno. This opens up several questions.

- In an environment, where the total available bandwidth remains the same, will widespread replacement of TCP-Reno by TCP-SACK result in increased packet loss rate?
- If the answer to the above questions is affirmative, then what is the benefit of widespread deployment of TCP-SACK? Would the send rate of TCP-SACK have less variance compared to TCP-Reno?

In an abstract version of this problem, we can consider various functions that map loss rate and round trip time to send rate, and try to determine the optimal response function.

BIBLIOGRAPHY

[1] Internet protocol, September 1981. RFC791.

[2] Allman, M., Paxson, V., and Stevens, W. TCP congestion control, April 1999. RFC2581.

[3] Bolliger, J., Gross, T., and Hengartner, U. Bandwidth modelling for network-aware applications. In *Proc. IEEE INFOCOM* (1999).

[4] Bolot, J., and Vega-Garcia, A. Control mechanisms for packet audio in the Internet. In *Proc. IEEE INFOCOM* (1996).

[5] Cardwell, N., Savage, S., and Anderson, T. Modeling TCP latency. In *Proc. IEEE INFOCOM* (2000).

[6] Chiu, D. M., and Jain, R. Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems 17* (June 1989).

[7] Clark, D., Shenker, S., and Zhang, L. Supporting realtime applications in an integrated service packet network: architecture and mechanism. In *Proc. ACM SIGCOMM* (1992).

[8] Clark, D., and Wroclawski, J. An approach to sevice allocation in the Internet, July 1997. IETF draft-clark-diff-svc-alloc00.txt.

[9] Fall, K., and Floyd, S. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *Computer Communication Review 26*, 3 (July 1996).

[10] Fall, K., and Varadhan, K. *ns: Manual*, February 2000.

[11] Firoiu, V., and Borden, M. Study of active queue management for congestion control. In *Proc. IEEE INFOCOM* (2000).

[12] Floyd, S., and Fall, K. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Trans. Networking* (August 1999).

[13] Floyd, S., Handley, M., Padhye, J., and Widmer, J. Equation-based congestion control for unicast applications. Tech. Rep. 00-03, International Computer Science Institute, 2000.

[14] Floyd, S., and Jacobson, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Networking 1*, 4 (August 1997).

[15] Handley, M., and Flyod, S. Strawman specification for TCP friendly (reliable) multicast congestion control (TFMCC). Unpublished manuscript.

[16] Handley, M., and Flyod, S. TCP-Friendly Simulations. Unpublished manuscript.

[17] Handley, M., Padhye, J., and Floyd, S. TCP ongestion window validation. Tech. Rep. 99-77, Department of Computer Science, University of Masschusetts at Amherst, 1999.

[18] Heyman, D. P., Lakshman, T. V., and Neidhardt, A. L. A new method for analyzing feedback-based protocols with applications to engineering web traffic over the Internet. In *Proc. ACM SIGMETRICS* (1997).

[19] Jacobs, S., and Eleftheriadis, A. Providing video services over networks without quality of service guarantees. In *World Wide Web Consortium Workshop on Real-Time Multimedia and the Web* (1996).

[20] Jacobson, V. Congestion avoidance and control. *Computer Communication Review 18*, 4 (August 1988).

[21] Jacobson, V. Modified TCP congestion avoidance algorithm, 1990. Note sent to end2end-interest mailing list.

[22] Jain, R. *The Art of Computer Systems Performance Analysis.* John Wiley and Sons, 1991.

[23] Jain, R., Ramakrishnan, K. K., and Chiu, D. M. Congestion avoidnace in computer networks with a connectionless network layer. Tech. Rep. 506, Digital Equipment Corporation, August 1987.

[24] Karn, P., and Partridge, C. Improving round-trip time estimates in reliable transport protocols. *Computer Communication Review 17*, 5 (August 1987).

[25] Keshav, S. *An engineering approach to computer networking.* Addison-Wesley, 1997.

[26] Kumar, A. Comparative performance analysis of versions of TCP in local network with a lossy link. *IEEE/ACM Trans. Networking 6*, 4 (August 1998).

[27] Kuo, B. *Automatic Control Systems.* Prentice Hall, 1995.

[28] Kurose, J. Open issues and challanges in providing QoS gurantess in high-speed networks. *Computer Communication Review 23*, 1 (January 1993).

[29] Lakshman, T. V., and Madhow, U. Performance analysis of window-based flow control using TCP/IP. In *Proc. Conference on High Performance Networking* (1994).

[30] Lakshman, T. V., and Madhow, U. The performance of TCP/IP for networks with high bandwidth-delay products. *IEEE/ACM Trans. Networking 5*, 3 (June 1997).

[31] Lakshman, T. V., Madhow, U., and Suter, B. Window-based error recovery and flow control with a slow acknowledgment channel: a study of TCP/IP performance. In *Proc. IEEE INFOCOM* (1997).

[32] Mahdavi, J., and Floyd, S. TCP-friendly unicast rate-based flow control, January 1997. Note sent to end2end-interest mailing list.

[33] Mathis, M., Mahdavi, J., Floyd, S., and Romonow, A. TCP selective acknowledgement options, October 1996. RFC2018.

[34] Mathis, M., Semke, J., Mahdavi, J., and Ott, T. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communication Review 27*, 3 (July 1997).

[35] McCanne, S., Jacobson, V., and Vetterli, M. Receiver driven layred multicast. In *Proc. ACM SIGCOMM* (1996).

[36] Mena, A., and Heidemann, J. An empirical study of Real Audio traffic. In *Proc. IEEE INFOCOM* (2000).

[37] Misra, V., Gong, W. B., and Towsley, D. Stochastic differential equation modeling and analysis of TCP windowsize behavior. In *Proc. Performance* (1999).

[38] Misra, V., Gong, W. B., and Towsley, D. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *Proc. ACM SIGCOMM* (2000).

[39] Mogul, J., Kent, C., Partridge, C., and McCloghrie, K. IP MTU discovery options, July 1988. RFC1063.

[40] Ostermann, S. tcptrace: tcpdump file analysis tool, 1996.

[41] Ott, T., Kemperman, J., and Mathis, M. The stationary behavior of ideal TCP congestion avoidance. Unpublished manuscript. ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps.

[42] Ozdemir, V., and Rhee, I. TCP emulation at the receivers (TEAR), November 1999. Presentation at the RM meeting.

[43] Padhye, J., Firoiu, V., and Towsley, D. A stochastic model of TCP reno congestion avoidance and control. Tech. Rep. 99-02, Department of Computer Science, University of Masschusetts at Amherst, 1999.

[44] Padhye, J., Firoiu, V., Towsley, D., and Kurose, J. Modeling TCP throughput: A simple model and its empirical validation. In *Proc. ACM SIGCOMM* (1998).

[45] Padhye, J., Kurose, J., Towsley, D., and Koodli, R. TCP-friendly rate adjustment protocol for continuous media flows over best effort networks. In *Proc. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)* (1999).

[46] Papadopoulos, C., and Parulkar, G. Reatransmission-based error control for continuous media applications. In *Proc. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)* (1996).

[47] Park, K., Kim, G., and Crovella, M. On the relationship between file sizes, transport protocols and self-similar network tarffic. In *Proc. International Conference on Network Protocols* (1996).

[48] Paxson, V. Automated packet trace analysis of TCP implementations. In *Proc. ACM SIGCOMM* (1997).

[49] Paxson, V. End-to-end Internet packet dynamics. In *Proc. ACM SIGCOMM* (1997).

[50] Paxson, V., and Floyd, S. Why we don't know how to simulate the Internet. In *Proccedings of the 1997 Winter Simulation Conference* (1997).

[51] Postel, J. User datagram protocol, August 1980. RFC768.

[52] Postel, J. Transmission control protocol, September 1981. RFC793.

[53] Qiu, L., Zhang, Y., and Keshav, S. On individual and aggregate TCP performace. In *Proc. International Conference on Network Protocols* (1999).

[54] Ramakrishnan, K. K., and Floyd, S. A proposal to add explicit congestion notification (ECN) to IP, January 1999. RFC2481.

[55] Ramesh, S., and Rhee, I. Issues in model-based flow control. Tech. Rep. 99-15, Department of Computer Science, North Carolina State University, 1999.

[56] Rejaie, R., Handley, M., and Estrin, D. An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. In *Proc. IEEE INFOCOM* (1999).

[57] Rhee, I. Error control techniques for interactive low-bit rate video transmission over the Internet. In *Proc. ACM SIGCOMM* (1998).

[58] Rizzo, L. Dummynet and forward error correction. In *Proc. Freenix* (1998).

[59] Ross, S. *Applied Probability Models with Optimization Applications*. Dover, 1970.

[60] Sahu, S., Nain, P., Towsley, D., Diot, C., and Firoiu, V. On achievable service differentiation with token bucket marking for TCP. In *Proc. ACM SIGMETRICS* (2000).

[61] Sahu, S., Towsley, D., and Kurose, J. A quantitative study of differentiated services for the Internet. In *Proc. IEEE GLOBECOM* (1999).

[62] Schulzrinne, H., Casner, S., Fredrick, R., and Jacobson, V. RTP: A transport protocl for real-time applications. RFC1989.

[63] Sisalem, D., and Schulzrinne, H. The loss-delay based adjustment algorithm: A TCP-friendly adaption scheme. In *Proc. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)* (1998).

[64] Stevens, W. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC2001.

[65] Stevens, W. *TCP/IP Illustrated, Vol.1 The Protocols*. Addison-Wesley, 1997. 10th printing.

[66] Stevens, W. *TCP/IP Illustrated, Vol.2 The Implementation*. Addison-Wesley, 1997. 4th printing.

[67] Suter, B., Lakshman, T. V., Stiliadis, D., and Choudhury, A. K. Design considerations for supporting TCP with per-flow queuing. In *Proc. IEEE INFOCOM* (1998).

[68] Tan, D., and Zakhor, A. Real-time internet video using error resilient scalable compression and TCP-friendly transport protocol. *IEEE/ACM Trans. Multimedia* (May 1999).

[69] Thomas, V. *IP multicast in RealSystem G2*, January 1998. Obtain via http://real.com.

[70] Thompson, K., Miller, G., and Wilder, M. Wide-area internet traffic patterns and characteristics. *IEEE Network Magazine 11*, 6 (November 1997).

[71] Turletti, T., Parisis, S., and Bolot, J. Experiments with a layered transmission scheme over the Internet. Tech. Rep. RR-3296, INRIA, France., 1997.

[72] Vicisano, L., Rizzo, L., and Crowcroft, J. TCP-like congestion control for layered multicast data transfer. In *Proc. IEEE INFOCOM* (1998).

[73] Widmer, J. Equation based congestion control. Master's thesis, University of Mannheim, February 2000.

[74] Willinger, W., Taqqu, M., Sherman, R., and Wilson, D. Self-similarity through high variability: Statistical analysis of ethernet LAN traffic at the source level. In *Proc. ACM SIGCOMM* (1995).

[75] Yajnik, M., Moon, S., Kurose, J., and Towsley, D. Measurement and modeling of the temporal dependence in packet loss. In *Proc. IEEE INFOCOM* (1999).