Congestion Control for Streaming Video and Audio Applications

by

Deepak Bansal

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2001

Author ....
Department of Electrical Engineering and Computer Science
January 4, 2001

Certified by ....
Hari Balakrishnan
Assistant Professor
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Congestion Control for Streaming Video and Audio Applications
by
Deepak Bansal

Submitted to the Department of Electrical Engineering and Computer Science
on January 4, 2001, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

As the Internet continues to grow, new applications like streaming video and audio and Internet telephony are emerging. These applications must implement end-to-end congestion control as the Internet relies on end hosts implementing congestion control for its stability. However, these applications have different requirements from the applications that the Internet has supported traditionally, and therefore call for new congestion control algorithms to be developed to suit their needs. In particular, they prefer smoother transmission rates over maximum possible transmission rates due to their real-time interactive nature.

This thesis introduces and analyzes a class of nonlinear congestion control algorithms called *binomial algorithms*, motivated in part by the needs of streaming audio and video applications for which a drastic reduction in transmission rate upon each congestion indication (e.g., packet loss) is problematic. Binomial algorithms generalize TCP-style additive-increase by increasing inversely proportional to a power k of the current window (for TCP, $k = 0$) ; they generalize TCP-style multiplicative-decrease by decreasing proportional to a power l of the current window (for TCP, $l = 1$). We show that there are an infinite number of deployable TCP-friendly binomial algorithms, all of which satisfy $k + l = 1$, and that all binomial algorithms converge to fairness under a synchronized-feedback assumption provided $k + l > 0; k, l \geq 0$. Our simulation results show that binomial algorithms interact well with TCP across a RED gateway. We also study the behavior of various classes of slowly-responsive TCP-friendly congestion control algorithms and their co-existence with TCP. We find that such algorithms lead to loss in throughput and sometimes affect Internet stability, unless packet conservation is employed. However, the various TCP-friendly congestion controls with different speeds of response to the network conditions are unfair to each other over short time-scales, as well as under varying network conditions.

Thesis Supervisor: Hari Balakrishnan
Title: Assistant Professor

Acknowledgments

This thesis is dedicated to my family in India. They have supported me at rough times and shared my joy at good times through all these years.

I owe the greatest thanks to my advisor, Prof. Hari Balakrishnan, whose energy and enthusiasm have been a constant source of inspiration to me. He kept me focussed on the problem and his insights and intuition helped me many times when I got stuck in my research. He is the best advisor a student can wish, both as a person and as a mentor.

I am grateful to Sally Floyd and Scott Shenker with whom I spent three months at ACIRI, Berkeley where part of this research was done. They provided useful comments on my basic results and helped me develop them further.

I would also like to thank my officemates, Bodhi Priyantha, Suchi Raman and Jorge Rafael, who apart from providing me a quiet, motivating environment, also helped me with many useful discussions. I am grateful to the other graduate students of my group, Network and Mobile Systems, especially Dave Anderson, Allen Miu and Alex Snoeren who helped me at various times. Professor John Guttag, Steve Garland, John Ankcorn, and Dorothy Curtis provided much support and useful comments on my work.

Some of the text in this thesis has been taken from my paper with Hari Balakrishnan, which is to appear at IEEE Infocom 2001. Some text has also been taken from an in-progress paper with Hari Balakrishnan, Sally Floyd, and Scott Shenker. My research at MIT was primarily supported by a grant from the NTT Corporation.

I would also like to thank the students, faculty, and staff at LCS for making LCS such an exciting place to be at.

Dedicated to my parents ...

Contents

List of Figures

Chapter 1

Introduction

"In the end the aggressors always destroy themselves,
making way for others who know how to cooperate and get along.
Life is much less a competitive struggle for survival
than a triumph of cooperation and creativity."
-Fritjof Capra

Congestion occurs in data networks when the users of the network collectively demand more resources like bandwidth and buffer space than the network has to offer. Unless appropriate action is taken to control network congestion, the network can end up in a state of persistent overload, potentially leading to *congestion collapse[15]*, resulting in degraded overall performance. Whereas some switched networks like the telephone network implement some form of admission control to ensure that the admitted users get acceptable performance even under heavy demand, the datagram-based, packet switched Internet does not adopt this approach. Motivated in part by an end-to-end argument [40], the "interior" of the Internet architecture has been kept simple and most of the intelligence and system management functions have been pushed to the end systems [9]. This design choice has allowed new applications and innovation to be incorporated easily into the Internet and has been the key driver for the growth and the ubiquitous deployment of the Internet over heterogenous link technologies. This principle, however, has resulted in the lack of any global infrastructure for coordinating congestion control among the various routers in the Internet. Thus, the Internet relies on the *cooperation* among end-hosts to implements suitable congestion avoidance and control mechanisms.

Traditionally, the Internet has been used primarily by applications like bulk data transfers and interactive sessions such as interactive terminals, electronic mail, file transfers and the Web ([12]). These applications require the reliable in-order transport of data reliably across the network as quickly as possible. These are "elastic" applications: the key metric of performance for these applications is the speed of transport of data across the network. The stability of the Internet to date has in large part been due to the congestion control and avoidance algorithms [20] implemented in its dominant transport protocol, TCP [33, 41]. Based on the principle of additive-increase/multiplicative-decrease (AIMD) [8, 41], a TCP connection probes for extra bandwidth by increasing its congestion window linearly with time, and on detecting congestion, reducing its window multiplicatively by a factor of two. Under certain assumptions of synchronized feedback, Chiu and Jain have shown that an AIMD control scheme converges to a stable and fair operating point [8], providing a sound basis for Jacobson's algorithms found in most current TCP implementations [2].

However, over the last few years, the Internet has seen a proliferation of a wide range of applications with differing requirements and metric of performance. These applications impose new constraints on the congestion control policies in the Internet, bringing up a pressing need to develop new congestion control policies for these applications. We discuss the problems with the traditional congestion control policies in the Internet below.

1.1 The Problem

TCP is not well-suited for several emerging applications such as streaming and real-time audio and video. These applications have a utility function that is a function of both the rate of data transport as well as changes in the rate of transport of data. This is because these applications render interactive data to users in real-time; variations in receive rate at the receiver are either directly made visible to the user or taken care of by buffering at the receiver. While the first option results in user discomfort in viewing the data (imagine viewing a video stream that randomly varies in quality between a high resolution and a degraded low resolution version), the second option results in user discomfort waiting for the required amount of buffering to occur (e.g. certain commercial products today buffer as much as 1-2min. of data before playing it) resulting in poor interactivity and delayed response.

1.1.1 Problems with TCP for multimedia streaming

1. TCP uses an algorithm called Additive Increase Multiplicative Decrease (AIMD) to probe for available bandwidth and to respond to congestion (packet losses). However, this algorithm entails a "factor-of-two" rate reduction on each packet loss. This results in huge variations in sending rates which are either made visible directly to the user by sending a video whose quality varies with the variation of the sending rate or the second option is to buffer the data before playing it out to the user which results in reduced interactivity and waiting periods for the user. Thus, either option results in degraded user perceived quality of streaming audio and video.

2. TCP ensures very strict reliability and ordering semantics at the cost of end-to-end delays and delay variations. Reliability is important for traditional applications like file transfer but when the data is being played out to the user in real-time (as in audio and video), delivery of the data to the receiver after the time when it was supposed to be played out is of no use. Also, human users are more resilient to errors and losses.

3. TCP being purely window based results in burstiness in data transmission which gets aggravated due to ACK compression. This burstiness makes the problem of preserving the relative time ordering of the various frames of data received at the receiver the same as at the sender much difficult and inefficient.

1.1.2 Deployment considerations: TCP-friendliness

Because of the problems mentioned in the previous section, we need to look for new congestion control algorithms for streaming multimedia traffic. However, the new congestion control algorithms are subject to several constraints. First, like TCP, they should be fair to each other, converge rapidly to fairness and be responsive to changes in network conditions (we explore these further later). In addition, they must interact well with the existing base of TCPs already being used in the Internet. Such algorithms are called "TCP-compatible" [7, 6]. They ensure that the TCP connections using AIMD get their fair allocation of bandwidth in the presence of these protocols and vice versa. One notion that has been proposed to capture "TCP compatibility" is "TCP-friendliness". It is well known that the throughput λ of a flow with TCP's AIMD congestion control (increase factor $\alpha = 1$ packet, decrease factor $\beta = 1/2$) is related to its loss rate p as $\lambda \propto S/(R\sqrt{p})$, where S is the packet size [16, 24, 30, 31]. An algorithm is TCP-friendly [25] if its throughput $\lambda \propto S/(R\sqrt{p})$ with the same constant of proportionality as for a TCP connection with the same packet size and round-trip time.

1.2 The Solution: Binomial Congestion Control

In this thesis, we introduce and develop a new family of congestion control algorithms, called binomial congestion control algorithms. Binomial algorithms generalize TCP-style additive-increase by increasing inversely proportional to a power k of the current window (for TCP, $k = 0$); they generalize TCP-style multiplicative-decrease by decreasing proportional to a power l of the current window (for TCP, $l = 1$). We show that there are an infinite number of deployable TCP-friendly binomial algorithms, those which satisfy $k + l = 1$, and that all binomial algorithms converge to fairness under a synchronized-feedback assumption provided $k + l > 0; k, l \geq 0$. Our simulation results show that binomial algorithms interact well with TCP across a RED gateway. We focus on two particular algorithms, IIAD (inverse-increase/ additive-decrease, $k = 1, l = 0$) and SQRT ($k = l = 0.5$), showing that they are well-suited to streaming video and audio applications that do not react well to large TCP-style window reductions.

We then study the various TCP-friendly congestion control algorithms with regards to their responsiveness to changes in the network conditions. We define TCP-equivalence as a behavior similar to TCP on time scales of one round trip or shorter. We evaluate the safety and performance issues for TCP-friendly congestion control algorithms that are extremely slow in responding to changes in network conditions.

1.3 Organization of the thesis

We begin by providing a brief description of previous related work in chapter 2. We motivate and develop a new family of congestion control algorithms with differing characteristics than AIMD in chapter 3. We evaluate the performance of these binomial algorithms in chapter 4. In chapter 5, the notion of slowly-responsive congestion controls is formalized and the performance and safety issues are considered. We conclude in chapter 6.

Chapter 2

Related Work

The earliest work on end-to-end congestion control largely focused on reliable transport protocols like TCP [20, 34]. The problem of developing congestion control algorithms for non-TCP applications has received much attention in recent years [25], especially in the research community which strongly believes that the Internet will eventually have the incentives for all the end hosts to do congestion control.

2.1 TCP's congestion control mechanisms

The congestion control and avoidance algorithms for TCP [33, 41], the most dominant protocol on the Internet, were developed in the mid-1980s [20]. They are based on the principle of additive-increase/multiplicative-decrease (AIMD), which can be represented by the following window increase/decrease rules:

$$\begin{aligned} &\text{I: } w_{t+R} \leftarrow w_t + \alpha; \alpha > 0 \\ &\text{D: } w_{t+\delta t} \leftarrow (1-\beta) w_t; 0 < \beta < 1, \end{aligned} \tag{2.1}$$

Here I refers to the increase in window as a result of receipt of one window of acknowledgements in a RTT and D refers to the decrease in window on detection of congestion by the sender, w_t the *congestion window* size at time t, R the round-trip time of the flow, and α and β are constants.

AIMD-based congestion control algorithms have been shown to result in a distributed adaptation to the dynamic state of the network and convergence to an optimal operating point while maintaining fairness [8, 22, 34]. These results provide a sound basis for Jacobson's algorithms found in most current TCP implementations [2]. In most versions of TCP, in steady state, the connection probes for extra bandwidth by increasing its congestion window by one packet on receiving an entire window of acknowledgements. On detecting congestion (usually via a packet loss), the sender reduces its window multiplicatively by a factor of two.

In addition to AIMD, TCP has other mechanisms that determine its behavior during non-steady state. These include slow-start which happens when a new connection is started and timeout which is responsible for reducing the sending rate to less than 1 packet per round trip time. TCP also implements packet conservation (self clocking) which ensures that new data is sent only on the receipt of an acknowledgement or discovery of a loss.

2.2 TCP-friendliness

Any new congestion control algorithm that may be deployed in the Internet must interact well with the already deployed base of TCP on the Internet. This requirement has been defined by the term "TCP-compatible" algorithm [7]. The idea is to ensure that the TCP connections using AIMD get their fair allocation of bandwidth in the presence of new algorithms and vice versa.

One notion that has been proposed to capture TCP-compatibility is "TCP-friendliness" [25]. It is well-known that the throughput λ of a flow with TCP's AIMD congestion control is related to its loss rate p as $\lambda \propto S/(R\sqrt{p})$, where S is the packet size and R the connection's round-trip time [24, 30, 16, 31]. An algorithm is said to be "TCP-friendly" if its throughput $\lambda \propto S/(R\sqrt{p})$ with the same constant of proportionality as for a TCP connection with the same packet size and round-trip time.

2.3 Congestion control for multimedia streaming

A number of algorithms have been developed for multicasting streaming multimedia, many of which are based on the layered encoding of data [13, 28]. These algorithms have tried to solve both the heterogeneity of the receivers as well as fairness to the deployed TCP base. To accommodate heterogeneity of receivers, the burden of rate adaptation is moved from the senders to the receivers. The most prominent among these is Receiver-driven Layered Multicast (RLM) [26]. In this algorithm, the different layers of multimedia are transported on different multicast addresses. The receivers probe for additional bandwidth by subscribing to additional layers and respond to congestion by unsubscribing from those layers. To emulate TCP's AIMD behavior, the receiver's exponentially back-off (in time to subscribe to additional layers) when their attempt to join an additional layer fails. To scale with a large number of receivers shared learning is used. It should be noted that the disparities and issues arising from multicast aside, the essential congestion control algorithm is AIMD based making it necessary for the additional layers to be multiplicatively spaced. Other layered multicast streaming protocols that implement rate adaptation at the receiver include PLM [1] and the scheme by Vicisano et al [43]. MTCP [38] and RMTP [32] are sender based algorithms for multicast congestion control but do not scale well with heterogenous receivers.

There have been number of proposals for congestion control for unicast delivery of multimedia traffic. Rejaie's RAP is essentially AIMD based windowing scheme without loss recovery [37]. Token-based variants of AIMD [10] try to make TCP essentially a rate based scheme while emulating TCP's AIMD windowing.

It should be noted that these variants of TCP for unicast streaming data try to solve only one side of the problem of congestion control for multimedia streaming i.e. loss recovery. Recently, interest has focussed on the other aspect of the problem i.e. huge variations in the sending rate of a congestion control sender. This results in huge variations in the user perceived quality of multimedia data if it is being streamed in real time. To alleviate this problem, [36] deals with the problem of deciding when to add an additional layer so that it can last sufficiently long. This involves suitably additional buffering at the receiver of the lower layers. This buffering will manifest itself only in an additional delay when a multimedia file in being played out. However, for live coverage, this additional delay is often unacceptable.

To achieve both smoother rates and higher interactivity, recently efforts have focussed on reducing the oscillations in the underlying congestion control algorithms themselves. This involves developing newer congestion control algorithms that compete fairly with TCP but do not show as much variation in sending rate as TCP. The proposals in this direction include TCP Emulation At Receivers (TEAR) [39] and Equation-Based Congestion Control (EBCC) [18]. TEAR [39] is a receiver-based implementation of AIMD and can smoothen the sending rate. EBCC explicitly monitors the loss rate and sets its sending rate based on the TCP-friendly equation suggested in [31] and [30].

In the commercial world, multimedia streaming products like real-player from Real Networks [35], Windows Media player [27] use proprietary protocols and algorithms over UDP. It is believed that these products either do not use any congestion control,or use their own versions of application-level congestion control (often not TCP-friendly). Such schemes may threaten the stability of the Internet itself and may result in congestion collapse. More recently, however, some streaming applications are being run over HTTP (over TCP) to successfully penetrate firewalls. However, this approach suffers from two drawbacks - first, the loss recovery done by TCP is not optimal for video and audio and second, the large oscillations in transmission rates that result due to TCP's AIMD.

Recently, an end-system infrastructure called Congestion Manager (CM) [3, 4, 5] has been developed for integrated congestion management, independent of specific transport protocols (like TCP) and applications. CM enables logically different flows (such as multiple concurrent Web downloads, concurrent audio and video streams, etc.) to adapt to congestion, share network information, and share (varying) available bandwidth well. Rather than have each stream act in isolation and thereby adversely interact with the others, the CM maintains host- and domain-specific path information, and orchestrates all transmissions. The CM's internal algorithms ensure social and stable network behavior; its API enables a variety of applications and transport protocols to adapt to congestion and varying bandwidth. This functionality is especially useful for adaptive audio and video applications.

We believe that the lack of use of congestion control (or use of TCP) by streaming media (even unicast) applications is to a large extent due to the unavailability of appropriate congestion algorithms and an architectural framework suitable for these applications. The binomial control algorithms (especially within the CM framework) are intended to fill this gap and provide easily deployable, useful and backward-compatible solution to this problem.

Chapter 3

Binomial Congestion Control Algorithms

In this chapter, we develop a family of nonlinear congestion control algorithms that generalize TCP's linear increase/decrease rules. We call this family of algorithms *binomial algorithms*. Our work is motivated by two goals:

1. TCP's AIMD algorithm results in huge oscillations in sending rate due to a "factor-of-two" rate reduction on each loss (Figure 3-1). Internet video and audio applications do not react well to these oscillations because of the drastic degradations in the user-perceived quality that results. We seek to develop and analyze a family of algorithms that have more smooth transmission rates for such applications.

2. Second, we seek to achieve a deeper understanding of TCP-compatible congestion control by generalizing the class of linear control algorithms, and understanding how a TCP-friendly algorithm competes with TCP for bottleneck resources.

3.1 The Binomial Family: Introduction

An AIMD control algorithm may be expressed as:

$$\begin{aligned} &\text{I: } w_{t+R} \leftarrow w_t + \alpha; \alpha > 0 \\ &\text{D: } w_{t+\delta t} \leftarrow (1-\beta) w_t; 0 < \beta < 1, \end{aligned} \tag{3.1}$$

where I refers to the increase in window as a result of the receipt of one window of acknowledgements in a round-trip time (RTT) and D refers to the decrease in window on detection of congestion by the sender, w_t the window size at time t, R the round-trip time of the flow, and α and β are constants.

The binomial algorithms generalize the AIMD rules in the following way:

$$\begin{aligned} &\text{I: } w_{t+R} \leftarrow w_t + \alpha / w_t^k; \alpha > 0 \\ &\text{D: } w_{t+\delta t} \leftarrow w_t - \beta w_t^l; 0 < \beta < 1 \end{aligned} \tag{3.2}$$

These rules generalize the class of all linear control algorithms. For $k = 0,\ l = 1$, we get AIMD; for $k = -1, l = 1$, we get MIMD (multiplicative increase/multiplicative decrease

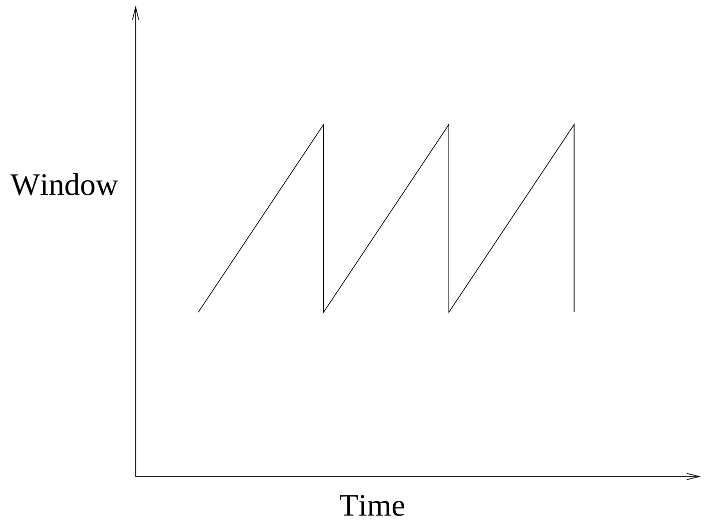

Figure 3-1: TCP's window evolution in steady state.

used by *slow start* in TCP [20]); for $k = -1, l = 0$, we get MIAD; and for $k = 0, l = 0$ we get AIAD, thereby covering the class of all linear controls.

We call this family of algorithms *binomial* congestion control algorithms, because their control expressions involve the addition of two algebraic terms with different exponents. They are interesting because of their simplicity, and because they possess the property that any $l < 1$ has a decrease that is in general *less than* a multiplicative decrease, a desirable property for streaming and real-time audio and video. If there exist values of k and l (other than $k = 0, l = 1$) for which binomial algorithms are TCP-friendly, then it provides a spectrum of potentially safe congestion avoidance mechanisms that are usable by Internet applications that do not react well to large and drastic rate reductions.

3.2 Analysis of binomial controls

In this section, we present and analyze the properties of binomial congestion control algorithms. We start by providing some intuition about the sample paths traversed by the congestion window in a binomial algorithm, and showing that it converges to fairness under simplified conditions of synchronized feedback to sources. We initially make a few simplifying assumptions, but build on it in subsequent sections by deriving an analytic formula that relates the throughput of a binomial algorithm to the loss rate it observes. We then use this formula to obtain the conditions under which a binomial algorithm is TCP-friendly.

3.2.1 Intuition

We use the technique of Chiu and Jain and represent the two-source case as a "phase plot," where the axes correspond to the current window sizes, x_i, of each source (for convenience, we normalize each x_i to a number between 0 and 1, so it represents the fraction of the total window size aggregated across all competing sources). As the system evolves with time, the two sources adjust their windows according to the control equations, leading to a sample path in this phase space. The key to understanding binomial controls is to realize how these paths move in the phase space. To start with, we summarize how linear controls behave [8]:

1. Additive-increase/decrease: Moves parallel to the 45^o-line. Additive-increase improves fairness (in the sense of Jain's fairness index[1]), additive-decrease reduces it.

2. Multiplicative-increase/decrease: Moves along the line connecting (x_1, x_2) to the origin. Fairness is unchanged.

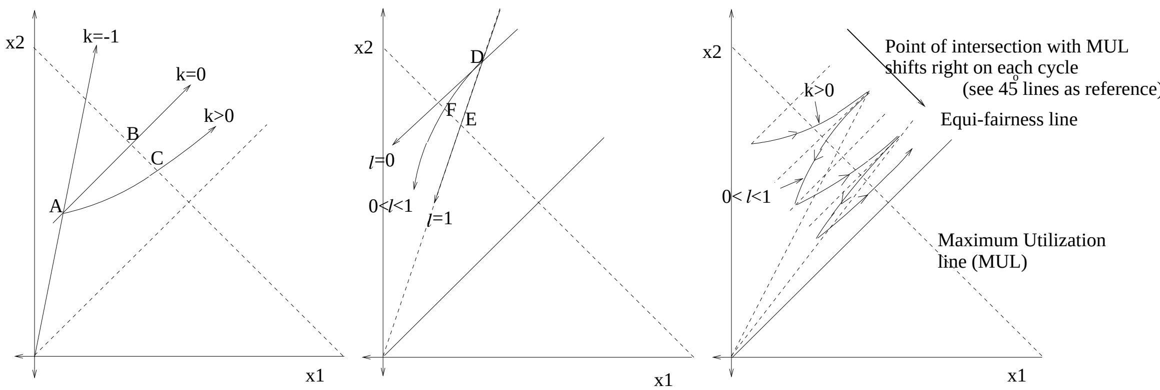

Figure 3-2: Sample path showing the convergence to fairness for an inverse increase proportional decrease algorithm.

Because binomial algorithms are nonlinear, their evolution in the phase plot is not always along straight line segments. Figure 3-2 shows a portion of one such sample path highlighting the increase and decrease parts. For all values of $k > 0$, the increase in x_1 and x_2 are not equal—the smaller of the two values increases *more* than the larger one. It is clear that this leads to a fairer allocation than if both sources did additive-increase by the same constant amount. On the other hand, values of $l < 1$ in the decrease phase *worsen* fairness. However, binomial algorithms still convergence to fairness as we show in Section 3.2.3.

3.2.2 The (k, l) space

The parameter k represents the aggressiveness of probing, while l represents the conservativeness of congestion response of a binomial control algorithm. A small value for k implies that the algorithm is more aggressive in probing for additional bandwidth, while a large value of l implies that the algorithm displays large window reductions on encountering congestion. Thus, it would seem that there is a trade-off between k and l in order for for a binomial protocol to achieve a certain throughput at some loss rate.

Indeed, in Section 3.2.4, we show that at any loss rate, the throughput depends on the sum of the two exponents, $k + l$. As a corollary, we find that a binomial algorithm is TCP-friendly if and only if $k + l = 1$ and $l \leq 1$. We call this the $k + l$ *rule,* which represents

[1]For a network with n connections each with a share x_i of a resource, the fairness index $f = (\sum x_i)^2/(n \sum x_i^2)$ [21].

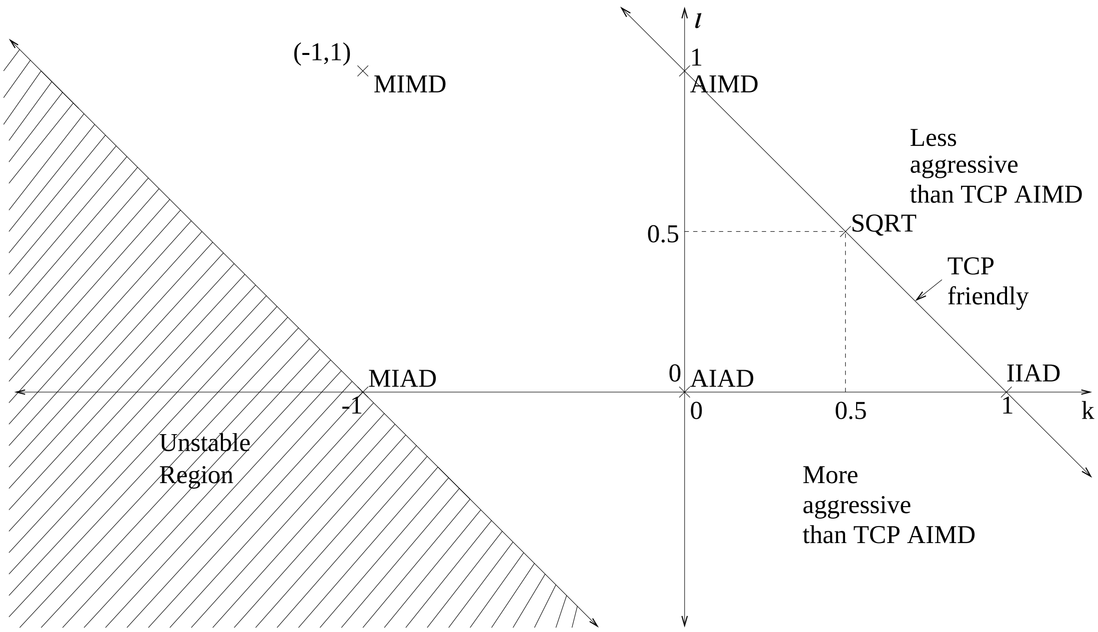

Figure 3-3: The (k, l) space of nonlinear controls from our family, with the $k + l = 1$ line showing the set of TCP-compatible controls.

a fundamental tradeoff between probing aggressiveness and the responsiveness of window reduction. Figure 3-3 shows the features of the (k, l) space. Schemes for which $k + l \leq -1$ are unstable because λ does not decrease with increasing p in this realm. As a result, the window for a connection will not decrease as loss rate increases.

3.2.3 Convergence to fairness

In this section, we show that a network with two sources implementing the same binomial control algorithm with $k, l \geq 0$ converge to a fair and efficient operating point ($x_1 = x_2 = 1/2$), provided that $k + l > 0$. The argument can easily be extended to a network of $n > 2$ sources by considering them pairwise. We assume that the network provides synchronized feedback about congestion to all the sources[2]. While this does not model all the details of Internet congestion, this analysis does provide good insight into the results that follow.

Without loss of generality, suppose $x_1 < x_2$, which corresponds to points above the $x_2 = x_1$ equi-fairness line in Figure 3-2 (an analogous argument can be made when $x_2 < x_1$). First, consider the left-most picture that shows how a window increase evolves. When $k = 0$, the increase is additive, parallel to the 45^o-line (along line AB). When $k > 0$, the increase curve lies below the $k = 0$ line since the amount of increase in x_1 is larger than the corresponding increase in x_2. Therefore, it intersects the maximum-utilization line $x_1 + x_2 = 1$ at a point C, to the *right* of where the $k = 0$ line intersects it. Such an increase improves efficiency, since $x_1 + x_2$ increases, and moves towards a fairer allocation (i.e., towards the intersection of the equi-fairness and maximum-utilization lines).

[2]This is the same network model as in Chiu and Jain's work [8].

Now, consider a window reduction. Observe that when $l = 0$ (additive decrease), the window reduction occurs along the 45^o-line (along line DE), worsening fairness. When $l = 1$, the decrease is multiplicative and moves along the line to the origin without altering fairness. For $0 < l < 1$, the window reduction occurs along a curve with the shape shown in the middle picture of Figure 3-2; this curve is in-between the previous two lines ($l = 0$ and $l = 1$ lines) and causes the system to evolve to an under-utilized region of the curve where $x_1 + x_2 < 1$. This curve lies strictly below the $l = 0$ line because the tangent at each point has a slope $= x_2^l/x_1^l > 1$ when $x_2 > x_1$. Therefore, it intersects the maximum-utilization line at a point F that is closer to the fair allocation point than the previous intersection of the sample path with that line.

The key to the convergence argument is to observe that the successive points of intersection of a binomial evolution curve with the maximum-utilization line $x_1 + x_2 = 1$ always progress toward the fair allocation point. When $x_2 > x_1$, this continually moves downwards, and when $x_2 < x_1$, it continually moves upwards towards the $x_2 = x_1$ point. Once $x_1 = x_2$, a binomial algorithm behaves exactly like a linear algorithm, moving on the $x_1 = x_2$ equi-fairness line.

It is easy to see that all we require in the above argument is for at least one of k and l to be larger than zero, since the sample path needs to move to the right at some stage. When $k = l = 0$, the algorithm is the linear additive-increase/additive-decrease scheme, which does not converge. The window evolution here remains on the 45^o-line passing through any point (x_1, x_2), without moving toward the fair allocation point.

This proof is valid under the synchronized feedback assumption and shows that a network in which all sources implement the same binomial control algorithm converges to a fair operating point. We note that it does not address the case of different binomial algorithms coexisting in the same network.

3.2.4 Throughput

We now analyze the throughput of a binomial algorithm as a function of the loss rate it experiences. We start with the *steady-state* model studied for TCP by Lakshman and Madhow [23] and Floyd [16]. Using the increase rule of Equation 3.2 and using a continuous fluid approximation and linear interpolation of the window between w_t and w_{t+R}, we get

$$\frac{dw}{dt} = \frac{\alpha}{w^k.R} \qquad \Rightarrow \frac{w^{k+1}}{k+1} = \frac{\alpha t}{R} + C \tag{3.3}$$

where C is an integration constant.

The functional form of this curve is shown in Figure 3-4. We are interested in two parameters marked in the figure: T_D, the time between two successive packet drops, and N_D, the number of packets received between two successive drops. Both these are independent of "time-shifting" the curve along the horizontal (time) axis, which implies that one can arrange it such that a downward extrapolation of the curve passes through the origin. That is, without loss of generality and with no change to T_D and N_D, one can set $C = 0$.

Let W_m be the maximum value of the window w_t at time t_2 (Figure 3-4), at which a packet drop occurs signifying congestion to the sender. Then, one can write expressions for

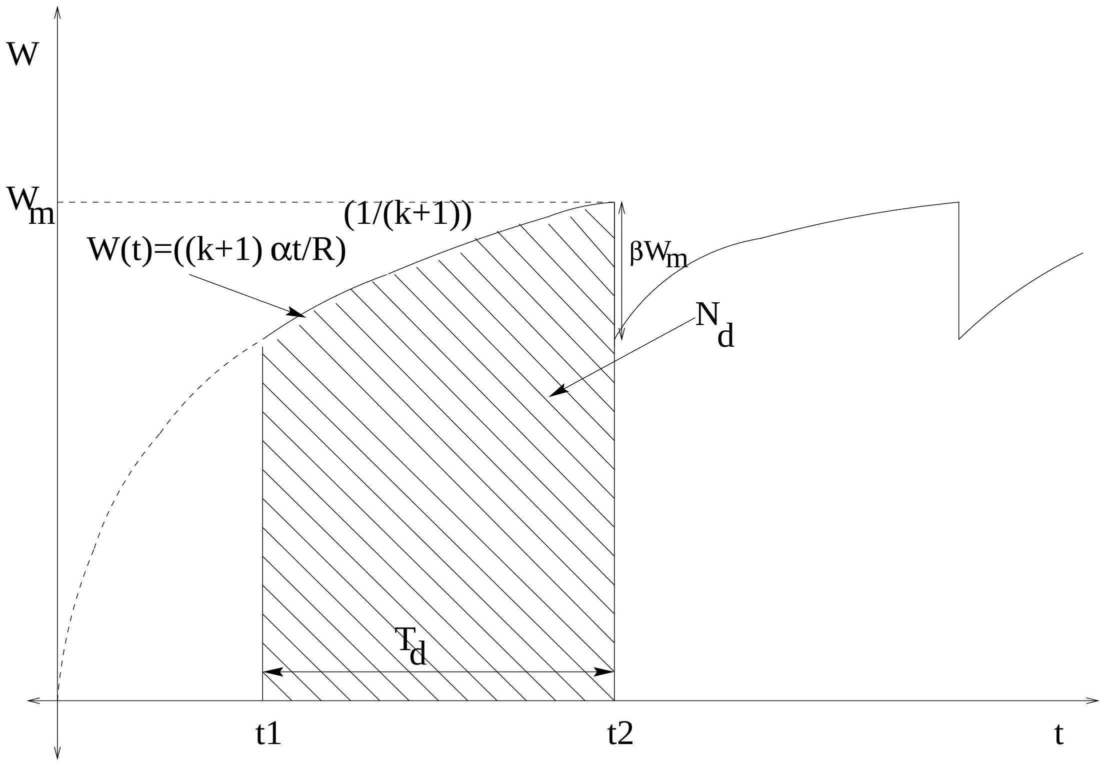

Figure 3-4: Functional form of window vs time curve.

T_D and N_D. Substituting $w_{t_2} = W_m$ and $w_{t_1} = W_m - \beta W_m^l$ in Equation 3.3, we get

$$\begin{aligned} T_D &= t_2 - t_1 \\ &= \frac{R}{\alpha(k+1)}[W_m^{k+1} - (W_m - \beta W_m^l)^{k+1}] \\ &= \frac{RW_m^{k+1}}{\alpha(k+1)}[1 - (1 - \beta W_m^{l-1})^{k+1}] \\ &= \frac{RW_m^{k+1}}{\alpha}\beta(W_m^{l-1} + O(W_m^{2l-2})) \\ &\approx \frac{\beta RW_m^{k+l}}{\alpha}(\text{when } l < 1 \text{ and } \beta << W_m^{1-l}) \end{aligned} \tag{3.4}$$

The leading term in T_D therefore varies as W_m^{k+l}, with the succeeding terms becoming increasingly insignificant.

N_D is the shaded area under the curve in Figure 3-4.

$$N_D = (k+1)^{\frac{1}{k+1}} \int_{t_1}^{t_2} \left[\frac{\alpha t}{R}\right]^{\frac{1}{k+1}} /R \, dt \tag{3.5}$$

Calculating the integral, we get:

$$\begin{aligned} N_D &= \frac{1}{(2+k)\alpha} W_m^{2+k} [1 - (1 - \beta W_m^{l-1})^{2+k}] \\ &\approx \frac{1}{(2+k)\alpha} W_m^{2+k} \beta (2+k) W_m^{l-1} (\text{leading term}) \\ &= \frac{\beta}{\alpha} W_m^{k+l+1} \end{aligned} \tag{3.6}$$

The average throughput (in packets per second), λ, of a flow using binomial congestion control is the number of packets sent in each epoch between successive drops (N_D) divided by the duration between drops (T_D). The packet loss probability, $p = 1/N_D$. Writing λ and p in terms of W_m by substituting the expressions for N_D and T_D yields:

$$\lambda = (\frac{\alpha}{\beta})^{1/(k+l+1)} \frac{1}{Rp^{1/(k+l+1)}} \tag{3.7}$$

Thus, $\lambda \propto \frac{1}{p^{1/(k+l+1)}}$ for a binomial algorithm. This implies that for such a binomial algorithm to be TCP-friendly, λ must vary as $\frac{1}{p^{1/2}}$, which implies that:

$$k + l = 1 \tag{3.8}$$

We call this the $k + l$ rule.

To first order, choosing α/β to be the same as for TCP would achieve similar performance. Note that in our analysis above we assumed a linear interpolation of window between w_t and w_{t+R} i.e we assumed an increase in window by one in a RTT for TCP rather than an increase by $1/w$ on the receipt of each acknowledgement.

These results also hold for the *random-loss* model first analyzed by Ott *et al.* in the context of TCP [30]. Unlike in the steady-state model where losses happen periodically when the sender's window reaches W_m, losses in the random-loss model are modeled as Bernoulli trials where each packet is independently lost with probability p.

We use the stochastic approximation technique for TCP performance described by Wang and Schwartz [44]. We treat the window value after receiving an acknowledgment with sequence number t, w_t, as a stochastic process and calculate its average value in the steady state. If we run the algorithm for a long time, the resulting congestion probability (the number of window reductions over the number of packets sent) is p. Then, in the steady state, the random process w_t evolves as follows: given w_t, with probability $(1 - p)$, the packet is not lost and the sender's window increases, so $w_{t+1} = w_t + \alpha/w_t^{k+1}$, whereas with probability p, the packet is lost, forcing the window to reduce giving $w_{t+1} = w_t - \beta w_t^l$.

Using this, we can calculate the average "drift" D in the window when $w_t = w$. $D(w) = (1 - p)\alpha/w^{k+1} - p\beta w^l$. Assuming that the stochastic process w_t has a stationary distribution, w_t must have most of its probability around the region for $w = W_{steady}$ such that $D(W_{steady}) = 0$. Thus:

$$\frac{(1-p)\alpha}{W_{steady}^{k+1}} = p\beta W_{steady}^l \tag{3.9}$$

$$\Rightarrow W_{steady} \approx (\frac{\alpha}{\beta p})^{1/(k+l+1)} \text{ if } p << 1 \tag{3.10}$$

We emphasize that p is the per-packet loss probability. As shown in the steady-state analysis above, W_{steady} is a good approximation to the time average of the random process w_t. The result, that $\lambda \approx (\frac{\alpha}{\beta})^{1/(k+l+1)} \frac{1}{Rp^{1/(k+l+1)}} \propto \frac{1}{p^{1/(k+l+1)}}$ therefore follows for the random-loss model as well.

This relationship establishes the fact that for a given loss rate and identical conditions, TCP AIMD and a binomial algorithm satisfying $k+l$ rule can achieve the same throughput. Furthermore, it shows that for a given loss rate, two binomial connections will achieve same throughput.

Note that our framework deals only with the congestion avoidance algorithms in a protocol like TCP, and does not deal with mechanisms like slow start, timeouts and loss recovery. These mechanisms are also important, but they are orthogonal to the study of congestion avoidance algorithms. In later chapters, we shed some light on these issues as well.

We note that varying α and β in AIMD congestion control also helps in reducing oscillations. However, this continues to maintain a multiplicative reduction and as a result, the same values of α and β cannot be used across a wide range of bandwidth-delay product paths by an application that desires an absolute bound on the inherent oscillations due to congestion control algorithm. To achieve this, (for example, if the layers in layered media are additively spaced), α and β have to be functions of current window values which is what binomial algorithms help achieve.

3.3 IIAD and SQRT Congestion Controls

In this section, we discuss two particular TCP-friendly binomial control algorithms that highlight the tradeoffs and advantages of different binomial algorithms. We call these algorithms IIAD and SQRT.

3.3.1 IIAD Congestion Control

IIAD (for "inverse increase/additive decrease") algorithms use the following increase/decrease equations:

$$\begin{aligned} \text{I: } & w_{t+R} \leftarrow w_t + \alpha/w_t; \alpha > 0 \\ \text{D: } & w_{t+\delta t} \leftarrow w_t - \beta; 0 < \beta < 1 \end{aligned} \tag{3.11}$$

This implies that $k = 1$ and $l = 0$ for these equations; so they satisfy the (k, l) rule.

IIAD congestion controls are interesting because they result in additive decrease, which may be beneficial for multimedia applications having additively-spaced layer encodings. Also, because of additive decrease, the variations in bandwidth caused by the congestion control to the application when the available bandwidth is constant are an order of magnitude smaller than the available bandwidth.

3.3.2 SQRT Congestion Control

The SQRT (for "square root") congestion control algorithms satisfy the following increase/decrease equations:

$$\text{I: } w_{t+R} \leftarrow w_t + \alpha/w_t^{0.5}; \alpha > 0$$

$$\text{D: } w_{t+\delta t} \leftarrow w_t - \beta w_t^{0.5}; 0 < \beta < 1 \tag{3.12}$$

i.e., both the k and l for these algorithms are 0.5. They are interesting because they represent a possible sweet spot where the magnitude of oscillations are small *and* they are reasonably responsive to changes in network conditions.

3.4 Implementation of Binomial Algorithms

We have implemented the IIAD and SQRT binomial algorithms in *ns* [29] and are available in the ns-2.1b7 release [29]. They have been implemented in CM under Linux and are available in the alpha release of the source code [42].

The *ns* implementation of binomial algorithms involves setting the k and l parameters which have been exported via Tcl. The *windowOption_* parameter needs to be set to 6 (the other options are used by various other variants of TCP already in *ns*). To use the CM implementation of the SQRT algorithm in an application, one needs to specify the type of congestion control algorithm desired through the CM API call *cm_setcongalg*. Binomial congestion controls have been implemented in *cmbin_cc.c*.

Chapter 4

Performance of Binomial Algorithms

In this chapter, we discuss the performance of IIAD and SQRT algorithms both in simulation as well as in implementation over the Internet.

4.1 Simulation results

In this section, we present the results of our *ns-2* simulations of various binomial algorithms. We start by investigating the interactions between connections running a TCP-friendly binomial algorithm (i.e., one that satisfies the $k+l$ rule) and TCP, as a function of k, which determines how aggressive the window-increase factor is. We then investigate the performance of IIAD and SQRT demonstrating the effect of these algorithms on the magnitude of oscillations in the transmission rates. We conclude this section by studying the performance of IIAD and SQRT in the presence of multiple bottlenecks.

Our single bottleneck simulations use the topology shown in Figure 4-1. It consists of n connections sharing a bottleneck link with total bandwidth equal to b, where all connections have an almost-identical round-trip propagation delay equal to RTT. There are 4 TCP connections in the reverse direction sharing the bottleneck to introduce ACK compression [11] and eliminate any synchronizations. We implemented the transport protocol by modifying the congestion avoidance algorithm used by TCP; we replaced AIMD with the binomial family. We did not modify the connection start-up or timeout routines; they continue to use slow-start and timeouts as before. Thus, the effect of slow start and timeouts on connection throughput is same as for a normal TCP connection. Each source always has data to send, modeled using *ns*'s "FTP" application. In all our experiments, we simulated each topology and workload ten times and calculated both the average and sample standard deviation of the observed values. The figures and graphs display this information.

We present performance results using the Random Early Drop (RED) buffer management algorithm at the bottleneck gateway [19]. The maximum queue size Q at the bottleneck was set to $b \times RTT$, the bandwidth-delay product of the path. The minimum and maximum drop thresholds (min_{th} and max_{th}) were set to $0.2Q$ and $0.8Q$ respectively, and the connections used a packet size of 1 KByte. Each connection was started at uniformly distributed random times in $[0, 2]$ seconds and throughput was calculated over the interval $t = 50s$ to $t = 200s$.

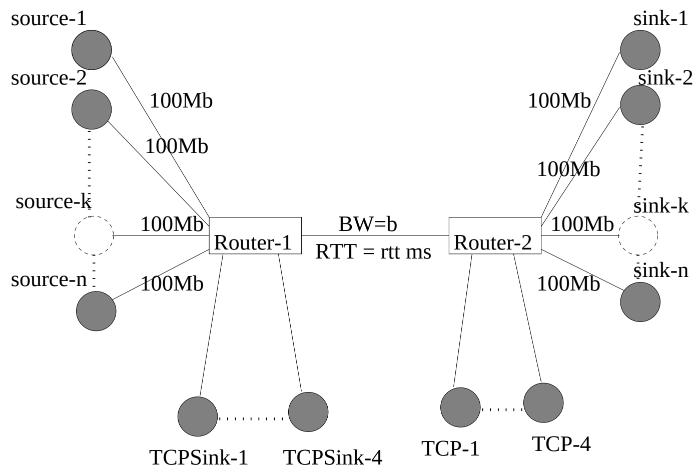

Figure 4-1: Simulation topology (delays of links for which no delay is specified are 1ms).

4.1.1 TCP-compatibility

Our first set of results (Figure 4-2) show how binomial algorithms interact with each other and with TCP. To study the effect of k and l on TCP, we simulated two connections ($n = 2$), one TCP and the other a binomial algorithm parametrized by k. We show three sets of results corresponding to the three cases $k + l$ equal to, less than, and greater than 1. For these simple scenarios, these results validate the $k + l$ rule for TCP-friendliness 3.2.4, since the long-term throughput for the binomial algorithms for which $k + l = 1$ are close to that of TCP.

4.1.2 IIAD and SQRT Performance

While IIAD is less aggressive than AIMD in the rate at which it probes for bandwidth ($k = 1$), it only reduces its window by a constant amount upon congestion ($l = 0$). We choose the values of α and β such that the theoretical throughput of IIAD is close to the throughput of TCP AIMD. There are an infinite number of values for α and β corresponding to this; we pick one pair, $\alpha = 1.5, \beta = 1$ (we use $\beta = 1$ to reduce the errors in window adjustment caused by *ns-2*'s requirement of integral window values. Although the analysis given in this paper suggests $\alpha = 2$ and $\beta = 1$, a more detailed analysis considering timeouts gives values close to $\alpha = 1.5$ and $\beta = 1$.

We compare the fairness of IIAD relative to another IIAD connection and to a TCP/Reno connection sharing the same bottleneck, using the topology and workload in Figure 4-1. In these experiments, $n = 10$, with five connections of each kind, and each connection was started at a random time in $[0, 2]$ seconds. Each experiment was conducted using a bottleneck bandwidth $b = 10$ Mbps and a round-trip time RTT between 10ms and 640ms.

Figure 4-3 plots the throughput ratio for two IIAD connections on one curve and for one IIAD and one TCP connection on the other. These results show that IIAD is fair to both the IIAD and to TCP across a wide range of RTT values. At very large RTTs, the standard deviation of results increases because the time for which simulations are run becomes small in terms of the number of round-trips for which the connections are active.

We also plot the effect of varying packet loss-rates in the Internet on the fairness of IIAD

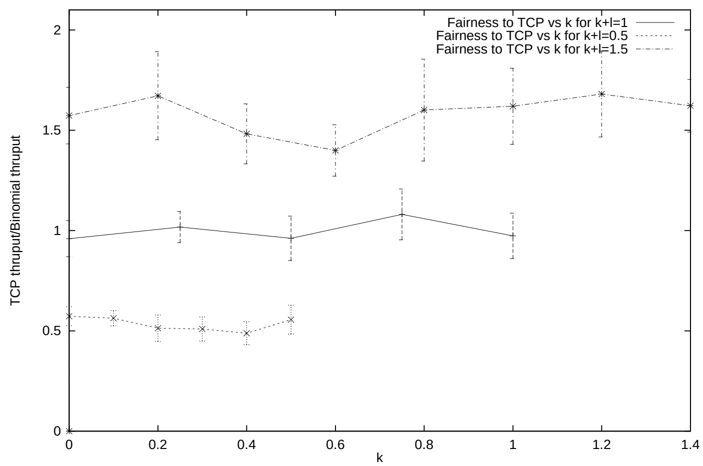

Figure 4-2: Ratio of the throughput of TCP AIMD to the throughput of a binomial algorithm, as a function of k. The algorithms that satisfy the $k + l$ rule are the ones for which this ratio is closest to unity. When $k + l = 0.5$, the binomial algorithm obtains noticeably higher throughput relative to TCP, while when $k+l = 1.5$, TCP obtains significantly higher throughput relative to binomial algorithm. The error bars show the 95% confidence interval of the throughput ratio. In these experiments, $b = 3$ Mbps and $RTT = 50$ ms.

algorithms in Figure 4-4. This curve also shows that IIAD connections are fair to TCP and to themselves across a wide range of loss-rates. At very high loss-rates, TCP gains because of its more aggressive probing for bandwidth, and it is able to recover faster from burst losses. Similar results were obtained with SQRT algorithm as well.

We now consider the impulse-response behavior of the binomial algorithms. Our experiences with these experiments across several binomial algorithms have convinced us that slow-start (or a similar mechanism) is an important component of any practical protocol to ensure that a connection converges relatively quickly, within a few RTTs to close to the fair value. We show an example of this in Figure 4-5, which shows how slow-start enables a new TCP connection to catch up and share bandwidth with a long-running IIAD connection.

These results also hold when the number of connections increases. Figure 4-6 shows the window variation for five of fifteen concurrent IIAD flows sharing the bottleneck link. At time $t = 20$ seconds, a TCP AIMD connection starts (the impulse at $t = 20$), and is able to grab its share of bandwidth even in the presence of several other long-lived IIAD connections, as can be seen from the TCP window evolution after about $t = 25$ seconds. Similar results hold for IIAD and SQRT connections in the presence of TCP connections.

4.1.3 Reduction in oscillations

In this section, we demonstrate the effectiveness of TCP-friendly binomial algorithms in reducing bandwidth oscillations. The loss-rate at the bottleneck link was switched between 25% and 0.5%. The low loss-rate period lasted for 50s while the high loss-rate period lasted for 1s. We study the effect of this loss pattern on TCP AIMD, IIAD, SQRT, and AIMD algorithm with increase/decrease parameters set to $\alpha = 0.31$ and $\beta = 0.125$ (AIMD algorithm with increase/decrease parameters different from TCP are henceforth called generalized-

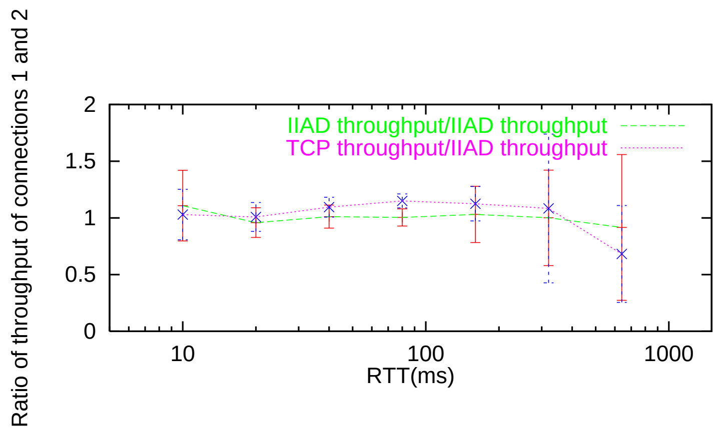

Figure 4-3: Ratio of throughputs of two connections sharing the bottleneck along with 95% confidence intervals ($n = 10, b = 10$ Mbps).

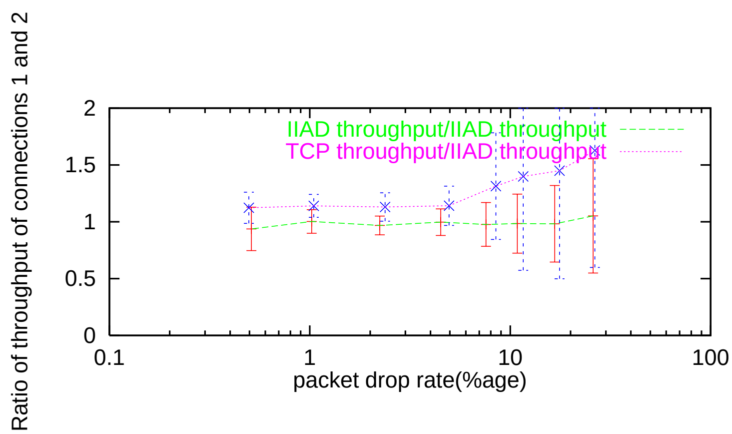

Figure 4-4: Ratio of throughputs of two connections sharing the bottleneck along with 95% confidence intervals ($n = 10, rtt = 100ms$).

AIMD algorithms) in Equation 1.

Figure 4-7 shows the packet drops and the transmission rate, averaged over 0.2s, 1s, and the entire connection, of a TCP AIMD connection for this loss pattern. As expected, the TCP AIMD algorithm shows a noticeable amount of oscillation in transmission rate even when the loss rate is constant. On the positive side, it responds fast to changes in loss rate. Figures 4-8 and 4-9 show the same results for IIAD and SQRT. IIAD shows negligible oscillations, but is slower to respond to bandwidth changes. We also plot the oscillations resulting from using SQRT and GAIMD. These results show the tradeoffs between increase/decrease rules, where slower oscillations also result in slower response in general. The relative merits of these various slowly responsive congestion control algorithms, including a comparison to equation-based approaches, is explored in Chapter 5.

4.1.4 Multiple connections and bottlenecks

This section investigates the impact of scale on binomial algorithms along two dimensions: (i) increasing the number of concurrent connections across a single bottleneck, and (ii) investigating performance across multiple bottleneck links.

To understand how several connections using different TCP-friendly binomial algorithms interact with each other, we simulate several concurrent connections running different al-

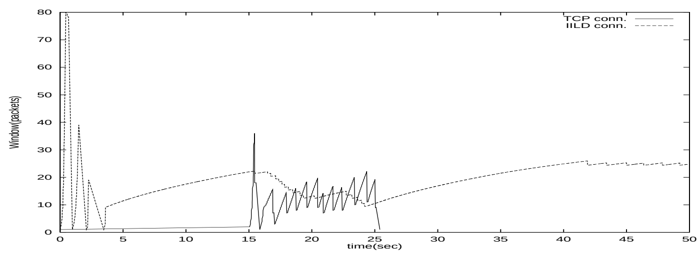

Figure 4-5: Response of a long-lived IIAD flow to a TCP impulse. In this experiment, $b = 1.5$ Mbps and $RTT = 50$ ms.

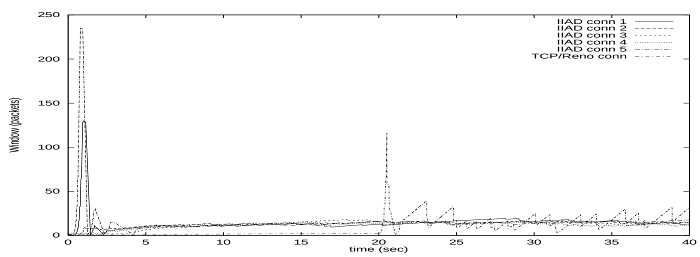

Figure 4-6: A TCP AIMD connection grabs its fair share of bandwidth in the presence of many long-lived IIAD flows ($n = 16, b = 9$ Mbps, $RTT = 50$ ms). For clarity, we only show the window sizes of five IIAD connections and the TCP connection.

gorithms sharing the bottleneck. The topology we use is shown in Figure 4-1 with $b = 50$ Mbps and $RTT = 50$ ms. We choose values of $k = \{0, 0.25, 0.5, 0.75, 1\}$ and $l = 1 - k$, and vary the total number of connections n. For each value of k, we set up $n/5$ connections, and start each connection at a random time in the interval $[0, 2]$ seconds. In Figure 4-11, we plot the mean value of the fairness index (ten runs for each point) along with 95% confidence intervals.

To study the impact of multiple bottlenecks and background traffic on the performance and fairness of binomial algorithms, we simulated the topology shown in Figure 4-12. The maximum number of HTTP connections for each HTTP source was set to 5 and all other parameters were set to the default values from *ns-2* for the HTTP and CBR sources and sinks. The window variation for the TCP AIMD and IIAD sources are shown in Figure 4-13. As this figure shows, the bottleneck bandwidth gets distributed fairly among these sources even in the presence of multiple bottlenecks. We observed the same behavior for other sources in this simulation and also when we replaced IIAD with SQRT.

4.2 TCP-friendliness vs TCP-compatibility

We now study the interactions between binomial algorithms and TCP AIMD over a drop-tail bottleneck gateway, observing some surprising effects. Figure 4-14 shows the window variation and bottleneck buffer occupancy for two connections, one TCP AIMD and the other IIAD, sharing a drop-tail bottleneck gateway. We see that TCP starts losing out and

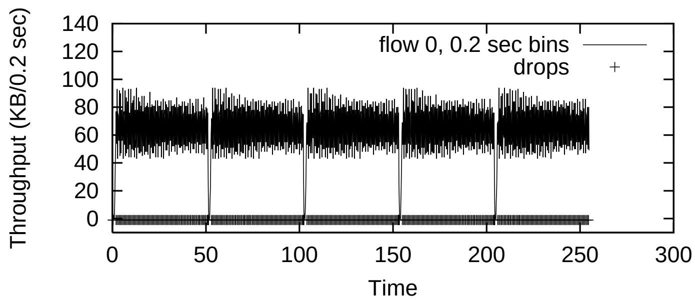

Figure 4-7: A TCP flow responding to periods of low loss-rate interspersed with periods of high loss-rate.

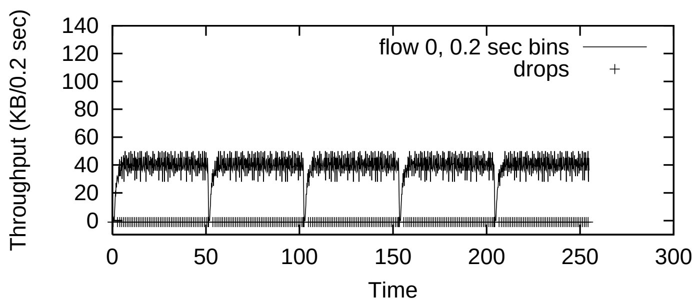

Figure 4-8: An IIAD flow responding to periods of low loss-rate interspersed with periods of high loss-rate.

its window keeps decreasing until it starts to oscillate below its fair share because no buffers are available to it. On the other hand, IIAD starts grabbing more and more bandwidth.

At first, we found this result puzzling because the theory and the $k+l$ rule had predicted that as long as $k+l=1$, the long-term throughput of a binomial algorithm would be equal to TCP AIMD. However, closer examination of the bottleneck buffer occupancy revealed the problem. In a congested network, the “steady state” of a drop-tail bottleneck queue is not empty. IIAD is less aggressive than AIMD, and when it reduces its window, does not completely flush the queue. When a drop-tail gateway has been configured with a queue size of $b \times RTT$, it ensures that TCP-style “factor-of-two” multiplicative decrease brings the reducing connection's contribution to the bottleneck occupancy down to (or close to) 0. This allows other competing connections to ramp up and also ensures that sufficient buffers are available for the window to increase before another “factor-of-two” reduction happens. In contrast, a non-AIMD TCP-friendly binomial algorithm, by its very design, ensures that window reductions are not drastic. As a result, it ends up with more than its fair share of the bottleneck, and a window reduction does not flush all of its packets from a queue optimized for TCP-style AIMD. In fact, the competing AIMD window oscillates as if it sees buffers equal to the additive-decrease term (the amount of buffer freed on a reduction) of the IIAD algorithm. The result is that the loss-rates observed by the two flows

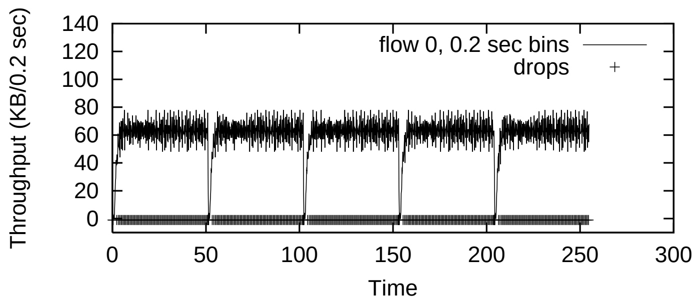

Figure 4-9: A SQRT flow responding to periods of low loss-rate interspersed with periods of high loss-rate.

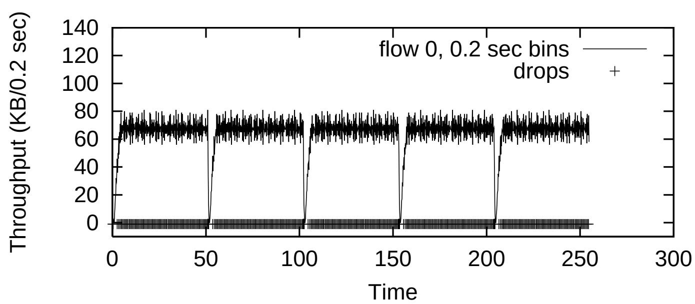

Figure 4-10: A generalized AIMD flow responding to periods of low loss-rate interspersed with periods of high loss-rate.

competing at a drop-tail bottleneck are not equal. This argument also shows how buffer provisioning is intimately tied to the window adjustment algorithm of the end-systems for drop-tail gateways.

In contrast, RED gateways are designed to accommodate bursts and maintain small average queue sizes by providing early congestion indications. They seem ideally suited to binomial algorithms because they do not tie buffer sizing closely to the precise details of window adjustment of the end-points. Instead they vary the drop rate as a function of queue size making all flows see the same drop rate. This is yet another among the many other compelling reasons for the Internet infrastructure to move to a more active queue management scheme like RED.

We do not view the TCP-unfairness of the binomial algorithms across drop-tail gateways as a deployment problem: first, the binomial algorithms obtain *better* throughput than TCP AIMD with drop-tail gateways, which augurs well for applications using them (and also provide an incentive for Internet Service Providers to move to active queue management schemes)! Second, any scalable scheme for detecting flows using more than their share of bandwidth would likely use an active queue management scheme and not a drop-tail gateway, which would ensure that true fairness to TCP is achieved. We emphasize that the adverse interactions of the binomial algorithms with TCP are primarily a consequence of

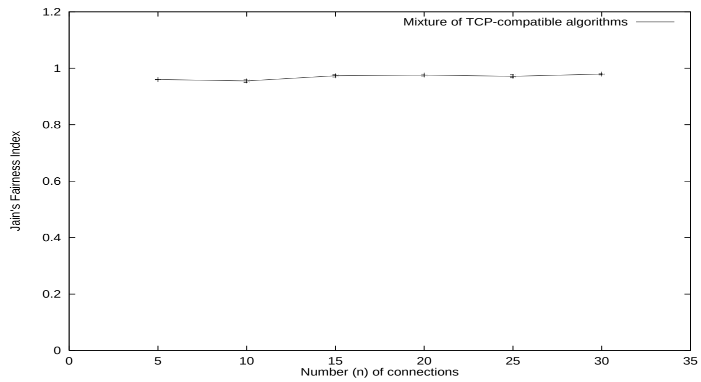

Figure 4-11: Plot showing Jain's Fairness Index as a function of the number of TCP-compatible connections sharing a bottleneck. In each experiment, the total number of connections was divided equally into five categories corresponding to $k = 0, 0.25, 0.5, 0.75, 1$. In these experiments, b = 50 Mbps, RTT = 50 ms. The (tiny) error-bars show 95% confidence intervals.

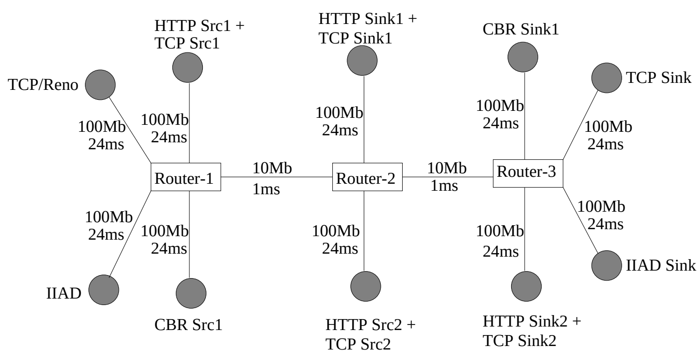

Figure 4-12: Topology with multiple bottlenecks and background traffic.

the adverse effects of drop-tail queue management.

4.3 Implementation

We implemented the SQRT congestion control algorithm in the Linux 2.2.9 kernel as part of the Congestion Manager (CM) [4] to provide congestion-controlled UDP sockets. We experimented with the Internet audio conferencing tool, *vat*, in unicast mode. Figure 4-15 shows the congestion window variation for a transfer as a function of 10ms time intervals for an audio session between two Linux machines. These machines were on the same LAN but had a pipe of bandwidth 50Kbit/s and RTT 900ms between them, configured using Dummynet [14]. The figure shows the effectiveness of SQRT congestion control in alleviating the large TCP-style "factor-of-two" reductions. The magnitude of oscillations are smaller than what AIMD would observe.

Figure 4-16 shows the congestion window variation for a *vat* transfer between two Linux

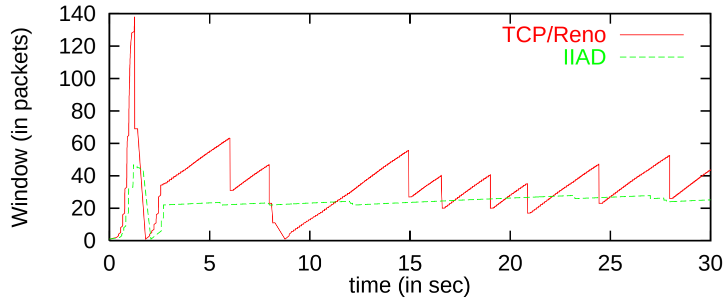

Figure 4-13: Window variation vs. time for the topology with multiple bottlenecks.

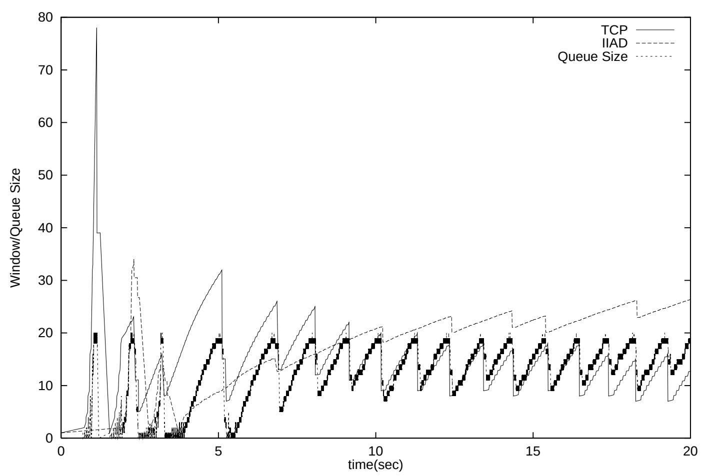

Figure 4-14: Plot showing window/queue size variation for TCP/Reno and SQRT algorithms sharing the bottleneck with drop-tail gateways($b = 3$ Mbps, $RTT = 50$ ms).

machines, one at MIT and the other at University of California, Berkeley. Again, the magnitude of oscillations are much smaller than with AIMD. The window keeps increasing because the bandwidth available between these two machines was much higher than the 64Kbps, rate at which *vat* samples audio data. This graph also demonstrates the working of SQRT across the Internet, since the occasional reductions are not drastic.

One concern for the large-scale deployment of IIAD algorithms in the Internet may be that their relatively mild reduction in window on experiencing congestion may affect Internet stability. However, we believe that the primary threat to the Internet stability comes not from the flows using some form of TCP-compatible congestion control but from flows that do not use any congestion control at all. Moreover, the prevention of congestion collapse *does not* require that flows reduce their sending rate by half in response to a single congestion indication. Furthermore, our protocols do reduce their transmission window to one upon persistent congestion (e.g., on a timeout), which helps during times of extreme congestion. We believe that it is important to have such mechanisms in any congestion control derived from TCP-friendly binomial controls. The aim of this paper has been to

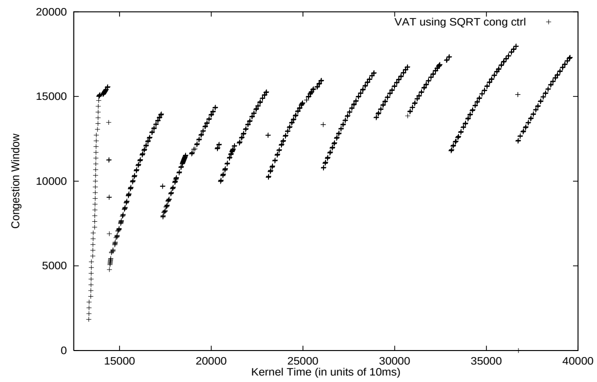

Figure 4-15: Window variation for a *vat* session using SQRT congestion control with a bottleneck configured using Dummynet ($b = 50$ Kbps,$RTT = 900$ ms).

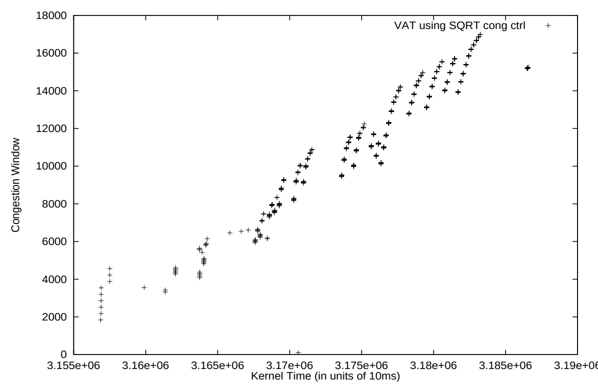

Figure 4-16: Window variation for a *vat* session using SQRT congestion control across an Internet path.

present binomial controls as a viable option; further real-world experimentation needs to be done to engineer these controls before widespread deployment can occur.

4.4 Summary

In this chapter we presented and evaluated a new family of nonlinear congestion management algorithms, called *binomial algorithms.* They generalize the familiar class of linear algorithms; during the increase phase, $w_{t+r} = w_t + \alpha/w_t^k$ and on experiencing a loss , $w_{t+\delta t} = w_t - \beta w_t^l$. We showed that a network with sources running the same binomial algorithm converges to fairness under a synchronized-feedback assumption if $k+l > 0$ and at least one of k or l is positive, and that the throughput of a binomial algorithm $\lambda \propto 1/p^{\frac{1}{k+l+1}}$, where p is the loss rate it encounters. As a corollary, a binomial algorithm is TCP-friendly if and only if $k + l = 1$ and $l \leq 1$ (the $k + l$ rule).

The $k + l$ rule represents a fundamental trade-off between probing aggressiveness and congestion responsiveness, with small values of l being less drastic in window reduction. Hence, we believe that binomial algorithms with $l < 1$ are well-suited to applications like audio and video that do not react well to drastic multiplicative decrease. Our preliminary experiments seem to justify this hypothesis, although more validation and research is needed before widespread deployment can be recommended. For applications that simply want to transmit as much data as quickly as they can without worrying about the degree of rate variations while doing so, the $k + l$ rule shows that AIMD is a very good strategy. Of all the TCP-friendly binomial algorithms, AIMD is the most efficient in aggressively probing for bandwidth. Our simulation results showed good performance and interactions between binomial algorithms and TCP across RED gateways.

Chapter 5

Observations on Slowly Responsive Congestion Control

In this chapter, we explore the general dynamics of TCP-compatible congestion control mechanisms with smoother sending rates and slower responses to congestion that that of TCP AIMD. These include the benefit of slower changes to the sending rate, as well as the costs of a slower response to persistent congestion and a limited ability to take advantage of a sudden increase in the available bandwidth.

We define a congestion control mechanism as *TCP-equivalent* if it displays congestion control behavior that, on time scales of a single round-trip time or longer, is roughly equivalent to that of TCP. TCP-equivalent congestion control mechanisms include TCP, rate-based versions of TCP such as RAP [37], and receiver-based variants of TCP like TEAR [39]. This chapter explores the implications of congestion control mechanisms that are not TCP-equivalent, but that are still TCP-compatible, that is, that compete fairly with TCP under steady-state conditions.

We define *SlowCC*, or slowly responsive congestion control, as TCP-compatible congestion control mechanisms with a slower response to a single packet drop than that of TCP. This slower response to individual packet drops allows applications using SlowCC to benefit from a smoother sending rate than that of TCP. Slow congestion control mechanisms include binomial congestion control algorithms, equation-based congestion control and generalized AIMD. We define *VerySlowCC* as SlowCC that takes more than ten round-trip times to increase its sending rate by one packet/RTT.

Equation-based congestion control uses the loss event rate measured over some interval of time to determine its transmission rate. In order to be TCP-compatible, equation-based congestion control can use the TCP response function characterizing TCP's sending rate as a function of the loss event rate and round-trip time. EBCC(k) denotes TCP-compatible equation-based congestion control that computes the average loss event rate over the most recent k loss intervals; TFRC [18] roughly corresponds to EBCC(6). Following [17], we let *AIMD(a, b) congestion control* refer to pure AIMD congestion control that uses an increase parameter a and a decrease parameter b. That is, after a loss event the congestion window is decreased from W to $(1-b)W$ packets, and otherwise the congestion window is increased from W to $W+a$ packets each round-trip time. We define AIMD(b) as AIMD with a decrease parameter of b, and an increase parameter a chosen so that AIMD(b) is TCP-compatible. We define TCP(b) as TCP using AIMD(b) instead of the traditional AIMD(1/2). AIMD $(4(2b-b^2)/3,\ b)$ can therefore be expected to be TCP-compatible. Thus, we use TCP(b) to refer to AIMD with a decrease parameter of b and an increase parameter of $4(2b - b^2)/3$.

AIMD(b) congestion control mechanisms have been compared with TCP and with TFRC in [17] which considers the tradeoffs between smoothness, responsiveness, and aggressiveness for TCP-compatible congestion control. Here *smoothness* is the largest reduction of the sending rate in one round-trip-time in the deterministic steady-state scenario, *responsiveness* is the number of round-trip times for sustained congestion to halve the sending rate, and *aggressiveness* is the maximum increase in sending rate in one round-trip time. [17] shows that TCP(1/8) and TFRC are similar in responsiveness and aggressiveness, while TFRC ranks higher than TCP(1/8) in smoothness.

5.1 Issues with slow congestion controls

A key reason to explore SlowCC (Slow congestion control) is to allow applications, if they so desire, to choose a slower congestion control mechanism with a smoother sending rate than that of TCP. Section 5.2 explores the smoothness benefits of SlowCC.

SlowCC is ideally suited to a steady-state environment with a fixed-bandwidth pipe, no routing changes, and no changes in the levels of congestion, and in such an environment would have few disadvantages. However, in an environment such as that of the current Internet, with changes in both the underlying available bandwidth and the level of congestion from competing traffic, SlowCC's performance is far less certain. SlowCC's slow response to congestion, or to the absence of congestion, can result in the following disadvantages, either for the SlowCC flow or for competing traffic in the network:

- The loss of throughput for the SlowCC flow, and possibly also for the aggregate traffic, from SlowCC's slow response to a sudden increase in the available bandwidth.
- High packet drop rates, for both the SlowCC flow and any traffic competing in a FIFO queue, from SlowCC's slow response to a period of persistent congestion.

These potential weaknesses of SlowCC are explored in Sections 5.3 and 5.4 respectively.

Note that the two disadvantages discussed above are of concern even in an environment where all of the traffic uses the same congestion control mechanism. However, for a congestion control to be deployed in the current Internet, there is an additional concern of fairness with TCP and with other competing congestion control mechanisms. While TCP-compatibility guarantees fairness with TCP in the long run under steady-state conditions, it does not necessarily ensure fairness in the short run, or even in the long run over changing network conditions. Section 5.5 addresses the following additional concern about SlowCC:

- The possibility of unfair competition with TCP, either over the short-term, or over the long-term with a rapidly-changing environment.

One of the main questions regarding SlowCC is determining the conditions under which SlowCC can be safely deployed in the global Internet. Here "safety" includes both fairness with TCP, including with the *web mice* of short TCP connections, and the avoidance of unnecessary periods of high packet drop rates (with the resulting loss of aggregate throughput if dropped packets have used scarce bandwidth on a congested link). Answering this question of the safety of SlowCC is one of the main goals of this chapter.

5.2 Smoothness Benefits of SlowCCs

This section illustrates the smoothness of SlowCC, relative to TCP, and explores the differences in smoothness between different algorithms. [17] has shown that, for a steady-state environment with a perfect, periodic packet drop rate, EBCC has a perfect smoothness metric of 1, in that the sending rate is held steady. In contrast, AIMD(b) congestion control has a smoothness metric of $1-b$, in that the sending rate in steady-state is still reduced by a factor of $1-b$ in response to a packet loss. Binomial congestion control algorithms have a smoothness metric which is dependent on the available rate. If the available bandwidth is B, then an IIAD algorithm with decrease metric b will have a smoothness of b/B while for SQRT, it is $b/\sqrt{B}$.

However, with the more irregular packet loss patterns typical of an actual Internet path, the relative smoothness of various congestion control algorithms become considerably more complex. This section explores the relative smoothness of AIMD(1/2), EBCC(8), IIAD, SQRT and AIMD(1/8) when subjected to a range of packet loss patterns.

Figure 5-1 shows that EBCC and IIAD are considerably smoother than TCP or AIMD(1/8) even with a mildly bursty loss pattern. These graphs show EBCC, IIAD, SQRT, AIMD(1/8), and TCP with a repeating loss pattern of three loss intervals of 50 packet arrivals followed by a packet drop, followed by three loss intervals of 400 packet arrivals followed by a packet drop. For each simulation there is a single flow responding to this pattern of packet drops. For each graph, the solid line shows the sending rate averaged over 0.2-second intervals, and the dashed line shows the sending rate averaged over one-second intervals. At the bottom of each graph is a mark for each packet drop.

Because EBCC estimates the loss rate over the most recent eight loss intervals, the EBCC flow in this simulation maintains a relatively stable estimate of the loss event rate, and as a consequence maintains a fairly smooth sending rate. In contrast, the TCP and AIMD(1/8) flows each have a burstier sending rate. IIAD reduces its window only additively and when the bandwidth becomes available, it increases window slowly. Thus, it achieves smoothness at the cost of throughput. SQRT, on the other hand, is less smooth than IIAD but it achieves higher throughput.

[17] defines the smoothness metric as the ratio between the highest and lowest sending rates in consecutive RTTs. We extend this metric to define the smoothness over t seconds of a flow as the ratio between the highest and lowest sending rates over t-second intervals. Thus, the smoothness metric from [17], defined for a steady-state environment with a periodic packet drop rate, corresponds to smoothness over a round-trip time in the new definition. From Figure 5-1, the smoothness over 0.2 s of EBCC, excluding after the initial start-up, is 1.28, the smoothness over 0.2 s of IIAD is 1.6, the smoothness over 0.2 s of SQRT is 1.73, the smoothness over 0.2 s of AIMD(1/8) is 2.27, and the smoothness over 0.2 s of TCP is 4.0.

In contrast to Figure 5-1, Figure 5-2 shows a packet drop pattern that illustrates a weakness of EBCC, compared to AIMD(b) congestion control. It shows that while EBCC is quick to replace a memory of good times (i.e., low congestion) with a later memory of bad times, EBCC can be quite slow in allowing a period of good times to supplant an earlier memory of bad times. As we explain below, this differential response to good and bad times is not an inherent property of EBCC, but instead comes from the specific method used by EBCC in estimating the loss event rate.

For each simulation in Figure 5-2, there is a repeating pattern a one-second heavy-congestion phase with a 25% packet drop rate, with every fourth packet dropped at the

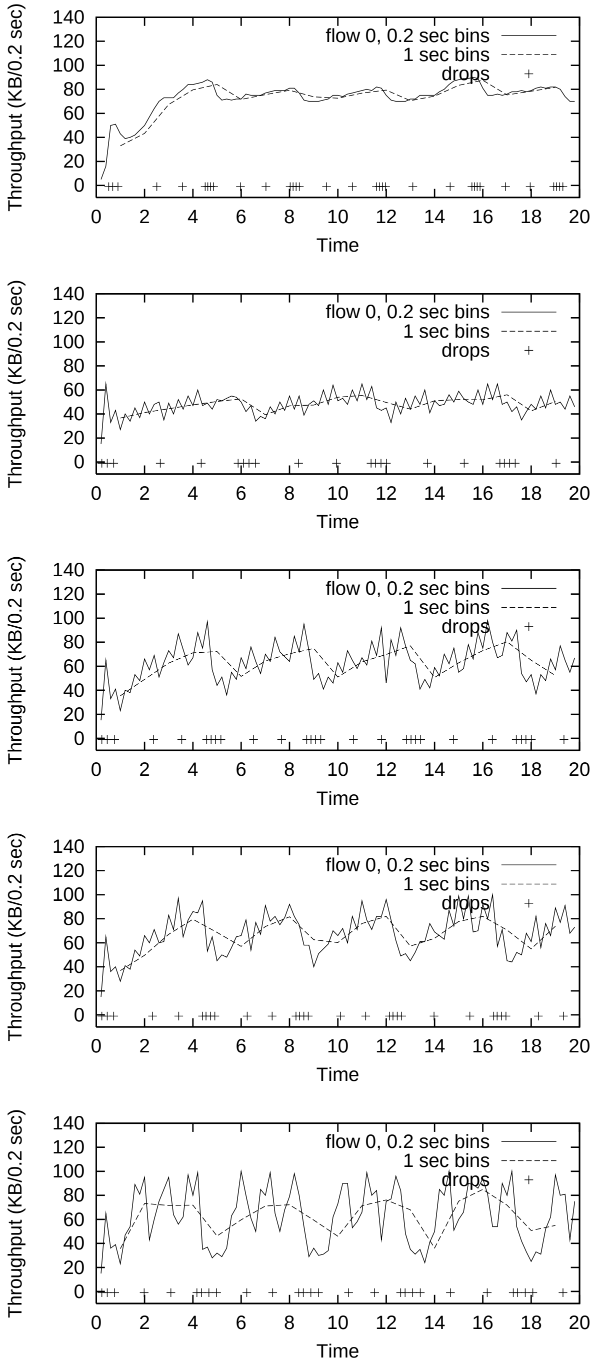

Figure 5-1: EBCC (top), IIAD (upper middle), SQRT(middle), AIMD(1/8) (lower middle), and TCP (bottom) with a mildly bursty loss pattern.

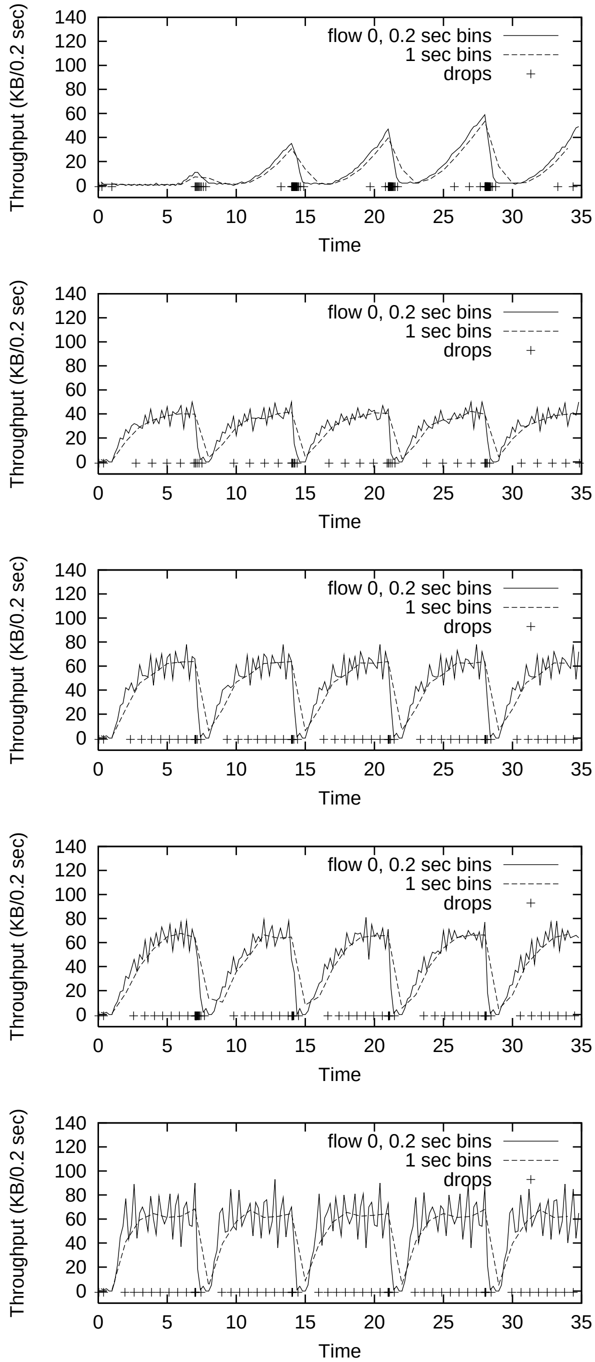

Figure 5-2: EBCC (top), IIAD (upper middle), SQRT (middle), AIMD(1/8) (lower middle), and TCP (bottom) with a more bursty loss pattern.

router, followed by a six-second low-congestion phase with a 0.5% packet drop rate, with every 200-th packet dropped at the router. The heavy-congestion phase lasts for roughly twenty round-trip times, sufficiently long in terms of loss intervals that at the end of the heavy-congestion phase, the EBCC flow has lost all memory of the earlier low-congestion phase. However, the EBCC flow is slow to ramp up during the uncongested phase, so the uncongested phase contains at most three long loss intervals, and this is not enough to totally supplant the memory of the previous congested phase. Thus, for EBCC eight short loss intervals from a congested phase quickly replaces the memory of an earlier uncongested phase, while a few long loss intervals from an uncongested phase do not completely replace the memory of an earlier congestion phase. An equation-based congestion control mechanism that used other methods for calculating the loss event rate would have different properties, in terms of the response to the onset and absence of congestion.

Figure 5-2 implies that if this EBCC flow were competing with the TCP flow in this environment with by a one-second heavy-congestion phase followed by a six-second low congestion phase, the TCP flow would receive the overwhelming share of the link bandwidth. This issue of relative fairness in environments with rapidly-changing congestion is explored in detail in Section 5.5.

5.3 The cost of a slow increase in the sending rate

As discussed in Section 5.1, the slow increase rate of a SlowCC can result in a loss of throughput for SlowCC, relative to TCP, when there is a sudden increase in the bandwidth available to a flow.

We define a metric $f(k)$ to characterize this loss in throughput as follows:

$f(k)$ = throughput (as fraction of available throughput) achieved by a congestion control algorithm in the first k RTTs after the bandwidth available to it is doubled.

For an AIMD algorithm with increase parameter a and round-trip time R, $f(k)$ can be approximated as follows; assume the available bandwidth for that flow is doubled from λ to 2λ packets/sec. By definition, the sending rate of AIMD with increase parameter a increases by a/R packets/sec. Consider a sender that was sending at exactly the available bandwidth when the available bandwidth was doubled. After k round-trip times, for $ka/R \leq \lambda$, with no packet drops and insignificant queue build-up, the instantaneous bandwidth of the TCP flow is approximately $\lambda + ka/R$.

Therefore, the average throughput θ achieved by the flow in the k round-trip times can be approximated by:

$$\begin{aligned}\theta &= ((\lambda + a/R) + ... + (\lambda + ka/R))/k \\ &= \lambda + (k+1)a/(2R)\end{aligned}$$

As a fraction of the available throughput 2λ, this becomes

$$f(k) = 1/2 + (k+1)a/(4R\lambda) \tag{5.1}$$

A similar analysis for binomial algorithms gives:

$$f(k) = 1/2 + (k+1)a/(4R\lambda^{k+1}) \tag{5.2}$$

Similarly, for EBCC, after an initial delay (for the new loss interval sample to be greater than the average loss interval), the increase rate is a function of the amount of history maintained by the EBCC sender. A similar analysis can be done to derive $f(k)$ for EBCC(b). Neglecting the initial delay in EBCC's increase, the increase δ in the sending rate per round-trip time is given by

$$\delta = 1.2\sqrt{\lambda + 3 * 1.2\sqrt{\lambda}/4b} - \sqrt{\lambda} \quad (5.3)$$

$$\approx 0.54/b \quad (5.4)$$

$f(k)$ for EBCC(b) then is given by

$$f(k) = 1/2 + (k+1) * 0.54/4bR\lambda \quad (5.5)$$

For EBCC(256), the increase rate is only 0.002 packets/RTT.

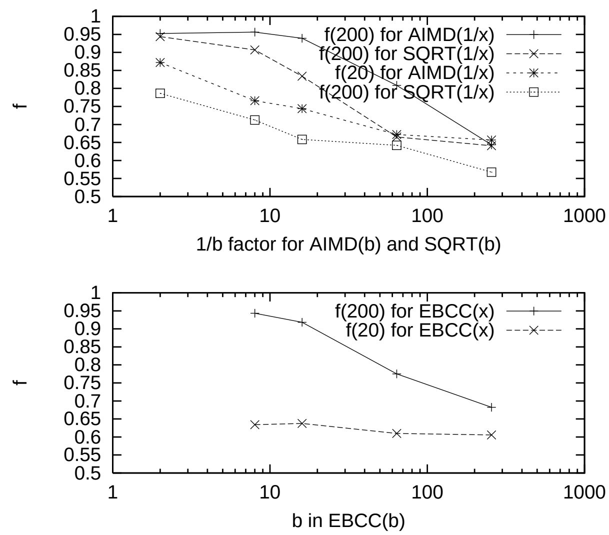

Figure 5-3: $f(20)$ and $f(200)$ for various SlowCCs

To experimentally evaluate f(k), we simulate a dumbbell-shaped topology with a RED gateway at the bottleneck with min_thresh set to $0.25 * bw * delay$, max_thresh set to $1.25 * bw * delay$, q_{weight} set to 0.002 and $bufsize$ set to $2.5 * bw * delay$. The bottleneck bandwidth is 10 Mbps. Initially, 10 identical flows are started. The flows use different algorithms in different simulations. At time 500 sec., after the flows have reached steady state, 5 flows are stopped, effectively doubling the bandwidth available to the remaining 5 flows. The total throughput as a fraction of the available bandwidth achieved by the SlowCC flows in the next 20 and 200 RTTs are observed and plotted in Figure 5-3 for AIMD(b) and SQRT(b) (top) and EBCC(b) for various values of b. We observe that while TCP achieves about 86% utilization in 20 RTTs after the bandwidth is doubled, SlowCCs receive far less utilization. AIMD(1/8) and EBCC(8) achieve 77% and 63% utilization respectively while SQRT achieves 77%. EBCC(8) achieves lower utilization than AIMD(1/8) or SQRT(1/8) because of the delay in its response to increases in the available bandwidth (as it takes some time for EBCC to flush its memory of low bandwidth period). Even slower congestion controls like AIMD(1/256), SQRT(1/256) and EBCC(256) achieve much lower utilization. AIMD(1/256), SQRT(1/256) and EBCC(256) receive close to 60% utilization and hence, perform quite badly on f(20) metric. We also observe that as k is increased, $f(k)$ becomes higher as more time is available for with clearing the memory for SlowCCs. However, for very SlowCC like AIMD(1/256), SQRT(1/256) and EBCC(256), f(200) is very low (close to 65%) compared to more responsive algorithm.

The above simulations showed the effect of sharp increase in bandwidth on SlowCCs. However, even in an environment with both sharp increases and decreases in bandwidth, SlowCCs receive lower throughput than a TCP connection would in the same scenario. This is because of their slower responsiveness to network conditions. Thus, even though SlowCC by definition would receive roughly the same throughput at TCP in an environment with a fixed, steady-state packet drop rate, this is not true for an environment with changing network conditions.

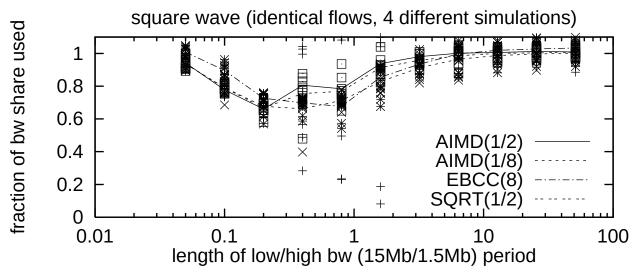

Figure 5-4: Effect of varying bandwidth on link utilization.

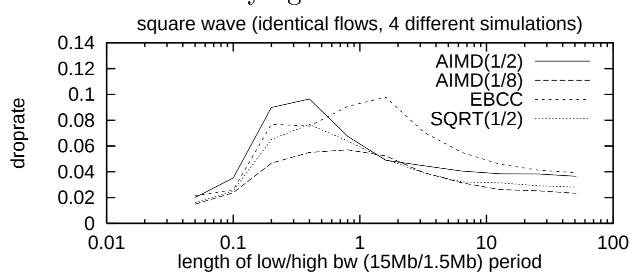

Figure 5-5: The drop rate.

To study this loss in throughput for SlowCC in an environment with changing network conditions, we use a simulation scenario with ten identical congestion-controlled flows competing with an ON/OFF CBR source with fixed, equal ON and OFF times. The bandwidth available to the congestion-controlled flows alternates between 15Mb and 5Mb (i.e., a 3:1 ratio). We do not pretend that this is a realistic scenario; however, this simple scenario provides useful insights into the dynamics of TCP and of SlowCC under dynamic network conditions.

Figure 5-4 shows the effect of the changing available bandwidth on the throughput of AIMD(1/2), AIMD(1/8), SQRT(1/2) and EBCC(8). The x-axis of Figure 5-4 shows the length of the ON and the OFF periods for the competing CBR flow in seconds, and the y-axis shows the throughputs of the congestion-controlled flows, as a fraction of the average available bandwidth (i.e., total bandwidth minus average CBR sending rate). The throughput of each congestion-controlled flow as well as the average throughput are plotted.

As we can see from Figure 5-4, the period of the competing CBR flow has a significant impact on the throughput of the congestion-controlled flows. For example, when the CBR flow has ON and OFF times of 50 ms, the congestion-controlled flows receive high throughputs, whether they are AIMD(1/2), AIMD(1/8), SQRT or EBCC. This shows that short bursts of competing traffic are not harmful to AIMD, SQRT or EBCC performance, and that these short bursts can be effectively accommodated by the active queue management at the congested router. In contrast, when the CBR flow has ON and OFF times of 200 ms, four times the round-trip time, a congestion-controlled flow receives less than 80% of the overall available bandwidth, whether the flow is using AIMD(1/2), AIMD(1/8), SQRT(1/2) or EBCC(8).

Figure 5-5 shows the packet drop rate for the simulations in figure 5-4. The drop rate varies depending on the congestion control mechanism used in the simulation, and depending on the period of the competing CBR flow. For some periods for the CBR flow, TCP has higher drop rates than EBCC, while for other periods of the CBR flow, the reverse is true. However, the drop rates when AIMD(1/8) are sharing the bottleneck are never higher than the drop rate with AIMD(1/2) flows. We explore this further in section 5.5.

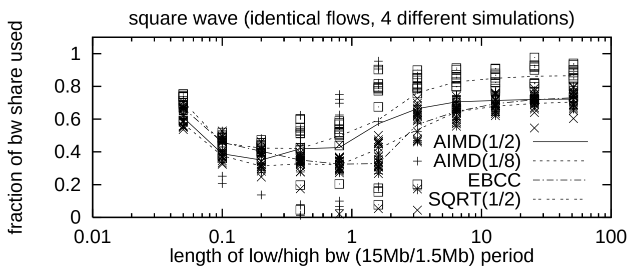

Figure 5-6: Effect of 10:1 oscillations in network bandwidth on bandwidth utilization of various congestion control algorithms.

In conclusion, even though in a steady-state environment SlowCC flows would get roughly the same throughput as would a TCP flow, this is not true for a more dynamic environment. In particular, an environment with varying load may result in lower throughput (and hence, lower link utilization) with SlowCCs than with TCPs (note that here we are considering only the case where all flows are using the same congestion control; the behavior of the network in the presence of different congestion control algorithms is explored later on in Section 5.5). This effect of reduced throughput for SlowCC is even more clear in Figure 5-6, where the available bandwidth to the congestion-controlled traffic varies by a factor of 10:1.

5.4 The dangers of a slow decrease in the sending rate

In addition to being slow to respond to increases in the available bandwidth, SlowCC is also slow to respond to reductions in the available bandwidth. This means that after a sudden increase in demand (e.g., from many new connections starting up, or from a high-bandwidth flow without end-to-end congestion control), SlowCC could result in high packet drop rates for an extended period of time. This is a grave concern, since it implies that the network with SlowCC could show persistently higher drop rates for long durations. These higher drop rates could result in transient unfairness for competing traffic, and, for a network with multiple congested links, could also result in a loss in the aggregate throughput of the network.

To study the effect of SlowCC on drop rates after a sudden increase in congestion, we run a simulation with five long-lived SlowCC flows, along with a CBR source that starts at time 0, stops at time 150, and restarts at time 180, taking 50% of the bandwidth of the

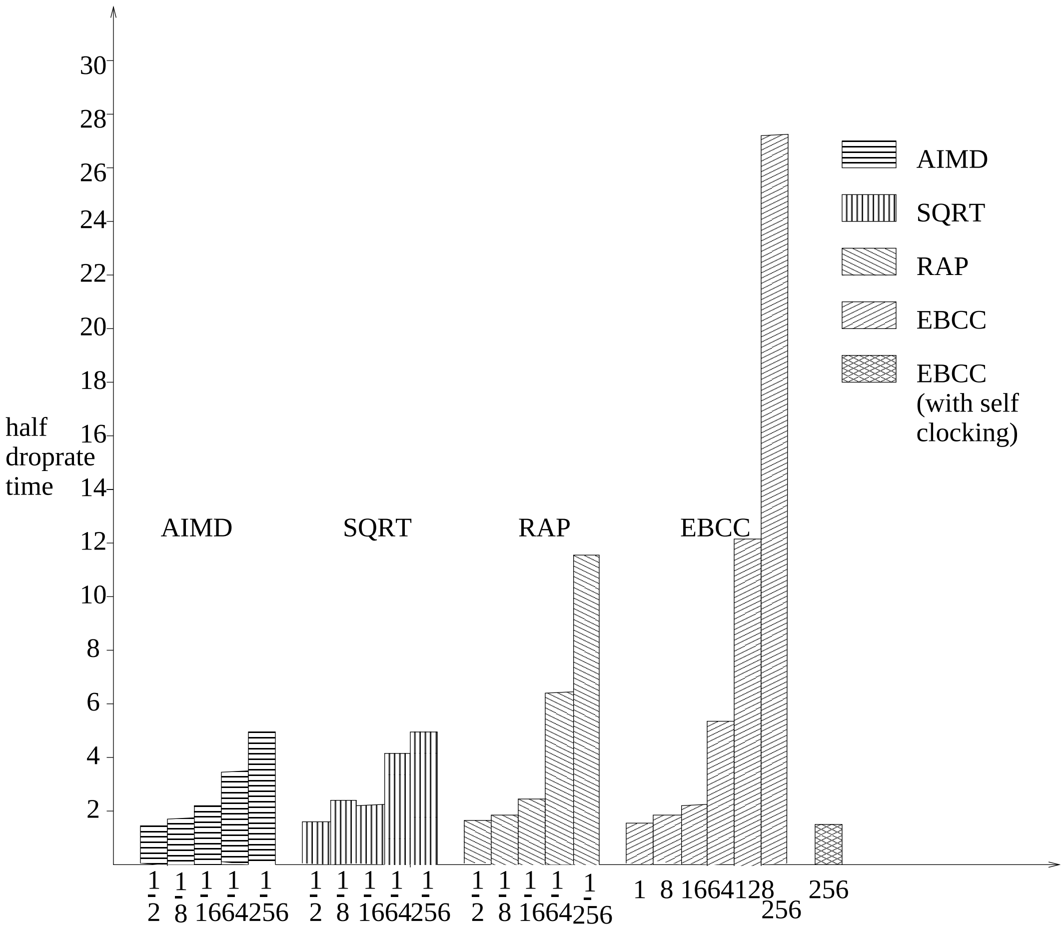

Figure 5-7: The time taken for drop-rate to reduce to 1.5 times the steady state drop-rate after sudden decrease in available bandwidth to half.

bottleneck link during its ON times. We ran separate simulations with AIMD(b), SQRT(b), RAP(b) for various values of b as well as for different EBCC algorithms. The bottleneck has suitably provisioned RED gateways, and the simulation scenario includes traffic in the reverse direction. Figure 5-7 summarizes our results.

As Figure 5-7 shows, EBCC(256) and RAP(1/256) take very long time (100s of RTTs) before the drop-rate reduces to within 1.5 times the steady state drop-rate. This shows the detrimental effect of VerySlowCC unless some other mechanisms are available. Surprisingly, we did not find persistently high drop-rates with extremely slow variants of AIMD or IIAD which seem to quickly respond to congestion and bring down the droprate to normal values within a few RTTs (response time of the order of RTO).

5.4.1 SlowCC's safeguards against persistent congestion

To understand the reasons why AIMD(1/256) and SQRT(1/256) are safe with regards to reducing the drop-rates quickly while EBCC(256) and RAP(1/256) are not, we focus on the other mechanisms (rather than just the increase/decrease algorithm) present in TCP and SQRT that are missing in EBCC and RAP. One of the key differences is that while TCP and SQRT are window based and transmit packet only on the receipt of an acknowledgement (called self-clocking from now on), EBCC and RAP are rate based and transmit data based on the rate determined by the increase/decrease algorithm irrespective of the number of acknowledgements received (though they do use the acknowledgement to update their sending rate, but data is not triggered by the acknowledgements but sent out based on the determined rate). Self clocking helps TCP and SQRT to reduce their transmission rates drastically when the available bandwidth reduces even if the window does not cut down rapidly. This is because, the acknowledgements start arriving only at the rate of the bottleneck and thus, the sending rate gets limited by the bottleneck rate in the previous RTT. This is not true for EBCC or RAP which may take some time to update their measure of available bandwidth.

To evaluate whether self clocking is really responsible for the difference seen in the behaviors of AIMD, SQRT, EBCC and RAP, we introduce self clocking in EBCC. EBCC already limits the EBCC sender's sending rate to at most twice the rate at which data is received by the receiver in the previous round trip; this is critical to prevent severe over-shooting, and emulates TCP's self-clocking during slow-start. To emulate TCP's self-clocking during congestion avoidance, we introduced a "conservative_" option which, for the round-trip time following a packet loss, limits the EBCC sender's sending rate to at most the rate at which data is received by the receiver in the previous round trip (that is, the round-trip time containing the loss). In addition, one also needs to limit the maximum amount by which sending rate can increase in congestion avoidance phase. This is because once the sending rate is reduced due to self clocking, absence of losses may cause the sending rate to jump back to high values corresponding to the memory of good times in the past. The pseudo-code is as follows:

```
if a loss is reported {
  sending rate = max (specified rate, reported received rate)
}
```

Thus, after heavy losses in the network, the conservative_ option causes the sending rate to immediately reduce to the reported receive rate.

This illustrates that the self-clocking mechanisms help VerySlowCC to remain safe (with regard to drop-rates) for deployment in the Internet by preventing sustained periods with high packet drop rates. However, these same self-clocking mechanisms exacerbate SlowCC's bias towards reducing the sending rate. Section 5.5 shows that SlowCC loses out in competition with TCP in environments with sharp changes in network conditions. As a result, they may end up loosing more bandwidth as compared to increase in bandwidth then the amount they gain on decreases in bandwidth. This may cause them to loose out in competition to TCP on varying network conditions. We will explore this in section 5.5.

5.5 Fairness with TCP

In this section, we study the possible effect of SlowCC on fairness in the Internet. We consider several related aspects of fairness: the long-term fairness between TCP and SlowCC in a changing environment; the short-term fairness between TCP and SlowCC after a sudden change in the level of congestion; and the effect of SlowCC on the start-up behavior of a competing TCP flow. We also consider fairness with VerySlowCC, to identify the limits (if any) to the acceptable slowness of response for end-to-end congestion control.

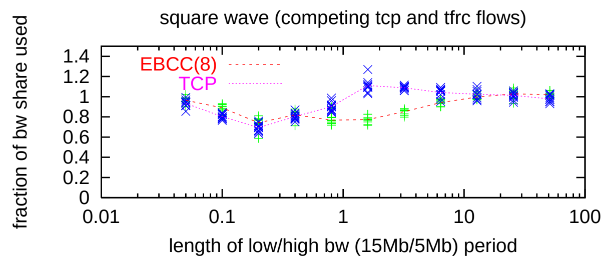

Figure 5-8: Throughput of competing TCP and EBCC(8) flows.

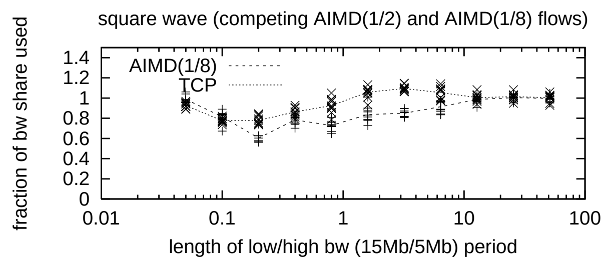

Figure 5-9: Throughput of competing TCP and AIMD(1/8) flows.

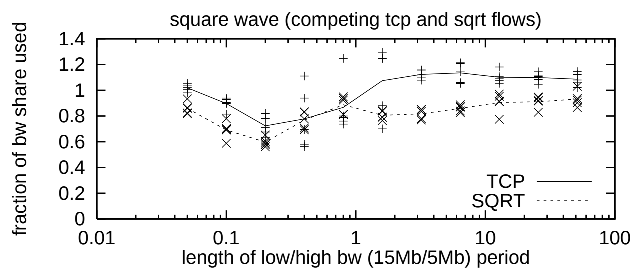

Figure 5-10: Throughput of competing TCP and SQRT(1/2) flows.

5.5.1 Long-term fairness

This section explores fairness between TCP and SlowCC in an environment with changing network conditions. We have defined SlowCC as restricted to TCP-compatible congestion control mechanisms, which have been shown in previous work to compete fairly with TCP as long-lived flows in a stable network environment.

Figure 5-8 shows the effect of varying bandwidth (with alternating high and low bandwidth periods) on the throughput of five TCP and five EBCC(8) flows sharing a bottleneck link. In the absence of queueing delays, the RTT is roughly 50 msec. The congested router uses RED, and there is some reverse-path TCP traffic as well. The bandwidth on the congested link is set to 15 Mbps for the high-bandwidth periods, and to 5 Mbps for the low-bandwidth periods, a 3:1 variation. During an extended high-bandwidth period in this scenario, we would expect the packet drop rate to be roughly 0.7%, with an average TCP congestion window of 14.4 packets.

In Figure 5-8, each column of marks shows the results from a single simulation, with a single mark for the throughput of each of the ten flows. The x-axis shows the length in seconds of a combined high- and low-bandwidth period in that simulation, and the y-axis shows the throughput, normalized by a single flow's share of the link bandwidth. The two lines show the average throughput received by the TCP and the EBCC(8) flows.

As Figure 5-8 shows, overall link utilization is high when the period of the changing bandwidth is low, while the overall link utilization suffers when the period of the changing bandwidth is 0.2 seconds. When the period of the changing bandwidth is between one and three seconds, the TCP flows receive more throughput that the EBCC(8) flows, showing that varying network conditions can favor TCP over EBCC(8). Similar behavior is shown for TCP competing with AIMD(1/8) in Figure 5-9.

An important point to note from the above figures is that, while there is a wide range of scenarios for which TCP receives more bandwidth than SlowCC, we could find no scenarios with varying bandwidths in which SlowCC receives more bandwidth than TCP. Over short periods of time, immediately after bandwidth reduction or increased congestion, SlowCC may get higher throughput than TCP. However, in the long run, the throughput gained by TCP during bandwidth increases more than offsets the throughput lost by TCP, relative to SlowCC, during bandwidth reduction. This suggests that there are no concerns about unfair competition with TCP that would prevent SlowCC from being safely deployed in the current Internet.

As Figure 5-9 shows, SlowCC variants of TCP also receive lower throughput than competing TCP traffic in a environment with periodic bandwidth changes. Because of TCP's self-clocking, even SlowCC TCP reduces its sending rate quickly in response to persistent congestion, when many packets are dropped from a window of data. In addition, retransmit timeouts can cause abrupt decreases in the effective sending rate even for SlowCC TCP. However, SlowCC TCP has no comparable mechanisms that result in a faster-than-expected *increase* in the effective sending rate. The behavior of SQRT algorithms competing with TCP is also similar (Figure 5-10).

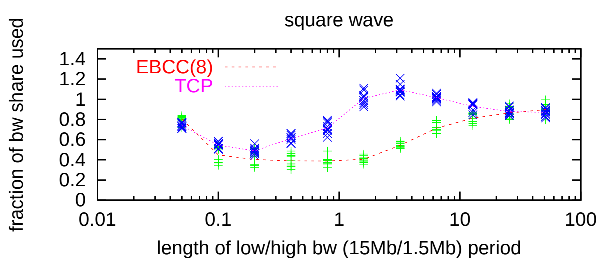

Figure 5-11: Throughput of competing TCP and EBCC(8) flows, with 10:1 changes in the available bandwidth.

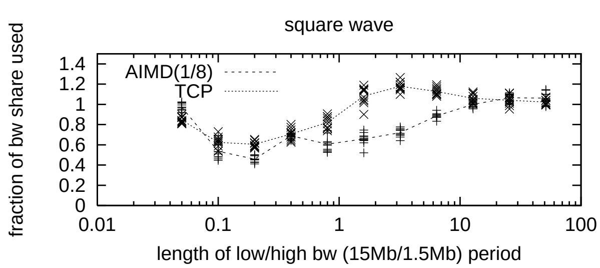

Figure 5-12: Throughput of competing TCP and AIMD(1/8) flows, with 10:1 changes in the available bandwidth).

Figures 5-11 5-12 and 5-13 show that with are 10:1 changes in the available bandwidth. the throughput difference is comparatively larger as expected (Figures 5-11 and 5-12). In a

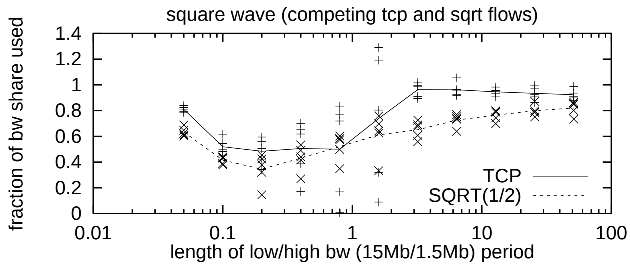

Figure 5-13: Throughput of competing TCP and SQRT(1/2) flows, with 10:1 changes in the available bandwidth).

nutshell, SlowCC loses to TCP under dynamic network conditions over the long run because their response to network conditions is slow; they do not send data fast enough when the bandwidth is actually available.

5.5.2 Transient fairness

In this section, we consider the effect of SlowCC on the short-term fairness under dynamic conditions. Short-term fairness is important because it determines the effect of SlowCC on short flows. We discuss the time for convergence to fairness for two identical SlowCCs.

Time for convergence to fairness

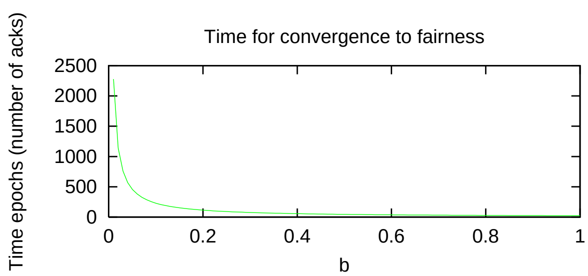

Figure 5-14: Time epochs (number of ACKs) for convergence to δ-fairness for AIMD(b) flows

In this section, we explore the convergence to fairness times for SlowCCs. We assume that the total bandwidth of the bottleneck link is B and two connections with identical RTTs share the bottleneck. Suppose both connections are running TCP-friendly AIMD(a, b) algorithm with the same values for a and b. Suppose initially, one connection has all the bandwidth and the other connection has negligible bandwidth. This may arise if one source was initially going through some other much smaller (in bandwidth) bottleneck which just vanished (due to a source sharing that bottleneck stopping transmission). So, now this connection has to compete with the other flow for the bandwidth of the new bottleneck.

We consider the following question: How long does it take for the allocation to become $(B(1+\delta)/2, B(1-\delta)/2)$ starting from $(B, 0)$ where X_1, X_2 in (X_1, X_2) denote the bandwidths of flow 1 and flow 2 respectively. We call this the "δ-fair allocation" metric.

Suppose the loss probability at the new bottleneck is p. Let us consider the bandwidth share of each of the two flows at time epochs marked by the arrival of an ACK (feedback) at either of the two flows. If the allocation for the two flows are X_1^i and X_2^i at i^{th} time epoch, then, in the $(i+1)^{th}$ time epoch, the probability that the ACK belongs to flow 1 is given by $\frac{X_1^i}{X_1^i+X_2^i}$ and the probability that the ACK belongs to flow 2 is given by $\frac{X_2^i}{X_1^i+X_2^i}$. Since p is the per-RTT congestion probability, the probability that a particular notification (acknowledgement) indicates a loss is given by Thus, at the $(i+1)^{th}$ epoch, their shares become $X_1^i + \frac{X_1^i}{X_1^i+X_2^i}(\frac{a(1-p)}{X_1^i} - bpX_1^i)$ and $X_2^i + \frac{X_2^i}{X_1^i+X_2^i}(\frac{a(1-p)}{X_2^i} - bpX_2^i)$ respectively, assuming stochastic uniform random loss model. This means that the difference in the bandwidth allocation of these two flows changes from $\rho_i = |X_1^i - X_2^i|$ to $\rho_{i+1} = |(X_1^i + \frac{a(1-p)}{X_1^i+X_2^i} - bp\frac{(X_1^i)^2}{(X_1^i+X_2^i)}) - (X_2^i + \frac{a(1-p)}{X_1^i+X_2^i} - bp\frac{(X_2^i)^2}{(X_1^i+X_2^i)}) = \rho_i(1-bp)$. Thus the total number of time epochs needed for δ-fair allocation are given by $log_{1-bp}\delta$. Figure 5-14 plots the time epochs (or feedbacks) needed for δ-fair allocation for various values of b for $\delta = 0.1$ (and $p = 0.1$; other values of p give almost identical curves).

As we can see from Figure 5-14, for values of $b >\approx 0.2$, δ-fair allocation is achieved fairly rapidly but for smaller values of b, it takes exponentially longer. This suggests that on the convergence to fairness metric, values of $b >\approx 0.2$ for AIMD are suitable while lower values are highly unsuitable. This highlights a significant drawback of choosing VerySlowCCs.

5.6 Summary

In this chapter, we studied the various SlowCC algorithms. SlowCCs achieve smoothness at the cost of throughput in dynamic network conditions. We find that SlowCCs are generally stable but packet conservation is essential to ensure that packet drop rate gets reduced rapidly on a bandwidth reduction. We also find that even though SlowCCs are statically equivalent, they may not be fair even in long term in dynamic environments like oscillating bandwidth. We also analyzed the transient fairness of SlowCCs using the δ-fairness metric which suggests that VerySlowCCs may be unacceptable.

Chapter 6

Conclusions

In this thesis, we presented and evaluated a new family of nonlinear congestion management algorithms, called *binomial algorithms.* They generalize the familiar class of linear algorithms; during the increase phase, $w_{t+r} = w_t + \alpha/w_t^k$ and on experiencing a loss , $w_{t+\delta t} = w_t - \beta w_t^l$. We showed that a network with sources running the same binomial algorithm converges to fairness under a synchronized-feedback assumption if $k+l > 0$ and at least one of k or l is positive, and that the throughput of a binomial algorithm $\lambda \propto 1/p^{\frac{1}{k+l+1}}$, where p is the loss rate it encounters. As a corollary, a binomial algorithm is TCP-friendly if and only if $k+l = 1$ and $l \leq 1$ (the $k+l$ rule).

The $k+l$ rule represents a fundamental trade-off between probing aggressiveness and congestion responsiveness, with small values of l being less drastic in window reduction. Hence, we believe that binomial algorithms with $l < 1$ are well-suited to applications like audio and video that do not react well to drastic multiplicative decrease. Our preliminary experiments seem to justify this hypothesis, although more validation and research is needed before widespread deployment can be recommended. For applications that simply want to transmit as much data as quickly as they can without worrying about the degree of rate variations while doing so, the $k+l$ rule shows that AIMD is a very good strategy. Of all the TCP-friendly binomial algorithms, AIMD is the most efficient in aggressively probing for bandwidth.

Our simulation results showed good performance and interactions between binomial algorithms and TCP, especially using RED. We also found that TCP-friendliness does not necessarily imply TCP-compatibility in a network with drop-tail gateways—a binomial algorithm like IIAD or SQRT obtains higher long-term throughput than TCP because of a higher average buffer occupancy. Active queue management schemes like RED allow binomial algorithms and TCP to interact well with each other, which may be viewed as another among many important reasons to eliminate drop-tail gateways from the Internet infrastructure.

We also studied through simulations, the potential effect of slower response congestion controls and their interactions with TCP under dynamic network conditions. We found that the slower response congestion controls lead to loss in throughput. However, with suitable mechanisms like self clocking, they do not increase the drop-rates over extended period of time. Also, various TCP-friendly congestion controls are unfair to each other over short term and in varying network conditions.

We believe that the results presented in this thesis lead to a deeper understanding of the issues involved in the increase and decrease phases of a congestion management algorithm and in the notions of TCP-friendliness and TCP-fairness. We hope that our findings will spur further research into congestion control dynamics to obtain a fundamental understanding of a future Internet with multiple coexisting congestion control algorithms and protocols. We also hope that with the availability of appropriate congestion control algorithms and the congestion manager framework, Internet video and audio streaming applications will start using them.

Bibliography

[1] Legout A. and Biersack E. W. Fast Convergence for Cumulative Layered Multicast Transmission Scheme. *Tech. Report, Eurecom, Sophia-Antipolis, France*, October 1999.

[2] M. Allman and V. Paxson. *TCP Congestion Control*. Internet Engineering Task Force, April 1999. RFC 2581.

[3] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan. System support for bandwidth management and content adaptation in internet applications. In *Proc. 2000 Symposium on Operating Systems Design and Implementation*, Oct 2000. to appear.

[4] H. Balakrishnan, H. S. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proc. ACM SIGCOMM*, pages 175–187, Sep 1999.

[5] H. Balakrishnan and S. Seshan. *The Congestion Manager*. Internet Engineering Task Force, Nov 2000. Internet Draft draft-balakrishnan-cm-03.txt (`http://www.ietf.org/internet-drafts/draft-balakrishnan-cm-03.txt`). Work in progress, expires May 2001.

[6] D. Bansal and H. Balakrishnan. Binomial Congestion Control Algorithms. In *Proceedings INFOCOM 2001 (to appear)*, April 2001. Extended version available as MIT-LCS-TR-806 from `http://nms.lcs.mit.edu/papers/cm-binomial.html`.

[7] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. *Recommendations on Queue Management and Congestion Avoidance in the Internet*. Internet Engineering Task Force, Apr 1998. RFC 2309.

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

[9] D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proc. ACM SIGCOMM*, pages 109–114, August 1988.

[10] D. Clark, M. L. Lambert, and L. Zhang. NETBLT: A High Throughput Transport Protocol. In *Proc. ACM SIGCOMM*, August 1988.

[11] D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanisms. In *Proc. ACM SIGCOMM*, August 1992.

[12] D. E. Comer and D. L. Stevens. *Internetworking with TCP/IP, Volume II*. Prentice Hall, Englewood Cliffs, N.J, 1994.

[13] Taubman D. and Zakhor A. Multipoint communication by hierarchically encoded data. *Proceedings, IEEE INFOCOM*, 1992.

[14] Dummynet. `http://www.iet.unipi.it/~luigi/ip_dummynet`, Sept. 1998.

[15] S. Floyd. *Congestion Control Principles*. Internet Engineering Task Force, September 2000. RFC 2914.

[16] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Trans. on Networking*, 7(4):458–472, August 1999.

[17] S. Floyd, M. Handley, and J. Padhye. A comparison of equation-based and aimd congestion control, February 2000. `http://www.aciri.org/tfrc/`.

[18] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. `http://www.aciri.org/tfrc/`, Jun 2000.

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

[20] V. Jacobson. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM*, pages 314–329, Aug 1988.

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

[22] R. Jain, K.K. Ramakrishnan, and D-M. Chiu. Congestion Avoidance in Computer Networks with a Connectionless Network Layer. Technical report, Digital Equipment Corp., 1988.

[23] T. V. Lakshman and U. Madhow. The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss. *IEEE/ACM Trans. on Networking*, 5(3), 1997.

[24] T. V. Lakshman, U. Madhow, and B. Suter. Window-based Error Recovery and Flow Control with a Slow Acknowledgement Channel: A study of TCP/IP Performance. In *Proc. Infocom 97*, April 1997.

[25] J. Mahdavi and S. Floyd. The TCP-Friendly Website. http://www.psc.edu/networking/tcp_friendly.html, 1998.

[26] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. In *Proc ACM SIGCOMM*, pages 117–130, August 1996.

[27] Microsoft Windows Media Player. `http://www.microsoft.com/windows/mediaplayer/`.

[28] Shacham N. Multipoint communication by hierarchically encoded data. *Proceedings, IEEE INFOCOM*, 1992.

[29] ns-2 Network Simulator. http://www.isi.edu/nsnam/ns/, 2000.

[30] T. Ott, J. Kemperman, and M. Mathis. The Stationary Distribution of Ideal TCP Congestion Avoidance. ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps, 1996.

[31] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A Simple Model and its Empirical Validation. In *Proc. ACM SIGCOMM*, September 1998.

[32] Lin J. C. Paul S., Sabnani K. and Bhattacharyya S. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, April 1999.

[33] J. B. Postel. *Transmission Control Protocol*. Internet Engineering Task Force, September 1981. RFC 793.

[34] K. K. Ramakrishnan and R. Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Transactions on Computer Systems*, 8(2):158–181, May 1990.

[35] Real Networks. `http://www.real.com/`.

[36] R. Rejaie, M. Handley, and D. Estrin. Quality Adaptation for Unicast Audio and Video. In *Proc. ACM SIGCOMM*, September 1999.

[37] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proc. IEEE INFOCOM*, volume 3, pages 1337–1345, March 1999.

[38] Balaguru N. Rhee I. and Rouskas G. N. MTCP: Scalable TCP-like congestion control for reliable multicast . *IEEE Infocom*, 1999.

[39] Rhee, I., Ozdemir, V. and Yi, Y. TEAR: TCP Emulation at Receivers - Flow Control for Multimedia Streaming. NCSU Technical Report, April 2000.

[40] J. Saltzer, D. Reed, and D. Clark. End-to-end Arguments in System Design. *ACM Transactions on Computer Systems*, 2:277–288, Nov 1984.

[41] W. R. Stevens. *TCP/IP Illustrated, Volume 1.* Addison-Wesley, Reading, MA, Nov 1994.

[42] The Congestion Manager Software. `http://nms.lcs.mit.edu/software/CM/`.

[43] L. Vicisano and L. Rizzo. TCP-like Congestion Control for Layered Multicast Data Transfer. In *Proc. IEEE INFOCOM*, 1998.

[44] A. Wang and M. Schwartz. Achieving bounded fairness for multicast and TCP traffic in the Internet. In *Proc. ACM SIGCOMM*, September 1998.