Modeling the Bandwidth Sharing Behavior of Congestion Controlled Flows

Kang Li

A dissertation presented to the faculty of the
OGI School of Science & Engineering
at Oregon Health & Science University
in partial fulfillment of the
requirements for the degree
Doctor of Philosophy
in
Computer Science and Engineering

November 2002

The dissertation "Modeling the Bandwidth Sharing Behavior of Congestion Controlled Flows" by Kang Li has been examined and approved by the following Examination Committee:

[illegible]
Dr. Jonathan Walpole
Professor
Thesis Research Advisor

[illegible]
Dr. Molly H. Shor
Assistant Professor
Thesis Research Advisor

Dr. Calton Pu
Professor
Georgia Institute of Technology

[illegible]
Dr. Wuchang Feng
Assistant Professor
Oregon Graduate Institute

Acknowledgement

I have been fortunate enough to have the help and support of a large number of people. I would especially like to thank my advisors Jonathan Walpole and Molly H. Shor, who have been extremely helpful in identifying my thesis topic, developing the ideas, publishing paper, construct the structure of this thesis, and for all other guidance throughout many years.

I would also like to thank

- The members of my thesis committee members, Professor Calton Pu and Wuchang Feng, for their insightful comments on my proposals, and the dissertation. I would like to thank Calton for his helps on many strategy points. Without his help, I would not finish this thesis.

- Charles "Buck" Krasic, Ashvin Goel that have been working with me through a long way form DSRG, DISC, to SYSL, and all the members of the system software group, Jonathan Walpole, Wuchang Feng, Wuchi Feng, David Steere, Dylan McNamee, Andrew Black, Perry Wagle, Jie Huang, Francis Chang, Mike Shrea, Jin Choi, Jim Snow and many others.

- Mark Morrissey and Jim Binkley from Portland State University for their many useful and insightful discussions about networking and other random topics.

- Rainer Koster, Tong Zhou, Yong Xu, Erik Walthinsen, Peter Geib, Anne-Francoise Lemeur, Dan Revel, Songtao Xia, Wei Tang, Wei Han, Henrique Paques and others that brought a lot of fun to my student life in OGI.

- Shanwei Cen, who recommended me to this graduate school.

Finally, I would like to thank National Science Foundation, and the Defense Advance Research Project Agency, for their financial support.

Table of Contents

List of Figures

Abstract

Modeling the Bandwidth Sharing Behavior of Congestion Controlled Flows

Kang Li

OGI School of Science & Engineering
at Oregon Health & Science University

August 2002

Thesis Advisors: Dr. Jonathan Walpole and Dr. Molly H. Shor

Multimedia applications have become increasingly popular in the Internet. TCP is the dominant Internet congestion control protocol, but it does not serve all applications well. Thus, many new congestion control protocols have been proposed recently, in particular for multimedia applications.

To ensure that these flows share bandwidth fairly with TCP flows, TCP-friendliness is proposed as a criterion for designing new protocols. Currently, the TCP-friendliness criterion is defined based on the assumption that all flows experience the same static congestion signal. However, the bandwidth sharing and congestion signal is a result of the dynamic behavior of all participating flows. The claim of this thesis is that the bandwidth sharing behavior among competing flows should be studied in a dynamical environment.

To understand a dynamic phenomenon one needs a theoretical model that adequately describes the behavior of the system being studied. In this dissertation, we propose a state-space model to study the dynamics of the bandwidth competition, in particular among AIMD-based TCP-friendly flows. It characterizes a dynamic system by a set of related state variables, which can change with time in a manner that is predictable provided that the external influences acting on the system are known. We use the model to describe the stability of bandwidth competitions, which is characterized as convergence to a dynamically oscillating limit cycle in the state space. This stability description clearly distinguishes transient and long-term fairness.

Along with the state-space modeling, we build an adaptive AIMD-based congestion control protocol that exposes its parameters to applications. This dissertation presents some example uses of this adaptive protocol to verify the results derived from the state-space model. As an example, we adjust the AIMD parameters to achieve a uniform fairness that is independent of round-trip-times.

Chapter 1

Introduction

At the heart of the success of the Internet is its congestion control behavior. The Internet serves flows in a best-effort way, which does not use bandwidth reservation mechanisms. Careful design thus is required to prevent applications from congesting the network with a heavy load. Until now, it has been the congestion control protocol at each end-host that has prevented applications from overloading the network.

A congestion control protocol limits a flow's rate to its portion of the available bandwidth and prevents it from sending too fast and causing network overload. A congestion control protocol does not know the available bandwidth a priori. Thus it has to probe the available bandwidth by increasing its rate until it detects signs of congestion, and then decreasing its rate. A congestion control protocol detects congestion by indications of packet loss (timeouts or duplicated acknowledgements) and explicit marks (with support from active queue management mechanisms [BCC+98]). We call these indications the *congestion signals.*

Transmission Control Protocol (TCP) [JK89, APS99] is the de-facto standard congestion control protocol in the Internet. It uses an *additive-increase and multiplicative-decrease* (AIMD) algorithm, which probes the available bandwidth by increasing its transmission rate additively, and responds to a congestion signal by decreasing its rate by half.

TCP, although powerful and effective, is not sufficient to satisfy applications that require different rate behaviors than TCP. For instance, TCP's behavior of cutting its rate by half upon a congestion signal could cause too much rate variation for a streaming media application that prefers a smooth rate [BBFS01, FF99]. Fortunately, TCP is not the only possible congestion control protocol. It uses one specific algorithm with specific parameters for the congestion control, whereas many alternative algorithms with different rate behaviors are available. Recently, many congestion control protocols [BMP94, CPW98, RHE99a, TCPF] have been proposed, particularly for streaming media applications in the Internet.

A congestion control protocol, depending on its manner of probing the available bandwidth and responding to congestion, produces various rate behaviors. In general, applications care about the following aspects of a congestion control protocol's rate behavior:

- *Rate Smoothness*: how large the magnitude of rate variations is and how often the rate varies.
- *Responsiveness*: how fast the congestion control protocol responds to changes in network conditions.
- *Fairness*: what the bandwidth share ratio is when it competes with other flows.
- *Average throughput*: whether there are underutilizations of resources caused by the congestion control scheme.

Typically, applications prefer a congestion control that has a smooth rate and a fast responsiveness. They also prefer a congestion control that shares bandwidth fairly with other flows while still maintaining a high average throughput. However, not all of these preferred behaviors can be achieved at the same time. For instance, a tradeoff exists between responsiveness and smoothness. When a protocol's sending rate is smooth and less sensitive to the variations in network conditions, it will be less responsive to changes in the network. Different applications have different preferences on this tradeoff based on their requirements, thus each of them could prefer a different congestion control protocol that meets its preferred rate behavior.

Although many congestion control protocols have been built to produce desired rate behaviors, an important aspect, the stability of systems composed by new protocols and TCP, has been largely ignored by most of the protocol designers. Understanding the system stability is important because many, if not all, of the above rate behaviors are defined based on the system stability. For example, responsiveness is actually the time for a system to reach a stable state upon a certain input. Without understanding a system's stability, there is no basis for discussing responsiveness. Similarly, fairness is mostly only interesting when competing flows achieve a stable state. If a system has no stable state, fairness is just a transient aspect and thus is not very meaningful as a rate behavior. Interestingly, the stable state of a system doesn't necessarily need to be a static state, it could be an oscillation through a cycle of system states. In the latter case, the fairness behavior should also include the deviations and oscillating frequencies, etc.

Besides understanding the system stability, protocol designers have the problem of predicting the rate behavior of the new congestion control protocols. Most new protocols' rate behaviors, for example the average throughput, are designed based on a static assumption that the congestion control protocol will perceive the same congestion signals no matter how it behaves. We use *statically derived protocols* to denote those designed based on the assumption that congestion signals are independent of the rate behaviors of the congestion control protocols.

Recent experiments [BBFS01, FHP00, LSW01] have shown that some of the statically derived protocols that are supposed to share bandwidth equally with TCP flows do not share bandwidth equally with TCP flows, or with each other, as predicted. Although Bansal et. al. [BBFS01] show that their statically derived protocols tend to use less bandwidth than TCP (which means they are more friendly to TCP flows) and can thus be safely deployed in the Internet, they do not erase the question of how to predict accurately the rate behavior of a congestion control protocol in a dynamic environment.

These statically derived protocols do not produce the expected rate behaviors because the rate behaviors of congestion control protocols are dynamically determined. By "dynamically determined", we mean that their rate behaviors and congestion signals are inter-dependent. The rate behaviors of a congestion control protocol are directly related to the congestion signals perceived by it, whereas its rate behaviors directly contribute to the production of the congestion signal. Because of this inter-dependency, adjusting a congestion control protocol, such as changing its parameters or using alternative algorithms, implies that the resulting rate behavior, and congestion signal behavior, must be studied in a dynamic environment. We refer to the rate behavior of a congestion control protocol as its *dynamic behavior,* because it must be characterized in a dynamic environment.

1.1 The Research Problem

The topic of this thesis is *to understand the dynamic rate behavior of congestion control protocols.* It is important to understand system stability because a lot of rate behaviors are related to it. It is also important to study the dynamic behaviors of newly developed congestion control protocols, which are designed for applications that are not served well by certain dynamic rate behavior. Any efforts to predict accurately the dynamic rate behaviors must take into consideration the interdependency between a congestion control protocol's input (congestion signal) and its output (the rate behaviors).

1.2 Our Approach

Instead of using an approach that separates the rate behaviors and the congestion signals, we study them within one *bandwidth sharing system,* which consists of a bottleneck link used by all competing flows. In a bandwidth sharing system, congestion signals are generated by the rate behaviors of all competing flows.

Our goal is to understand the stability and dynamic rate behaviors of a system. This naturally raises the need for a theoretical model that adequately describes the behavior of the system being studied. Generally, modeling work can choose either a deterministic or a stochastic approach. A deterministic approach tends to produce an accurate description of the system behavior but requires deterministic information of all the inputs. Whereas a stochastic approach requires only statistically based values, but produces only a high level description of the system.

We choose a deterministic approach because this thesis work focuses on the rate behaviors of individual flows in various time scales, rather than the aggregate behavior of many flows over a large time scale. The latter issue is also interesting but is outside of the scope of this thesis.

We model the bandwidth sharing system using the state-space modeling technique of modern control theory [B91, C86]. The goal of this modeling approach is to produce a description of the dynamic rate behaviors. In a state-space model, a dynamic system is characterized by a set of related state variables, which change with time in a manner that is predictable provided that the external influences acting on the system are known. In particular, the dynamic behaviors of a system are described by a group of differential equations and state jumps in our model. Using these mathematical descriptions of the model, we can study theoretically the system stability. Along with the mathematical description of system states, our state-space model of bandwidth sharing includes a state plot analysis that help one visualize the transient and stable behaviors.

In addition, we design a general adaptive congestion control protocol that is based on the general AIMD algorithm [LSW01, YL00]. We build simulations and real-world implementations so that we can verify the outcomes from the state-space model, which describes the rate behaviors of individual flows.

1.3 Contributions

This study of dynamic behaviors makes the following contributions:

- **State Space Model**

We design a state-space model for the bandwidth sharing system. It is the first model that defines the dynamic stability of the bandwidth competition, which is a limit cycle in the system state-space. We use this model to show that statically derived TCP-friendly protocols actually do not share bandwidth equally in dynamical environments. With this state-space model, we also derive many implications for real systems, such as the appropriate time-scale for measuring TCP-friendly fairness, the quantization effect of packet sizes, and the minimal buffering delay requirements for delay-sensitive applications.

- **Controllable Fairness**

Using the implications from the state-space model, we develop the mechanism to achieve various sharing ratios by tuning the parameters of AIMD algorithms. In particular, a uniform fairness, independent of the RTTs, can be achieved rather than the TCP-style fairness.

- **Adaptive Congestion Control**

We build an adaptive congestion control protocol that exposes the control parameters to applications rather than using a fixed set of parameters. Applications can adjust the behavior of the congestion control, such as the tradeoff between responsiveness and smoothness, based on application requirements. In addition, the current definition of TCP-friendly behavior can be satisfied by constraining the relationship between parameters.

1.4 Dissertation Overview

The dissertation is organized as follows:

Chapter 2 reviews TCP and its AIMD algorithm.

Chapter 3 describes the state-space modeling technique. It contains the definitions of system state and system dynamics in general, and it introduces the difference between static and dynamic equilibriums. It then presents the result of applying the state-space model to a system of competing AIMD-based flows. This model shows that the system has a dynamic equilibrium and describes its stability by a limit cycle in the state space. Through the model's state plots, we also investigate the impacts of various parameters on a single flow, and demonstrate different dynamics of bandwidth sharing behaviors among various competing flows. Among them, we show how the changes of a single flow's behavior over short time scales can change the behavior of the whole system over long time scales. In addition, we show a way to aggregate states of many AIMD flows to one flow, which can help us overcome the scalability limitation of keeping per-flow state in a state-space model.

Chapter 4 presents a list of implications from the state-space model for AIMD-based flows. It indicates that the time-scale of the fairness measurement must be larger than the period of the stable limit cycle. It also indicates that an AIMD flow's rate oscillations at its stable state cause a buffering delay that is quadratic to the flow's round-trip-time and is proportional to its average rate. In addition, the state-space model shows the relationship between a flow's AIMD parameters and its average bandwidth share. We use the relationship to produce a different fairness paradigm other than TCP-style fairness among competing flows.

Chapter 5 describes our work on building an adaptive AIMD congestion control protocol that exposes the AIMD control parameters to applications. By exposing the parameters of the AIMD algorithm, applications can tailor the congestion control protocol based on their requirements. With this adaptive AIMD congestion control, we conduct experiments in a controlled real-world setup to verify the implications from the state-space model. These real-world experiments verify the following aspects of the bandwidth sharing behaviors: (1) the timescale of fairness among AIMD flows, (2) the unfairness between TCP-friendly AIMD congestion control protocols, and (3) the possibility of adjusting AIMD parameters to achieve different share ratios other than TCP-style fairness.

Chapter 6 reviews some related work and addresses the differences between previous work and the work presented in this dissertation.

Chapter 7 concludes the dissertation and outlines some of our future plan.

Chapter 2

TCP Overview

The main concern of this thesis is the rate behavior of congestion control protocols. TCP is the dominant congestion control protocol that is used in almost every computer in the Internet, and thus it is important first to understand TCP and its congestion control algorithm.

TCP as a transport layer protocol has many functions, and congestion control is only one of them. The major functions of TCP are: *reliability control, flow control,* and *congestion control.*

- ***TCP Reliability Control*** provides an in-order reliable data transmission. In each TCP connection, packets are marked with increasing sequence numbers. The receiver sends acknowledgements to the sender for the packets that have arrived correctly in-order. When packet losses happen, the sender retransmits the lost packets to the receiver.

- ***TCP Flow Control*** prevents a fast sender from overflowing the receiver buffer of a slow receiver. In TCP, the receiver reports its open buffer space to the sender in each acknowledgment. The sender is only allowed to send as much data as indicated by the acknowledgements. When the receiver's buffer is full, the sender will stop sending any data until the receiver acknowledges it with new open space indications.

- ***TCP Congestion Control*** prevents a fast sender from overflowing the buffer in the bottleneck router on the path between the sender and the receiver.

In this thesis, we simply use TCP to refer to its congestion control. We review the TCP congestion control protocol and its well-known AIMD algorithm in the following sections.

2.1 TCP Congestion Control

Congestion control is imperative in order to allow the network to recover from congestion and operate in a state of low delay and high utilization. Ideally, to achieve high utilization, end systems need to send as fast as they can. However, if their sending rates exceed the network capacity, data accumulates in buffers in the network, which can cause long delay. Furthermore, routers have limited buffer space to tolerate temporary overloading. When the network becomes overloaded, the buffer in the bottleneck router starts to fill up, and eventually overflows. The network overloading stage is generally called *congestion*, which can cause packet losses and long delays to applications. Most lost packets are detected by end systems and retransmitted. But if end systems did not slow down their transmission rates during congestion, most of the bandwidth would be used to transmit packets that would be dropped before reaching the receivers. This behavior is called “pouring gasoline on fire” in a computer network [JK88]. The ideal behavior, which is also the goal of congestion control, is to keep end systems sending as fast as the network capacity for high bandwidth utilization without creating too much congestion.

TCP congestion control attempts to achieve this goal as follows. It starts with a low rate and probes for the existence of additional unused link bandwidth on its path by progressively increasing its rate. It continues to increases its rate until a congestion signal occurs. When TCP detects congestion, it reduces its rate to a “safe level” and begins probing again. In the following subsections, we review the way TCP detects congestion and limits its rate, and we discuss the algorithm TCP uses for probing and responding to congestions.

2.1.1 Acknowledgments

TCP uses acknowledgments to carry feedback information for all three control functions mentioned above.

Each time a receiver gets a packet[1], it informs the sender of the sequence number of the next in-sequence packet. The packet used to inform the sender is called an *acknowledgement*. Acknowledgments can be piggybacked on data packets when the receiver has data packets to send back to the sender.

When there are no packet reordering events or losses, the acknowledgment contains the sequence number of the packet following the one that just arrived. If there is a packet loss, the acknowledgments of later packets contain the sequence number of the lost packet, which is the sequence number of the next in-sequence packet in the data stream.

2.1.2 Congestion window

TCP limits its sending rate by controlling its *congestion window* size, which is the number of packets that may be transmitted-but-yet-to-be-acknowledged in a flow. Normally, the time between delivering a packet and receiving its acknowledgement is one round-trip-time (RTT). A TCP sender can send up to the congestion window size of data packets during one RTT. Once TCP sends out a window size of data packets, it can send new data packets only after some acknowledgements arrive. Thus the average rate of a TCP over one RTT is roughly the window size divided by the RTT.

[1] When TCP's delay acknowledgement is enabled, the TCP receiver could send one acknowledgement only after receiving multiple packets.

2.1.3 Congestion signals

In most wire-connected modern networks, packet losses due to link-level noise have become very rare because of technology improvements. Loss typically results from the overflowing of router buffers as the network becomes congested. TCP uses packet losses as an indication of congestion.

TCP detects packet losses with two mechanisms. The first one is the *timeout*. A TCP sender starts a timer when it sends a packet to a receiver. If the timer expires before the sender receives the packet's corresponding acknowledgement, TCP thinks the packet is lost. Clearly, the timeout interval should be larger than TCP's RTT. Actually, TCP adapts this interval dynamically based on its RTT estimations. In most of the TCP implementations, the timeout interval is set to the average of RTT estimations plus 4 times the deviation. A detailed study of the effect of various timeout settings is presented in [AP99].

The second way that TCP detects packet losses is through *duplicate acknowledgments*. A TCP receiver only acknowledges the sequence number of the next in-sequence packet. A packet loss causes the receiver to re-acknowledge the sequence number of the lost packet when the next packet arrives. The sender thus receives duplicate acknowledgements for the same sequence number. Since packet reordering in the network can also cause duplicated acknowledgements, TCP uses a threshold to avoid treating reordering as packet losses. Typically, TCP sets the threshold to three. Only when a TCP receives three or more duplicated acknowledgements does it consider that a packet is lost and thus generates a retransmission. This mechanism to detect packet losses is also referred as the "*Triple Duplicated ACK Hack*". Duplicated acknowledgements may detect packet losses earlier than the timeout timer. Thus detecting congestion by duplicated acknowledgements is called *Fast Retransmission*.

Notice that packet losses are not always caused by congestion. Noise, especially in wireless networks, can cause a significant amount of packet loss. Taking these random packet losses by noise as indications of congestion can significantly impact the performance of TCP congestion control. Many recent research studies have addressed this issue, but, since our focus is not on congestion detection, we assume TCP's congestion detection is adequate.

Recently, explicit congestion notifications (ECN) [RFB01] may be used to detect congestions. ECN capable routers can mark packets with a congestion notification when they experience congestion. In this way, TCP can be informed of congestion earlier and more accurately than using timeouts or duplicate acknowledgements. TCP can thus adjust its rate according to the congestion signal without getting dropped packets.

In this thesis, both indications of packet losses and explicit congestion marks are called *congestion signals.*

2.1.4 Slow Start & AIMD

Besides congestion signals and how TCP uses a congestion window to limit its rate, the remaining aspect of TCP congestion control is its dynamical window adjustment algorithms.

The basic rate control mechanisms are an exponential initialization stage called *Slow Start* and an *additive-increase-multiplicative-decrease (AIMD)* steady-state stage. An example of a TCP flow's two stages is shown in Figure 2.1.

The Slow Start is used when TCP is in the initial stage or after a timeout. During a Slow Start stage, TCP starts with an initial window size (typically one or two packets) and increases its window size by one packet upon the receipt of each acknowledgment. This behavior leads to an exponential increase[2] in sending rate.

[2] Slow Start is called "slow," compared to jumping to a fast rate immediately, but it actually accelerates very quickly (exponentially).

The AIMD algorithm is used in TCP's steady-state stage. During an AIMD stage, TCP increases its current window by one packet for each full window of data acknowledged. This is the *Additive Increase* (AI) part of AIMD. Once a Fast Retransmission (duplicate acknowledgements) happens, TCP cuts its window by half and then restarts the additive increase. The halving of window size is the *Multiplicative Decrease* (MD) part of AIMD. The procedure of cutting the window by half and then immediately going back to additive increase is also called *Fast Recovery*, which is fast compared to the alternative of cutting the window to the initial value.

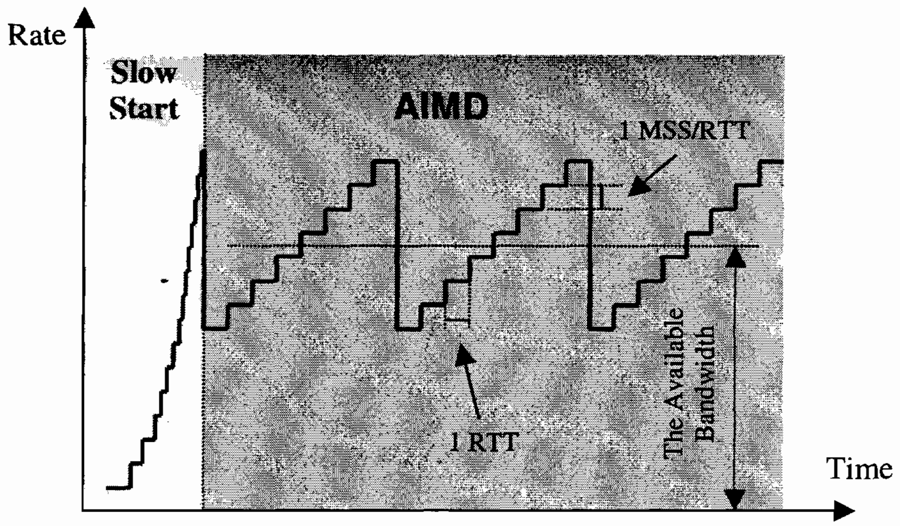


Figure 2.1: TCP Slow Start and AIMD rate control (RTT is the TCP flow's round-trip-time, MSS is the TCP flow's packet size)

The transitions between Slow Start and AIMD are controlled by a threshold and timeout events. TCP starts with a Slow Start, and once the congestion window goes across a threshold, it switches to the AIMD stage. During an AIMD stage, if a timeout event happens, TCP sets its window back to the initial value and enters Slow Start again. Both the threshold and the timeout interval are adjusted dynamically. Details can be found in [APS99].

2.1.5 TCP Flavors

TCP has evolved in the last decade, and thus many different TCP flavors are deployed in the Internet today. Here we list the key features of a few TCP flavors:

- **Tahoe**: TCP Tahoe detects congestion only by timeouts and has only Slow Start and AI stages. It starts with a Slow Start. Once the window size passes a threshold, TCP Tahoe switches to the additive increase stage. Once a timeout happens, it sets the window size back to the initial value and does Slow Start again.

- **Reno**: TCP Reno adds both Fast Retransmission and Fast Recovery to TCP Tahoe, and thus includes both Slow Start and AIMD. TCP Reno does the same thing as TCP Tahoe upon timeouts. In addition to the timeouts used in TCP Tahoe, TCP Reno also detects congestions by duplicated acknowledgements (Fast Retransmission). Upon the triple-duplicated-acknowledgements, TCP Reno cuts the congestion window by half and continues the additive increase stage (fast recovery) rather than resetting to initial windows size of one and doing slowstart.

- **New Reno**: TCP New Reno treats multiple packets losses in one RTT as one congestion signal instead of several as in TCP Reno, and thus does at most one multiplicative decrease per RTT.

- **SACK (Selective Acknowledgments)**: TCP SACK's receiver informs the sender with sequence numbers of multiple missing packets, rather than only acknowledging the sequence number of the first missing one. Thus TCP SACK can retransmit the lost packets, earlier than TCP Reno if more than one packet is lost.

- **FACK (Forward Acknowledgments)**: TCP FACK's receiver informs the sender of the highest sequence number that has arrived (even out of order). With this information, the sender can estimate accurately how many packets have left the network for this particular flow, and thus could make a better control on its window size.

- **Vegas**: TCP Vegas is generally not viewed as a member of the classic TCP family. However, it is as well-known as any of the above flavors, and we list it here. The difference between Vegas and other TCP flavors is that TCP Vegas does not rely on AIMD as the major congestion avoidance algorithm. Instead, TCP Vegas monitors the RTT of each packet and adjusts its rate to control the router queue length, which is approximately derived from the RTT measurements.

2.2 TCP-friendliness

Congestion control protocols not only prevent flows from overloading the Internet, but also determine the *fairness*, which is the bandwidth sharing ratio among them. Because the bandwidth resource is shared among all participating users, a stable bandwidth sharing fairness is desired in the heterogeneous Internet, and the case of one user gaining most of the bandwidth and starving others should not happen.

2.2.1 TCP-style Fairness

TCP's congestion control algorithm ensures that similarly situated TCP flows (same RTT, same packet size) receive roughly equal throughput. We call it *TCP-style fairness.* Notice that it does not assure equality of throughput between flows with different round-trip-times (RTT), or using different packet sizes. More than a decade of deployment of TCP in the Internet has proven that TCP congestion control maintains this TCP-style fairness across a very wide range of network environments.

2.2.2 TCP-Friendliness

Recently, many new congestion control protocols [BMP94, CPW98, RHE99a] have been proposed for applications that are not served well by TCP. This emergence of new congestion control protocols raises the issue of inter-flow fairness across different protocols. Flows using newly proposed congestion control protocols may not preserve the bandwidth sharing equality with similarly situated TCP flows.

The Internet community has struggled with this tension between preserving TCP-style fairness, and meeting the demands of applications for which TCP is a far-from-ideal solution. A recently proposed resolution is the *TCP-friendliness* paradigm [FF99, TCPF]. A congestion control protocol is called *TCP-friendly* when it uses the same amount of bandwidth on average as a similarly situated TCP flow.

2.2.3 State of the Art

Currently the notion of TCP-friendliness is defined in terms of the average throughput over a long time interval (many seconds to minutes). The cornerstone of this approach is the observation [PFTK98] that one can roughly characterize the average throughput $\bar{r}$ of a TCP flow in the presence of a constant packet loss rate p with the following equation:

$$\bar{r} = \frac{1.22MSS}{RTT * \sqrt{p}} \qquad (2.1),$$

in which, *MSS* is the flow's packet size and *RTT* is the round-trip-time.

The model is based on the following assumptions:

- TCP congestion control is working in the AIMD steady state, which means TCP is assume to detect congestions by duplicate acknowledgements. This assumption is reasonable when the packet loss rate is low. When the loss rate is high, timeout becomes the dominant mechanism for congestion detections, and a more complex throughput model [PFTK99] is needed.
- Congestion signals are independent from the sending rate, which means a congestion control protocol is assumed to experience the same congestion signal no matter how its transient rate behaves.

The detailed derivation of the throughput equation (2.1) can be found in [PFTK98].

This TCP-friendliness definition enables a wide variety of TCP-friendly congestion control protocols that can be tailored to different application requirements. The work [TCPF] has a rich collection of existing TCP-friendly congestion control protocols.

Although many TCP-friendly protocols have been proposed recently, several aspects of the bandwidth sharing behavior between various congestion-controlled flows are still unclear. Examples of these issues are: whether the bandwidth sharing among competing TCP-friendly flows is stable, how long it takes the system to converge to its stable behavior, and in what time scale the average throughput should be measured to judge a flow's TCP-friendliness, and whether transient sending rate behavior affects congestion signals. The research presented in this thesis will shed some light on these issues.

2.3 AIMD-based Algorithm

The steady-state stage of the TCP congestion control protocol uses the AIMD algorithm. TCP's AIMD algorithm is actually a specific example of an AIMD-based algorithm. A AIMD-based algorithm can be described as:

$$\begin{array}{lll} \textit{Increase (Additive):} & W(t+RTT) = W(t) + \alpha & \alpha > 0 \\ \textit{Decrease (Multiplicative):} & W(t+\delta) \leftarrow W(t) - \beta W(t) & 0 < \beta < 1 \end{array} \qquad (2.2)$$

where *W(t)* is the window size at time *t*, and *RTT* is the round-trip-time. In the absence of congestion signals, the algorithm uses the *Increase* rule in (2.2), which increases its window by a constant α in every RTT. When the congestion control detects congestions at time *t*, the algorithm uses the *Decrease* rule in (2.2), which decreases its window by a constant factor β. The new window size is denoted as $W(t+\delta)$, in which $t+\delta$ indicates the time instance just after time *t*. TCP's algorithm can be viewed as a special case of the AIMD-based congestion control algorithm with $\alpha = 1$ (packet) and $\beta = 1/2$. Throughout this paper, we use AIMD(α, β) to denote an AIMD-based algorithm using parameters α and β, and thus use AIMD(1,1/2) to denote TCP's algorithm.

For an AIMD-based algorithm to be TCP-friendly, α and β are not independent, but have to follow the relationship

$$\alpha = \frac{3\beta}{2-\beta} \qquad (2.3)$$

The derivation is based on the throughput model (2.1) with the assumption that the AIMD-based protocol will experience the same congestion signal as a similarly situated TCP flow. The derivation of this relation can be found in [LSW01, YL00] [3]. Intuitively, in order to make a flow with a smaller α parameter get the same throughput as a TCP flow, it must back off less than the TCP flow. By choosing a smaller α and a smaller β, an AIMD-based algorithm's rate varies by a smaller magnitude than a normal TCP does.

[3] In some work, the multiplicative decrease part of AIMD is described as $W(t) \leftarrow \beta W(t)$ instead of $W(t) \leftarrow W(t) - \beta W(t)$. Thus the TCP-friendly relationship is presented in a slightly different form.

Chapter 3

A State-space Model

This chapter presents our work on applying the state-space control modeling technique [B91, C86] to a bandwidth sharing system. State-space modeling is one modern control technique that has been developed to study the dynamic behaviors of systems in areas of physics, mechanics, electronics, aerospace, etc. The term "dynamic behaviors" is a familiar concept in these areas. In this chapter, we use a state-space model to capture the dynamic rate behaviors of congestion-controlled flows.

The chapter starts with a general description of the state-space control modeling technique, and then describes a state-space model for a bandwidth sharing system that is composed of congestion-controlled flows. At the end of this chapter, some examples are given of using this model to study the dynamic behaviors of bandwidth sharing systems.

3.1 State-Space Modeling

A state-space model for a system is a representation that describes the evolution of the system state, which captures the dynamic behaviors of the system.

Before explaining the details, we first introduce some terms used to describe a state-space model.

- *State Variables* – State variables [C86] can be viewed as a running collection of a system's initial conditions. Knowledge of these conditions at a given time together with a fixed explicit input is all that is necessary to specify future behaviors. For convenience of notation, we usually collect the state variables into a vector, called a *state vector*. In this document, we simply use *system state* to refer to a system's state vector.

- *Event-driven State Jumps – Events* in a system are generated based on certain conditions of the system state (for example, when a state variable exceeds a predefined value). Event-driven state jumps are state transitions that are associated with those events and can't be described as differential equations. A state jump is always associated with an event, or with a certain delay after an event.

- *State Space* – A state space is a multi-dimensional space in which each dimension represents a state variable. Thus any system state can be represented as a point in the state space[4].

- *State-Space Plot* – A state-space plot is a geometrical representation of how a system's state evolves over time in a state-space model. The solution of the differential equations and state jumps is visualized as a *trajectory* in the state space. State space plots are also called *phase plots* in the literature.

3.1.1 A State-Space Model

A state-space model consists of a set of equations describing the evolution of the system state. The future behavior resulting from a particular input can be calculated from the state-space model once the current state is known. Our state space models use differential equations and state jumps to describe the evolution of the system state. We use $x(t)$ to represent the system state at time t and $u(t)$ to represent the input to the system at time t. The derivative of $x(t)$ is represented by $\dot{x}(t)$. The differential equations are written in the form of (3.1), and state jumps are in the form of (3.2). In this mathematical representation, $h(x(t))$ is the *event trigger* function, which returns zero when no state-jump events should happen. During this period, the function $f(x(t),u(t))$ of the differential equation governs the system state evolution. Function $h(x(t))$ returns a non-zero value when some state-jump events should happen. At this instance, the function $g(x(t))$ is the function that controls the state-jumps. We use t_- and t_+ to represent the times immediately before and after the time t.

$$\begin{cases} \dot{x}(t) = f(x(t),u(t)) \quad when \quad h(x(t)) = 0 & (3.1) \\ x(t_+) \leftarrow g(x(t_-)) \quad when \quad h(x(t)) \neq 0 & (3.2) \end{cases}$$

In addition to the above mathematical representation, a state-space model can also have a geometrical representation that helps one visualize the system state transitions and shows the relationship among the state variables. The geometrical view of the system states is a state-space trajectory in a state-space plot.

An important property of the geometrical representation is the uniqueness of its states in the state-space. Since the system state is supposed to contain sufficient information about the system at any given time for its subsequent behavior to be predicted if the future input is known, it's necessary that the differential equation and state-jumps for $x(t)$ should have a unique solution for every initial state $x(t_0)$ and input $u(t)$, $t \geq t_0$. Because of this uniqueness property, there is one and only one trajectory from any given point[5] in the geometrical representation, for a particular input $u(t)$, $t \geq t_0$.

3.1.2 System Stability

Capturing a system's dynamic behavior is the goal of state-space modeling. A system's dynamic behaviors can be divided into steady-state behaviors and transient behaviors. The steady state behaviors are how a stable system behaves once it converges to its steady state, and the transient behaviors are how the system behaves on its way from an

[4] Notice here that not every point in the state space is necessary a valid state in the system.
[5] Because of the state jumps, we could have state transitions from many points to one point.

initial state to steady state. Generally, steady-state behaviors get more focus because they are the dominant behaviors if a system is stable. To study the steady-state behaviors, or even the transient behaviors, we need to first study the system's stability. In this subsection, we present a brief review of the system stability in the context of a state-space model.

3.1.2.1 Equilibrium

Equilibrium describes possible a steady-state behavior of the system. Equilibria are generally classified as static or dynamic. Here we assume the input u is fixed at $u=0$.

A *static equilibrium* is commonly referred to in the classical control theory as an equilibrium point. An equilibrium point is a state $\hat{x}$ that if once the system state $x(t)$ is equal to $\hat{x}$, it remains equal to $\hat{x}$ for all future time and a fixed input $\hat{u}$. Mathematically, it means that

$$h(\hat{x}) = 0 \qquad (3.3)$$

and

$$\dot{x}(t) = f(\hat{x},\hat{u}) = 0 \qquad (3.4),$$

in which equation (3.3) guarantees no state-jumps and (3.4) guarantees that the system state can not leave $\hat{x}$ under the control of the differential equation.

Static equilibria do not cover all the interesting steady state behaviors of a system. A common feature of nonlinear systems with state jumps is the occurrence of a special type of trajectory that takes the form of a closed curve. This is known as a *limit cycle* and represents a periodic solution of the system equations since, when the system state returns to its initial value, it must necessarily repeat its previous motion and so continue indefinitely. We call a limit cycle a *dynamic equilibrium state* of a system. It represents an oscillation that is intrinsic to the system and is not caused by external input variations.

3.1.2.2 Stability

The existence of an equilibrium is a necessary condition for system stability but not sufficient. Stability requires a *stable equilibrium*, in which a small perturbation won't cause the system to leave the neighborhood of the equilibrium state or trajectory. Roughly, an equilibrium point is asymptotically stable in a region around the equilibrium if whenever the system starts from any place within the region, it ends up returning to the equilibrium. It is unstable if it moves away when starting at some position in the region. A limit cycle is asymptotically stable in a nearby region if all trajectories starting in that region approach it asymptotically and is unstable if some trajectory starting in the region moves away. An asymptotically stable limit cycle is also called a periodic *attractor.*

Above is a geometric description of the stability of a system. Mathematically, system stability is defined in the following way:

An equilibrium state $\hat{x}$ (or dynamic limit cycle) is said to be stable, if, for any arbitrarily small number $\varepsilon>=0$, there exists real numbers $\delta > 0$, and $T>0$, such that, if $||x(0)-\hat{x}|| <\delta$, then $||x(t)-\hat{x}||< \varepsilon$ for all $t>T$. Otherwise, the equilibrium state (or the limit cycle) is unstable. It is asymptotically stable if it is stable and, in addition, $||x(t)-\hat{x}||\rightarrow 0$ as $t\rightarrow$ inf.

Here $||x(t)-\hat{x}||$ is defined as the distance between the state x(t) and the equilibrium state $\hat{x}$ when $\hat{x}$ is an equilibrium point. If $\hat{x}$ is a limit cycle, the distance is the shortest distance from x(t) to any points on the limit cycle.

3.2 A State-Space Model for a Bandwidth Sharing System

After briefly describing the state-space modeling technique, now we are ready to present a state-space model for a bandwidth sharing system.

3.2.1 Target System

We are generally interested in the rate behaviors of flows in the Internet. However, the Internet is too big and has too many factors to be characterized by a single state-space model. To simplify the study, we choose a target system that is composed of a fixed number of flows that all use AIMD-based congestion control algorithms. To focus on the steady state congestion control algorithm, we ignore all timeouts and other components in a congestion control protocol, and we make no distinctions between retransmitted data and new data, and abstract all of them just as a data rate. In addition, we assume a single bottleneck link for all competing flows in the system. Figure 3.1 illustrates the target system of our study, in which N congestion controlled flows are competing for the same bottleneck link L, which has a fixed rate R and a limited queueing capacity B.

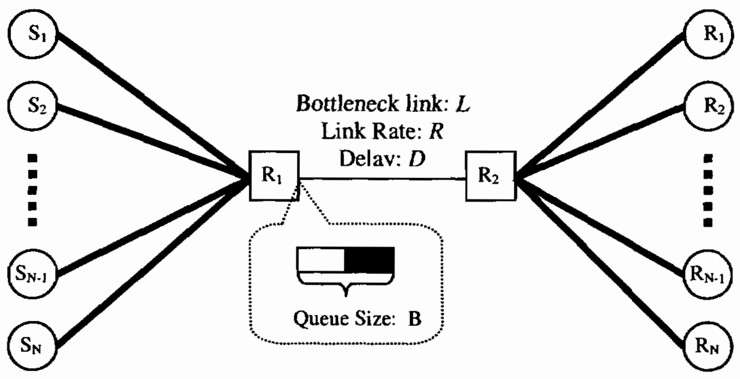


Figure 3.1: A Bandwidth Sharing System

Models for a network system can generally be divided into packet-based and fluid-based ones. A packet-based model uses packets as the basic units and the system's behavior is related to the detailed character of each packet, such as its size and timing information. A fluid-based model abstracts a system's behavior by rates, and thus ignores the information of various packet sizes and the packet interval times. In general, packet-based models are close to reality but are harder to use for any theoretical studies than fluid-based models. We choose to use a fluid-based approach because it requires fewer state variables. In later chapters, we address the quantization effect and randomness caused by packet sizes.

To construct a model for the target system, we first divide it into a bottleneck subsystem and N AIMD rate control subsystems. We then locate the key factors in each kind of subsystem to determine the system state.

3.2.1.1 Bottleneck Subsystem

We abstract a bottleneck link as a leaky bucket that has a constant leak rate (the bottleneck rate) R and a bucket (the bottleneck queue) with a limited size B. The bucket fill-level $fl(t)$ is controlled by the leak rate and the input rate $r_S(t)$ to the bucket, which is the sum of input rates of all the flows to the queue. The equations for the evolution of $fl(t)$ are summarized in (3.5).

$$\begin{cases} \dfrac{dfl(t)}{dt} = \min(r_S(t) - R, 0) & \text{if } fl(t) = B \\ \dfrac{dfl(t)}{dt} = r_S(t) - R & \text{if } 0 < fl(t) < B \\ \dfrac{dfl(t)}{dt} = \max(r_S(t) - R, 0) & \text{if } fl(t) = 0 \end{cases} \tag{3.5}$$

A congestion signal is produced when $fl(t) = B$ and $r_s(t)>R$. However, it is not necessary that every flow perceives the congestion signal. Determining which flow should perceive the congestion signal is not simple. In reality, some flows might be "unfortunate" and get their packet dropped when the bottleneck queue is full, while some flows do not. This random phenomenon happens in the taildrop queue, and could be exaggerated by some advanced queue management schemes [BCC+98] such as RED [FJ93]. We address this issue when we define the event trigger function $h(x)$.

3.2.1.2 AIMD Subsystem

We view the AIMD rate controller of each flow in the target system as a feedback subsystem, which outputs a signal onto the network to probe the bottleneck state and uses the probe result to control the data output rate. This feedback subsystem is illustrated in

Figure 3.2. An AIMD rate controller probes the network's state with the data it sends. Data packets travel from the sender to the receiver, and acknowledgments for each packet travel back from the receiver to the sender. The time from sending a packet to receiving its acknowledgment is the round-trip time (RTT). The RTT is important because it is the delay around the feedback loop.

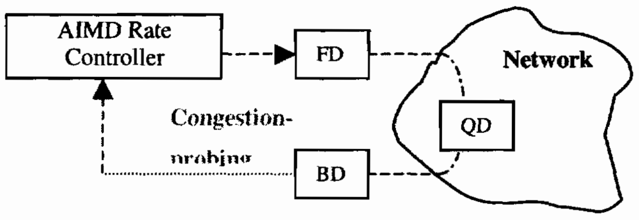


Figure 3.2: AIMD Rate Control Subsystem

We divide the RTT into three parts: the *forward delay FD*, the *bottleneck queueing delay QD*, and the *backward delay BD* (that is: RTT = FD + QD + BD). The forward delay is the time between the instant that a flow increases its rate to the instant that the increment starts contributing to the input rate to the bottleneck queue. The bottleneck queueing delay is the time that is taken by a packet to go through the bottleneck queue. The backward delay is the time between the instant that a congestion signal is generated at the bottleneck to the instant that the flow receives it. Since we assume a single bottleneck link, packets should only accumulate at the bottleneck and not at other links. Thus, both the forward delay and backward delay are composed by link propagation delays, and are therefore treated as constants in our study. Since the queueing delay is related to the amount of data in the queue, we simply let QD = fl(t)/R.

Now we start to abstract the behavior of an AIMD rate controller. An AIMD rate controller limits the rate at which data is sent out on the network by using a congestion window. The congestion window size defines the maximum amount of outstanding data, data that has been sent but not yet acknowledged; hence, the amount that is sent out in one RTT. If no congestion signals have arrived at the AIMD controller, it should have received acknowledgments for all packets that were sent during the last RTT. An AIMD rate controller uses packet losses or explicit notifications as congestion signals. For details of packet loss detections, please refer to Chapter 2. Under the absence of congestion signals, an AIMD controller increases its congestion window size by α packets in every RTT (that is increasing its rate r_i by α packets/RTT in every RTT); otherwise it decreases its congestion window by β times the current window size. Equation (3.6) and (3.7) summarize an AIMD controller's behaviors.

When no congestion occurs (uncongested state):

$$\frac{dr_i(t)}{dt} = \frac{\alpha \times MSS}{RTT_i^{\,2}} \qquad (3.6),$$

in which MSS is the packet size.

When congestion occurs:

$$r_i \leftarrow (1-\beta) \times r_i \qquad (3.7).$$

With the above abstractions of the bottleneck link and AIMD controllers, the target system can be represented as a decentralized control system, whose behavior is controlled by N AIMD controllers and one bottleneck queue. The link between these distributed controllers and the key variables of the system is presented in Figure 3.3. Our studies of bandwidth sharing systems are based on this abstracted system.

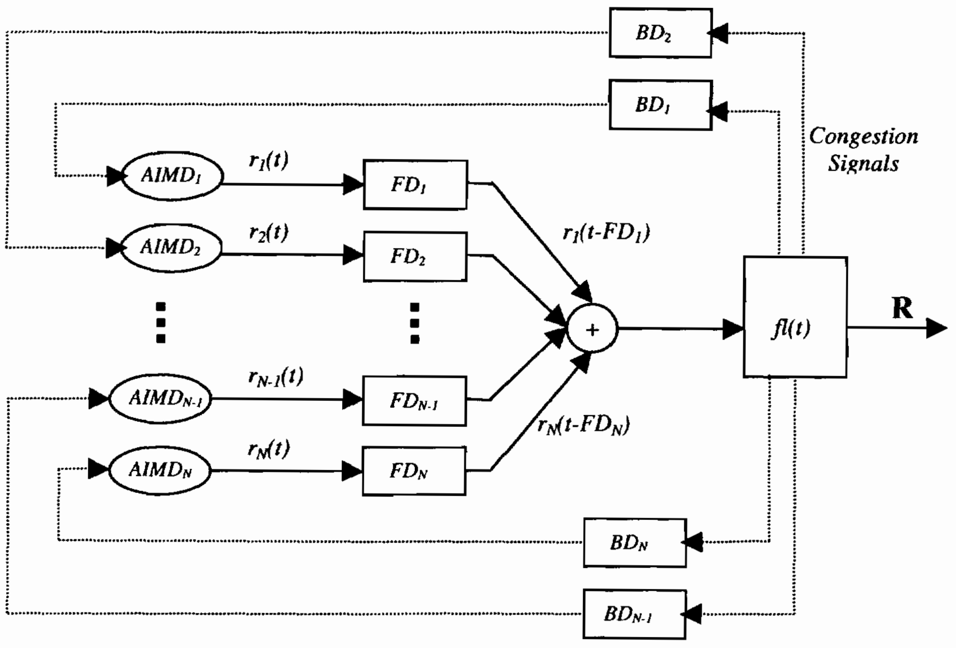


Figure 3.3: The Bandwidth Sharing System with Key Information only

3.2.2 System State

The essential feature of the state of a state-space model for a system is that it contains all information about the past history of the system that is relevant to its future behavior. In the target system, the important information that captures the state of the system is the instantaneous transmission rate of each flow and the queue fill-level in the bottleneck router at any instant. Each flow adjusts its rate based on its current rate and the congestion signals from the bottleneck link, which are determined by the aggregated rate of all competing flows. Thus, the transient transmission rates of all competing flows and the queue fill-level of the bottleneck router hold all the history information that is relevant to determine the future behavior of the system. Therefore, we choose the transmission rate of every competing flow and the fill-level of the bottleneck router queue as the state variables. We use $r_i(t)$ to denote the transmission rate of flow i, and $fl(t)$ to denote the bottleneck queue fill-level at time t. For a system with N competing flows, the system's state at time t in our model is a vector $x(t) = [r_1(t), r_2(t), \cdots, r_N(t), fl(t)]^T$.

The target system has a few other key factors that affect the system behaviors but are not counted as state variables because they are static or being assumed static by us. These factors are the bottleneck rate R, the queue size B at the bottleneck router, and the forward and backward delay between each sender and the bottleneck queue. In a real system, these factors might be varied by router software dong custom queuing or differential services.

3.2.3 Differential Equations and State-Jumps

Based on the above system state definition, we now present the rules that control the state evolution. We first look at the event trigger function $h(x)$, and then discuss the functions $g(x)$ and $f(x, u)$ for the system evolution with and without state-jumps.

The event trigger function *h(x)* is not straightforward. As mentioned in subsection 3.2.1.1, one issue that needs to be addressed by the trigger function is which flow perceives the congestion signal. We have a few choices on the selection of the trigger function. One choice is a pure deterministic approach in which every flow perceives all congestion signals. This choice maps to the reality when severe congestions happen. An alternative choice is using a random process that links the possibility of perceiving a congestion signal to the transient rate of the flow. The goal of building the model is to study the dynamic rate behaviors. This goal includes two tasks: understanding the system stability and understanding the interaction between controller behavior and its congestion perceptions. For the task of studying system stability, the first choice (every flow perceiving a congestion signal) provides an easy start. For the task of studying the interaction between rate and congestion perceptions, we can no longer assume that all flows perceive all congestion signals. In later chapters, we will show through experiments that flows with different parameters in their rate controller have different congestion perceptions. In this chapter, since the major theme is to use the state-space model to study the system stability, we decide to start with the simple assumption that every flow perceives a congestion signal whenever congestion happens. We will extend our model to more general cases and investigate the effects of these choices in the next two chapters using simulations and real-world experiments.

The event trigger function *h(x)* also needs to address the effect of feedback delays on AIMD controllers. An AIMD controller will not back off continuously during queueing overflows at the bottleneck. Because RTT is an AIMD controller's feedback delay, an AIMD controller lets back offs happen no more than once in a single RTT period[6]. In addition, different AIMD controllers could perceive congestion signals at different times when they have unequal backward delays.

For this delay issue, and since the backward delay could differ among flows, we define a separate event trigger function $h_i(x)$ for each flow *i* as:

$$\begin{cases} h_i(x) = 1 & \text{if } fl(t-BD_i)=B,\ r_S(t-BD) > R\text{, and } h_i(\sigma) \text{ is 0 for all } \sigma \text{ such that } t\text{-}RTT<\sigma<t \\ h_i(x) = 0 & \text{otherwise.} \end{cases} \quad (3.8)$$

This definition makes sure that, once the fill-level reaches its limit *B* and the total input rate from all flows $r_s(t)$ is higher than the link rate, the flow *i* receives a congestion signal after a delay BD_i (indicated by the condition $fl(t\text{-}BD_i)=B$ and $r_S(t-BD) > R$). The definition in (3.8) also makes sure that the flow receives at most one congestion signal per RTT, otherwise $h_i(\sigma)$ would not be zero for all σ such that t-RTT<σ < t.

Up to now, we know the trigger functions for all the state jumps. To define the functions that control the system behavior other than state-jumps, we want a function that indicates the absence of state-jumps. Therefore, we define a global event trigger function *h(x)*.

$$\begin{cases} h(x) = 0 & \text{if all } h_i(x) = 0 \text{ (no state-jumps)} \\ h(x) = 1 & \text{otherwise} \end{cases} \quad (3.9)$$

When the event trigger function *h(x) = 0*, the system state evolution is under the control of differential equations, in which the evolution function *f(x,u)* is presented as the following:

$$\begin{cases} f(x(t),u(t)) = [\frac{\alpha_1 \times MSS_1}{RTT_1^2(t)}, \frac{\alpha_2 \times MSS_2}{RTT_2^2(t)}, \cdots, \frac{\alpha_N \times MSS_N}{RTT_N^2(t)}, \min(r_S(t)-R,0)]^T & if \quad fl(t)=B \\ f(x(t),u(t)) = [\frac{\alpha_1 \times MSS_1}{RTT_1^2(t)}, \frac{\alpha_2 \times MSS_2}{RTT_2^2(t)}, \cdots, \frac{\alpha_N \times MSS_N}{RTT_N^2(t)}, r_S(t)-R]^T & if \quad 0<fl(t)<B \\ f(x(t),u(t)) = [\frac{\alpha_1 \times MSS_1}{RTT_1^2(t)}, \frac{\alpha_2 \times MSS_2}{RTT_2^2(t)}, \cdots, \frac{\alpha_N \times MSS_N}{RTT_N^2(t)}, \max(r_S(t)-R,0)]^T & if \quad fl(t)=0 \end{cases} \quad (3.10)$$

In *f(x,u)*, flow *i* increases its rate by $\frac{\alpha_i \times MSS_i}{RTT_i^2(t)}$, in which α_i is the AIMD increment parameter of flow *i*, and $RTT_i(t)$ is its round-trip-time. Also in (3.10),

[6] This claim is right for TCP NewReno, TCP SACK, but it is not right for TCP Tahoe and TCP Reno. We make this claim because TCP NewReno and TCP SACK are the dominant flavors of TCP in the Internet today [PF01].

$r_S(t) = \sum_{i=1}^{N} r_i(t - FD_i)$, which is the total rate of all competing flows at the bottleneck, where FD_i is the forward delay of flow i. Notice that (3.10) is divided into three cases based on the value of $fl(t)$. When $fl(t)$ is larger than zero but less than B, then the derivative of fill-level can be either positive, zero, or negative. When it is equal to zero, since the fill-level cannot become negative, the derivative of $fl(t)$ can only be positive or zero.

When $h(x)$ is not zero, that means at least one state-jump happens. For every $h_i(x) = 1$, we have a state transition under the control of $g_i(x)$:

$$g_i([r_1(t), r_2(t), \cdots, r_N(t), fl(t)]^T) = [r_1(t), \cdots, (1-\beta_i) r_i(t), \cdots, r_N(t), fl(t)]^T \quad (3.11).$$

In (3.11), only flow i's rate is reduced by a factor of β_i and all other state variables of the system are unchanged. This is the AIMD's rate behavior in (3.7).

The state definition in subsection 3.2.2 and the equations (and transitions) (3.8) ~ (3.11) together comprise the state-space model for our target system.

3.2.4 Simulation

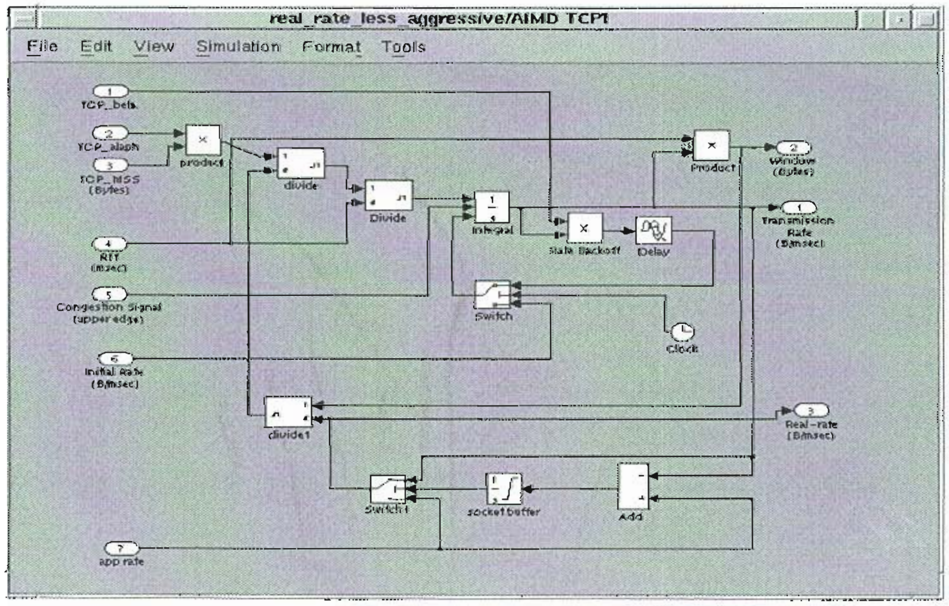


Figure 3.4 A Snapshot of a Simulink Simulation Block

Based on the state-space model presented above, we build a simulation for the target system using Matlab's Simulink toolkit [MATLAB]. The Simulink toolkit supplies building blocks to represent the differential equations, event triggers, and state jumps, which enable us to build a simulation according to our model. Actually the development of our model is motivated as an example of using the Simulink toolkit to study computer applications. As an example, we show a captured figure of the simulation block of an AIMD rate controller in Figure 3.4. The full Simulink simulation based on (3.8) ~ (3.11) is available on [SIMU].

3.2.5 System Dynamics

The goal of building the state-space model is to understand the dynamic rate behavior of competing flows in the network. The power of a state-space model is that it can help one visualize the dynamic behavior in its state space. In this subsection, we present some examples to demonstrate the geometrical representations.

To make a step-by-step demonstration, we first apply the model to a system with a single flow, and then show the cases with two flows and N flows.

In all the following examples in this section, we assume an equal backward delay for all competing flows. We also assume that every flow perceives the congestion signal when a congestion happens. Thus, all flows get synchronized congestion signals. We also assume identical AIMD controllers and the same forward delay for every flow in the system. We make such assumptions just to demonstrate the system behavior in simple environments. The reality has many more random factors that could cause more complex behavior than that presented in this chapter. We do not argue that the systems we show in these examples are realistic. On the contrary, we will discuss how realistic this assumption is at the end of this chapter, and we will extend our study to more complex cases in the following chapters.

3.2.5.1 A System with a Single Flow

For a system with only one flow, its system state is represented by a two-dimensional (2D) vector *x(t)* =[*r(t)*, *fl(t)*]^T. The state-space of the system is a two-dimensional space, and the system trajectory is presented in Figure 3.5, which is produced from a Simulink simulation with a 50KB/S bottleneck link rate and 1500B queue limit.

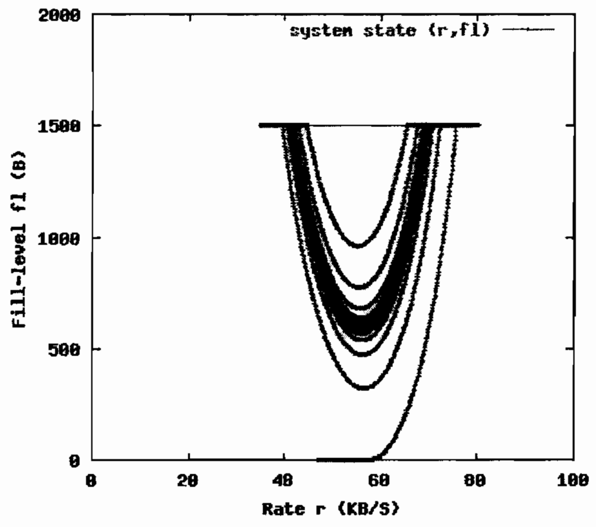


Figure 3.5 (a): Trajectory of a System with a Single Flow

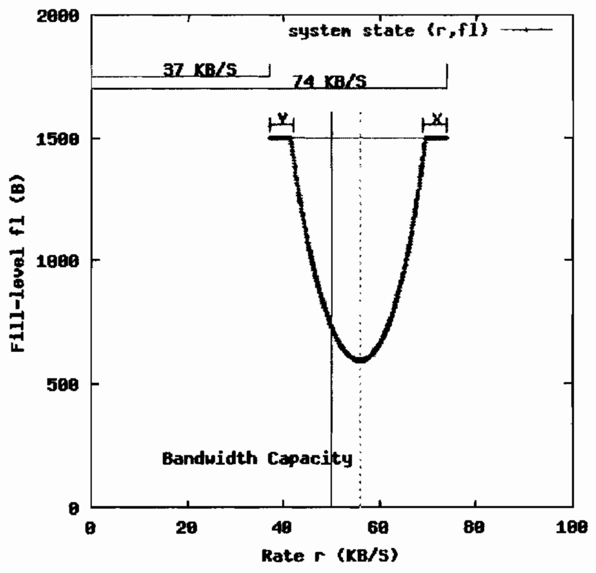


Figure 3.5 (b): The Limit Cycle of a System with a Single Flow

We show two figures of the system state. The first one, Figure 3.5(a), shows the system trajectory starting from the initial point (50KB/S initial rate and 0 fill-level) and ending up following a limit cycle. The second one, Figure 3.4(b), shows just the limit cycle, which is the system state during its stable state.

The state space figure in Figure 3.5(b) illustrates the significance of the state-jumps, which is the thin part of the limit cycle. Upon a state jump (congestion signal arriving at the AIMD controller), the flow's rate jumps from 74KB/S to 37 KB/S because this flow uses AIMD (1,1/2) as its controller. Compared to the thin line between the states before and after a state jump, the system states under the control of the differential equation tend to migrate gradually and are thus shown on the figure as the thick part of the limit cycle.

Compared to a normal time-sequence saw tooth figure, the state-space trajectory illustrates the connection between the internal states of a system. For example, the system trajectory shows a few interesting aspects of the system. First, the effect of the feedback delay is clear in Figure 3.4(b). After the queue fill-level reaches its limit B, the AIMD controller still increases its rate until the congestion signal arrives at the AIMD controller. This behavior is indicated by the state evolution marked X in the figure, and this period is BD, which is the backward delay[7]. Similarly, after the AIMD controller receives a signal, it reduces its rate and this rate reduction will only arrive at the bottleneck after a delay FD. The state evolution during this period is the marked Y in the figure.

The system experiences a period ($BD+FD$) of overflow before the queue fill-level goes down. In reality, this behavior can be alleviated by using the rule of packet conservation. The rule of packet conservation says that the sender should not send a new packet unless it knows a packet has successively left the network. In TCP congestion control, the packet conservation is implemented by using the arrival of acknowledgements as triggers for sending new packets. When a sender sends faster than the network capacity, it will not receive acknowledgments in a rate that can match its sending rate (because packets are delayed or dropped in the network). By applying the packet conservation rule, the

[7] Notice here that the distance on the state-space plot doesn't indicate time directly. The trajectory is plotted by sampling the state every constant period. The number of sampling points on the state trajectory actually indicates the time period.

sender would not keep increasing its rate once its acknowledgment rate cannot keep up. Thus the sender can avoid increasing its rate when its rate is already higher than the network capacity. Currently, our state-space model does not model the packet conservation rule for two reasons. First it is a rule that is independent of AIMD congestion control and some newly developed congestion control protocols do not use it [BBSF01]. Second, a long feedback delay could still cause a rate increase and congestion over a long period. To model the packet conservation rule is one of our future plans to extend the state-space model. The effect of feedback delay creating a long congestion period motivates the deployment of active queue management approaches, such as RED, at the bottleneck queue. An early congestion signal can effectively reduce this overflow period. It may also reduce the average queuing delay, which shortens the feedback delay.

In Figure 3.5(b), we plotted one more line, the link capacity line, in addition to the system state trajectory. Ideally, when the flow's rate is lower than the capacity, the bottleneck queue fill-level decreases. Once the flow's rate passes the capacity, the bottleneck queue fill-level increases. However, due to the same effect of the forward delay shown above, the changing of fill-level is actually delayed. We plotted a dotted line to indicate the system state at the time the fill-level changes from draining to filling. The distance between the bandwidth capacity line and the dotted line is caused by the forward delay. In a system with zero forward delay, the two lines overlap.

3.2.5.2 A System with Two Flows

For a system with two flows, the system state is represented by a vector $x(t) = [r_1(t), r_2(t), fl(t)]^T$. The system state is again plotted in the system's state-space.

One thing we can see from the 3D system trajectory figure, Figure 3.6(a), is that the system state converges to a limit cycle. The proof of this dynamic stability is presented at the end of this section.

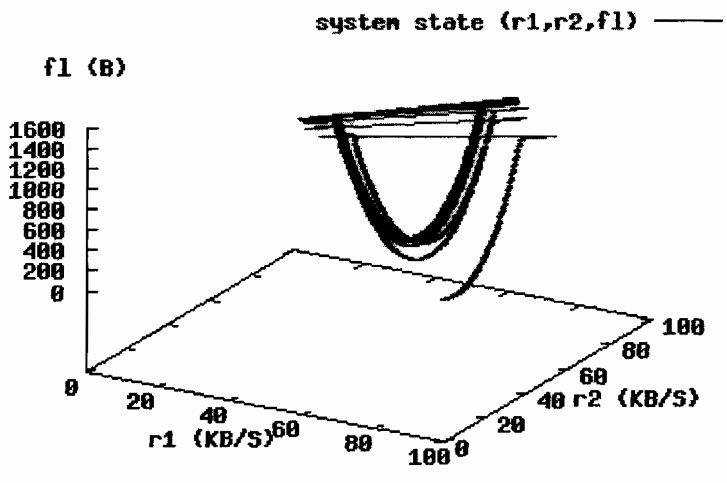


Figure 3.6 (a): Trajectory of a System with Two Flows

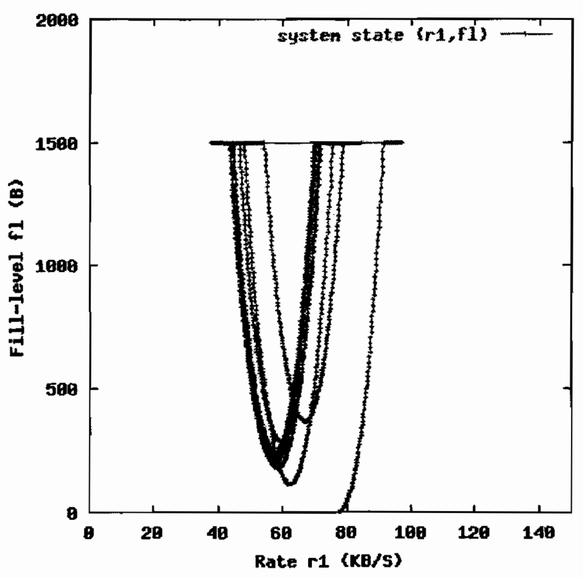


Figure 3.6 (b): Projection 1 of the System Trajectory

More can be observed from this state-space plot. Since this system has two flows, the fairness of bandwidth sharing between the two flows is one aspect we are interested in. We make projections of the 3-D plots onto 2-D figures. If the 2 dimensions we choose to project to are the queue fill-level and a flow's rate, the projected figure, Figure 3.6(b), is similar to the single flow case. However, if we choose the 2 dimensions as the rates of both flows, we get a figure, Figure 3.6(c), of the relative progress of the two rates. We add a 45-degree line, called the *fairness line*, in this projection. Points on the fairness line indicate states in which the two flows have the same transient transmission rate. We can see that the two flows achieve a fair sharing of bandwidth in the system's stable state. The use of 2-D system state projections to show the fairness has became a popular way to show the TCP-friendliness aspect in some recent papers [BBSF01, YL00].

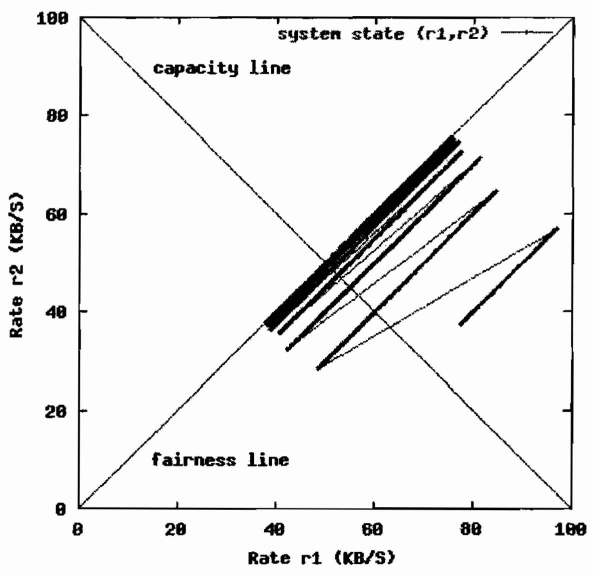


Figure 3.6 (c): Projection 2 of the System Trajectory

3.2.5.3 A System with *N* flows

For a system with *N* flows, its system state is represented by $x(t) = [r_1(t), r_2(t), \ldots, r_N(t), fl(t)]^T$. To present the system state in its state space, we need to overcome a representation problem: it is not easy to represent on paper clearly a plot with more than three dimensions. One solution to this problem is to use special graphics tools [WLG97] to draw high dimension trajectories on paper. Another one is to aggregate multiple dimensions to one as long as we are not interested in the individual state variables that are being aggregated.

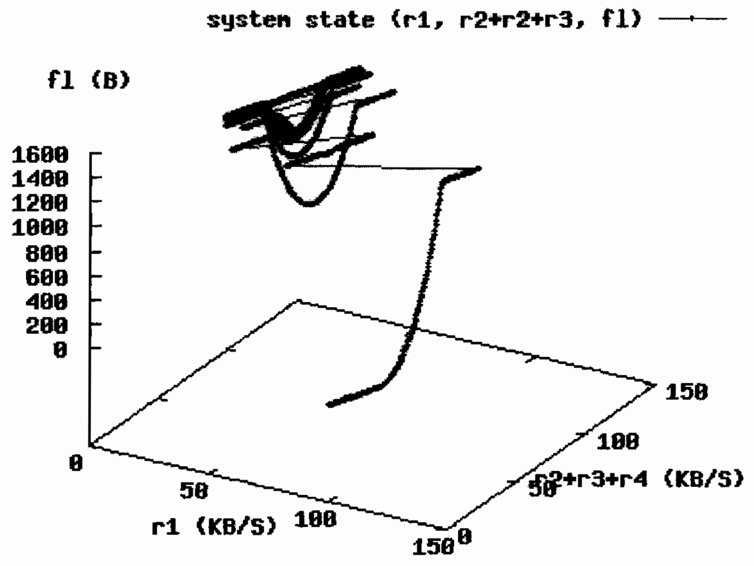


Figure 3.7 (a): Trajectory of a System with Four Flows

We design an aggregation technique to present the system state plot based on the assumption of synchronized congestion signals. If we focus on only one flow, we can merge the system state of the other flows as one, and we show that the aggregated behavior of a group of AIMD flows can be represented as an AIMD flow with different parameters. When we only care about the aggregated behavior as a whole rather than the behavior of each individual flow, we use the following way to aggregate competing traffic:

Under the synchronized back off assumption, N AIMD(α , β) flows can be viewed as one AIMD(N α , β) flow in terms of aggregated dynamics.*

The intuition of this aggregation is that *N* AIMD flows would get *N* times the bandwidth of a single AIMD flow. Here we show an example of a system with 4 flows. We aggregate flows 2 to 4 to one flow using the above method, and plot the system state in Figure 3.7 (a)~(c). Figure 3.7 (a) and (b) show that the state trajectory converges to a limit cycle as shown in earlier examples.

In Figure 3.7(c), the initial state is changed to the point at which the aggregated competing traffic holds zero bandwidth and the target flow has all the available bandwidth. The trajectory shows that the system state later stabilizes on a limit cycle that has an average sharing ratio 3:1, which is consistent with intuition. This result also indicates that an AIMD($N*\alpha$, β) flow will get N times the bandwidth of an AIMD(α, β) flow under the assumption of synchronized congestion signals.

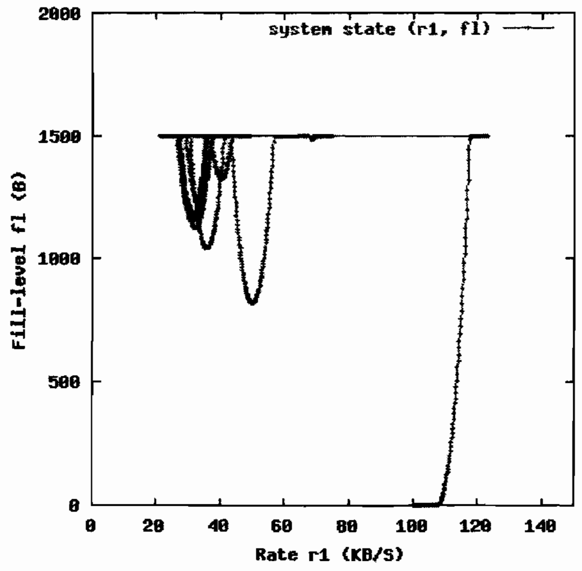


Figure 3.7 (b): Projection 1 of the System Trajectory

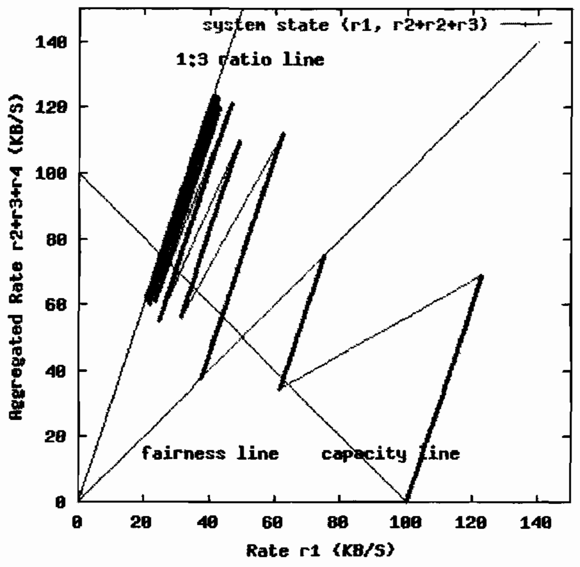


Figure 3.7 (c): Projection 2 of the System Trajectory

3.2.5.4 Stable Limit Cycle

Early examples have shown geometrically that the system state converges to a stable limit cycle. In this section, we present a theoretical proof of this dynamic stability. We state the goal of the proof in *Theorem-1* first, followed by a brief proof.

Theorem-1:

When multiple AIMD flows compete for a constant available bandwidth R, the system state, under the assumption of synchronized back off (with a common backward delay BD and a common forward delay FD), and packet conservation, converges to a limit cycle that passes through the point $P=[r_1,r_2,\cdots,r_N,B]^T$, *in which*

$$r_i=\frac{2\alpha_i(1-\beta_i)}{\beta_i}\times\frac{MSS_i}{RTT_i^2}\times\frac{R'}{\sum_{j=1}^{N}\frac{\alpha_j(2-\beta_j)MSS_j}{\beta_j RTT_j^2}} \qquad (3.12),$$

where $R'=R+\frac{BD}{2}\times\sum_{j=1}^{N}\frac{\alpha_j MSS_j}{RTT_j^2}+\frac{FD}{2}\times\sum_{j=1}^{N}\frac{\alpha_j MSS_j}{RTT_j^2}$,

and the AIMD flows' parameters satisfy the following constraint:

$$\sum_{i=1}^{N}\left(\frac{1-\beta_i}{\beta_i}\times\frac{\alpha_i MSS_i}{RTT_i^2}\right)<\frac{R}{(BD+FD)} \qquad (3.13)$$

When all the flows have the same AIMD parameters, MSS, and RTT, they get equal bandwidth share.

We argue in Theorem-1 that the given limit cycle is stable. This proof is based on the assumption that $\alpha>0$ and $0<\beta<1$. The proof of Theorem-1 is in 3 steps. First, we prove that system trajectories from any point would have a series of cross points with the plane *fl=B* in the state-space. Second, we prove the trajectory starting from the point stated in Theorem-1 comes back to the same point, and thus it is a limit cycle. Third, we prove the crossing points of a trajectory with the plane get closer and closer to the point *P* in the Theorem-1, hence showing that is stable. We briefly present the idea of each step as follows, and a more detailed proof is in Appendix A1.

Step-I

In this step, we prove that, for any starting state $X = [r_1, r_2, \cdots, r_N, fl]^T$, the system trajectory starting from X intersects plane *fl=B* again and again.

If $\sum_{i=1}^{N} r_i < R$, according to (3.8), we know that no state-jumps happen no matter what value *fl* has, and the system state is under the control of (3.10). Notice that for every flow *i*, $\frac{\alpha_i \times MSS_i}{RTT_i^2(t)}$>0, which indicates that r_i keeps increasing. In addition, the increment $\frac{\alpha_i \times MSS_i}{RTT_i^2(t)}$ is always larger than a positive constant $\frac{\alpha_{\min} \times MSS_{\min}}{RTT_{\max}^2(t)}$, in which $\alpha_{\min}$=min[α_1, α_2, ..., α_N] and RTT_{max}=max[FD_1, FD_2, ..., FD_N]+BD+B/R. Thus we know that $\sum_{i=1}^{N} r_i$ will continue to increase and eventually result in $\sum_{i=1}^{N} r_i \geq R$. Without state-jumps, $\sum_{i=1}^{N} r_i$ keeps increasing and thus *fl* increases. Eventually, the system arrives at the state $X' = [r_1', r_2', \cdots, r_N', B]^T$, which is a state on plane *fl=B*. Since $\sum_{i=1}^{N} r_i' \geq R$ and *fl=B,* state-jumps will happen until causing the system to leave the plane *fl=B* and drop below it when 0<$\beta < 1$. If the system leaves the plane *fl=B*, the above process will be repeated again in future.

To continue the proof, we divide the system migration along a trajectory to multiple rounds. The system state from any point would go across the plane *fl=B* followed by a state-jump. We choose the state-jump as the end of each round. Thus, each round starts with a state just after a backing off upon a congestion signal, and ends with a state jump on the next congestion signal.

Step-II

This step verifies that the trajectory starting from the point *P* in Theorem-1 is a limit cycle. The way we prove it is to prove that the trajectory that starts from *P* comes back to *P*. The detailed derivation is presented in Appendix A.1.

Step-III

As defined in Step-I, each round starts with a state just after a backing off upon a congestion signal, and ends with a state jump on the next congestion signal. We compare the distance of the system state at the beginning of every round to the point P in Theorem-1. If the distance is converging to zero, it indicates that the system state vector is approaching the limit cycle. Furthermore, we know if the system state ever reaches the above limit cycle, it will stay on it, unless there is noise to cause the system state to leave the limit cycle again.

We assume that the system states of the flows start from a random initial condition $P' = [r_1 + \Delta'_1, r_2 + \Delta'_2, \cdots, r_N + \Delta'_N, B]^T$, and the system states arrive at $P'' = [r_1 + \Delta''_1, r_2 + \Delta''_2, \cdots, r_N + \Delta''_N, B]^T$ after one more round. The position of the system is represented in a way to emphasize the distance from the one state to the point $P = [r_1, r_2, \cdots, r_N, B]^T$. If we can prove that the distance Δ'_i in each dimension becomes smaller and smaller, then we know that the system state gets closer to *P*.

To prove this, we first derive the following relationship between Δ'_i and Δ''_i:

$$\Delta''_i = (1 - \beta_i)[\Delta'_i - \frac{\alpha_i MSS_i}{RTT_i^2} \times \frac{2}{\sum_{j=1}^{N} \frac{\alpha_j MSS_j}{RTT_j^2}} \sum_{j=1}^{N} \Delta'_j] \tag{3.14}$$

In the Appendix A1, we prove that $\frac{\sum_{j=1}^{N}(\Delta''_j)^2}{\sum_{j=1}^{N}(\Delta'_j)^2} < 1$ when $0 < \beta < 1$, which indicates that the distance to the limit cycle becomes smaller as time goes by. Therefore, as time goes to infinity, the distance to the limit cycle approaches zero. Thus, the limit cycle is stable.

□

We have presented elsewhere that the trajectory of a system composed by a single AIMD flow is a limit cycle [SLW+00]. Here by proving that the system state converges to a limit cycle, we prove that a system with multiple AIMD flows converges to a dynamic stability. However, this proof depends on the assumptions of all flows receiving synchronized congestion signals.

3.3 Discussion

In this chapter, we described a state-space model that combines differential equations and state jumps. We used this model as a tool to study the dynamic behavior of systems in a computer network. We made several assumptions in this model and here we present a brief discussion of them and some miscellaneous issues of the model.

3.3.1 Linear versus Nonlinear Systems

In this work, the state-space model is mainly for nonlinear systems that are linear most of the time but have additional state jumps driven by internal events.

Even with only state jumps, the behavior of these systems can be quite different from the behavior of linear systems. Modeling techniques for linear systems do not fit these systems because they lack the global property of linearity. The stability of a nonlinear system in the neighborhood of an equilibrium point does not necessarily imply any global properties[8]. There may indeed be many equilibriums, some stable and others not, in which case there will be only a limited region of convergence (domain of attraction) around any equilibrium point which is locally asymptotically stable. Furthermore, there can be other nonlinear behavior, such as the persistent oscillations known as limit cycles, which constitute a type of dynamic equilibrium, rather than the static equilibrium points that are dominant in linear systems. One advantage of the state-space model is its geometrical representation. It can help us visualize the internal relationship among state variables, which is sometimes important for understanding the dynamics of the system, especially for nonlinear systems.

3.3.2 Modeling Competing Traffic

This chapter discusses the state-space model in a closed system that has no competing traffic from outside and thus has a zero input.

In general, competing traffic can be divided into responsive and unresponsive flows. Responsive traffic adjusts its rate according to the congestion status of the bottleneck link, and thus should be counted as one of the flows in the system. For unresponsive traffic, the model can deal with competing traffic by modeling it as an input to the system. Only one modification to the model, the aggregate rate $r_s(t)$, is required in order to take such external competing traffic into account. With a non-zero input, the system state is then determined by the differential equation, state jumps, and the external input. In the following chapters, we show examples of extending the model with competing traffic.

3.3.3 Assumptions about Congestion Signals

[8] Here the global property of linearity refers to the fact that the additivity property and homogeneity are globally applied in a linear system. Mathematically, let y1 be the output of x1, and y2 be the output of x2. Then the additivity property requires that y1+y2 is the output of x1+x2, and homogeneity requires that c*y1 is the output of c*x1, where c is a constant.

Early in this chapter we mentioned the complexity of generating congestion signals in the state-space model. Because events are very important parts of the state-space model, and congestion signals are the only events in the system, we believe it is worth discussing and recapitulating various assumptions made about congestion signal events.

Traditionally congestion signals are classified as either synchronized or asynchronous, and universal or non-universal. For synchronized congestion signals, all flows receive the same congestion signal at the same time. For asynchronous congestion signals, all flows receive congestion signals at different times. For universal congestion signals, signal is delivered to all flows. For non-universal congestion signals, it is delivered to a specific subset of the flows. Furthermore, there may be variation in the signal propagation time. That is the delay from when the congestion event is generated to when a flow's controller receives the signal. The signal propagation is captured in the backward delay (BD) in the state-space model.

Synchronized congestion signals are simply universal congestion signals with a common backward delay for all flows. Either non-universal congestion signals or different backward delays could cause asynchronous congestion signals.

Research work on bandwidth sharing behaviors can be divided according to the assumptions made about the congestion signals. The simulation complexity, as well as how close the assumption is to reality, varies as the assumptions change.

The most widely used and also the simplest assumption for congestion signals is synchronized back offs with a zero backward delay for all competing flows. It is used in the early study by Chiu and Jain [CJ89] before the design of TCP, and by many recent works on TCP-friendly congestion control protocols [BB01, PKTK99, YL01]. Early in this chapter we used a similar assumption with a non-zero common backward delay for all flows.

In the following chapters, we study the bandwidth sharing behavior with asynchronous congestion signals. We first continue with the assumption of universal congestion signals, but permit the backward delay to be different across flows. With different backward delays, the congestion signals to all competing flows are no longer synchronized.

After a study of the impact of different backward delays, we extend the assumptions one more step, so that each time a congestion happens some of the flows perceive the congestion signal and some not. The congestion signal distribution is controlled by a random process, which determines which flows get the signal based on their transient rates at the bottleneck and the congestion period. This random congestion distribution is becoming dominant as more and more random drop queues are deployed in the Internet.

Chapter 4

State-space Modeling Results and Analysis

In Chapter 3, we described the state-space model and studied its stability under the assumption of universal congestion signals and common backward delays. In this chapter, we extend the model to more complex cases, such as situations with non-common backward delays, or with non-universal congestion signals.

Based on the results in this chapter, we argue the following points about the bandwidth competition. These points are mostly about the fairness aspect among AIMD-based flows:

i. Fairness is only important between flows whose lifetimes are long enough compared to the stable limit cycle period.

ii. Fairness during the stable state is partly determined by the AIMD parameters of all competing flows.

iii. AIMD flows that are TCP-friendly under the assumption of universal congestion signals do not necessarily share bandwidth equally when congestion signals are non-univeral, because fairness is related to the congestion signal distributions, which in turn are related to the AIMD parameters.

iv. The inherent oscillations during the stable state cause an unavoidable buffering delay for CBR applications.

4.1 Fairness

This section discusses the fairness implications (points i, ii, and iii) of the state-space model. The section proceeds as follows: We first review TCP-style fairness in the state-space model with various feedback delays, and we then extend the study to the fairness between flows with different AIMD parameters. Both of these studies assume universal congestion signals. We relax this assumption at the end of this section by introducing non-universally distributed congestion signals and repeat the study of the fairness among AIMD flows.

4.1.1 TCP-style Fairness

In the current Internet paradigm, TCP ensures that flows share bandwidth equally under universal congestion signals when they have the same RTT. However, they do not share equally when they have different RTTs. We refer to this sharing behavior as TCP-style fairness.

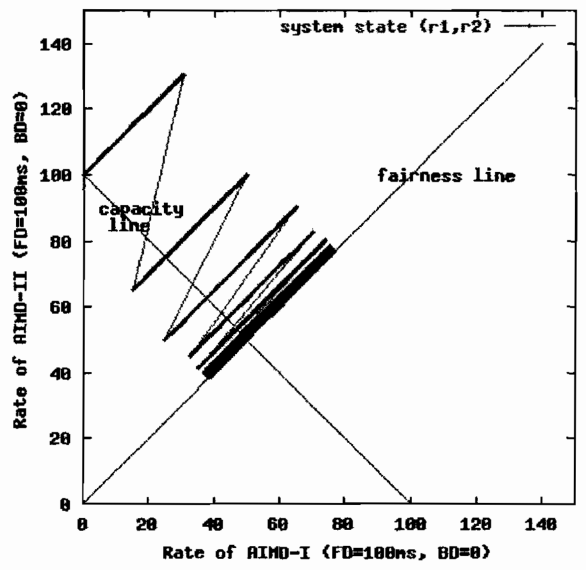


Figure 4.1: Fairness between two TCPs with the Same RTT.
(Universal Congestion Signals, Constant and Equal Signal Propagation Delay)

We plot the system state trajectory to illustrate TCP-style fairness in the state-space model. The system here has two competing flows and both of them use (1, ½) as their AIMD parameters. We first assume that every time the bottleneck queue overflows, each flow perceives a congestion signal after some constant backward delay. In addition, in all the following plot examples, we start with one flow (AIMD I) at a zero rate and the other flow (AIMD II) with the maximum bottleneck bandwidth. We plot the system state in the state space until it achieves a stable position. Since the fairness aspect is the focus here, we only plot a 2-D projection of the state-space trajectory with the rates of flows as the two dimensions.

Equal RTTs

We start with the case of two flows having the same forward and backward delays. The resulting trajectory in Figure 4.1 is essentially the same as the result presented in section 3.2.5.2: the system state converges to and oscillates on the fairness line.

Although the system state eventually enters a limit cycle, it is important to notice that the system state takes a while to converge to the fairness line. For a flow with a lifetime that is too short to reach the stable state, both its average and transient rates are not equal to the fair share (the link capacity divided by the total number of flows). Therefore, it does not make sense to discuss the fairness for short-lived flows, or to claim they share bandwidth unfairly with other flows. We argue that the fairness issue is only interesting for flows that live long enough to attain stability. Further discussion of the time-scales for fairness is presented in Section 4.1.2.

Unequal RTTs

The above result is only for a system with two identical flows. We now extend it to flows with different *RTTs*. When two flows in the target system have different *RTTs*, they could have different *forward delays (FD), backward delays (BD)* or both. Here we only look at the cases of (1) different *FD*s with the same *BD*, and (2) different *BD*s with the same *FD*.

Figure 4.2 and 4.3 show the trajectories of systems with universal congestion signals but different *FDs* and *BDs* respectively. Concretely, in Figure 4.2, AIMD-I has a 100ms delay, and AIMD-II has a 200ms *FD*. Both *BDs* of the two flows are set to 0. In Figure 4.3, AIMD-I has a 100ms *BD* and AIMD-II has a 200ms *BD*. Both *FDs* of the two flows are set to 0. In both systems, the two AIMD flows use the same AIMD parameters (1,1/2).

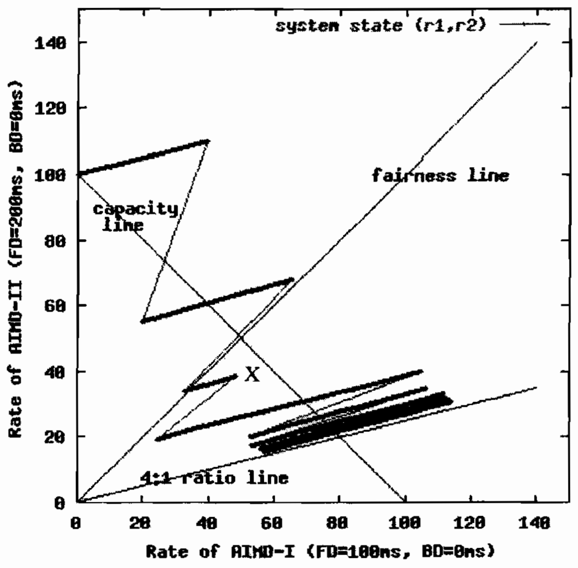


Figure 4.2: Fairness between two TCPs with equal BD but different FD.

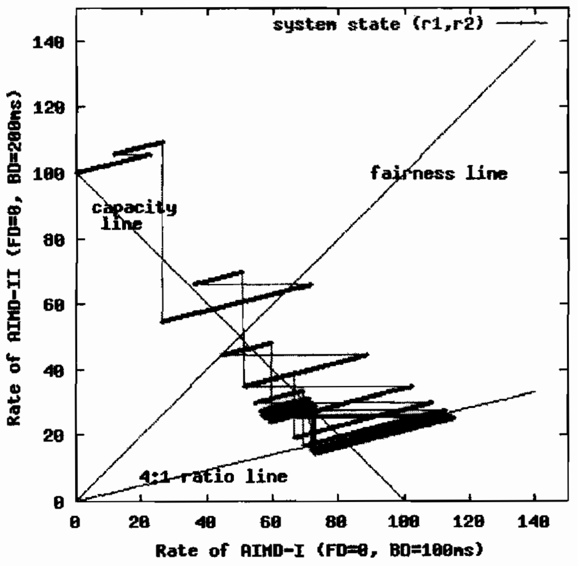


Figure 4.3: Fairness between two TCPs with equal FD but different BD

The trajectories in Figure 4.2[9] and 4.3 indicate a few interesting points. First, they show that the TCP-style fairness is actually unfair for AIMD flows with different RTTs. This

[9] Point X on Figure 4.2 might surprise someone because the rate reduction seems to happen before the total rate hits the capacity. In fact, the two state variables in all these figures are the transient rates at the senders, not the input rates to the bottleneck. The latter are the former with forward delays, and the delays here are different. Although the sum of the current rates at the senders is lower than the capacity, the sum of their rates at the bottleneck input is higher than the capacity and causes the congestion.

behavior is well-known in practice. In both 4.2 and 4.3, the sharing ratio between flows is inversely proportional to the square of RTT if congestion signals are universal[10]. Both Figure 4.2 and 4.3 show that the system state converges to the 4:1 ratio line.

Second, Figure 4.2 shows that, when the two flows have the same *BD*, they maintain the same bandwidth share ratio throughout the stable state, even though they have different RTTs due to the *FD* part. In contrast, Figure 4.3 shows that, when the two flows have different BD, they maintain the same bandwidth share only over a portion of the stable state. The ratio varies over a short time period during the stable state. Although both flows perceive the same congestion signal, the signal no longer arrives at the two AIMD controllers at the same time. Because of the asynchronous congestion signals, the resulting limit cycle on the 2-D projection is divided into parts, which are around the 4:1 ratio line rather than on it. Because the system state jumps around the 4:1 ratio line, the transient sharing ratio between the two flows actually varies during the stable state. This result also indicates the sensitivity of the bandwidth sharing ratio over different timescales.

We conclude from these trajectories that AIMD flows with different RTTs share bandwidth unfairly. A difference between backward delays causes more complex stability behavior, and actually affects the transient sharing ratio during the stable state. In Chapter 5, we verify these sharing results using packet-based simulations and real world experiments.

4.1.2 Fairness of Flows with different AIMD Parameters

In this section, we extend the study in 4.1.1 with changes of AIMD parameters. We study the bandwidth sharing between two competing AIMD flows (AIMD-I and AIMD-II) with equal forward and backward delays but different AIMD parameters. AIMD-II's

[10] Here, we have RTT=FD+QD+BD. Since QD is relatively small compared to the value we set to FD+BD, we use FD+BD as the RTT to calculate the ratio between flows. This approximation causes the system state to stabilize close to but not exactly on the 4:1 ratio line, as shown in Figure 4.2.

parameters are always (1,1/2). We choose four sets of parameters for AIMD-I: (1/2, 1/2), (1, 1/5), (1/3, 1/5), (2, 4/5). The first set is chosen so that flows have a common β parameter with TCP but a different α. The second set is chosen so that flows have a common α parameter with TCP but a different β. The last two sets have both α and β parameters different to TCP but based on the relationship (2.3) derived in [FHP00, LSW01, YL00]. Here, we repeat (2.3) as (4.1) for easy reference.

$$\alpha = \frac{3\beta}{2-\beta} \qquad (4.1)$$

We let AIMD-I start with zero rate and AIMD-II with all the bandwidth, and plot how the system state converges to stability. The trajectories of the four systems are presented in 2-D projections in Figure 4.4 (a), (b) and 4.5 (a), (b), respectively.

We can see from Figure 4.4 and 4.5[11] that AIMD parameters play an important role in determining the fairness at a system's stable state.

In Figure 4.4 (a), the flow AIMD-I using a smaller α (1/2 in this example) than the AIMD-II's (1 in this example) ends up using less bandwidth. Similarly, from Figure 4.4(b), we can see that a smaller β (1/5 in this example) causes a flow to get more bandwidth than the normal AIMD(1,1/2).

Ideally, maintaining a relationship between α and β is a way to make an AIMD flow share bandwidth equally with a TCP flow even though it has different parameters than TCP. The TCP-friendly equation (4.1) is designed with this thought.

[11] Because of the forward and backward delay, each flow tends to overshoot its rate. In sum, they have a higher transient rate than the bottleneck link capacity when the input rate at the bottleneck (the rate at sender after a forward delay) is equal to the capacity. Thus, the centers of the stable limit cycles in both figures are not on the capacity line but are actually higher than it. However, according to our measurements, both the centers are still on the fairness line, and thus the average share ratio between two flows over long time scales is equal.

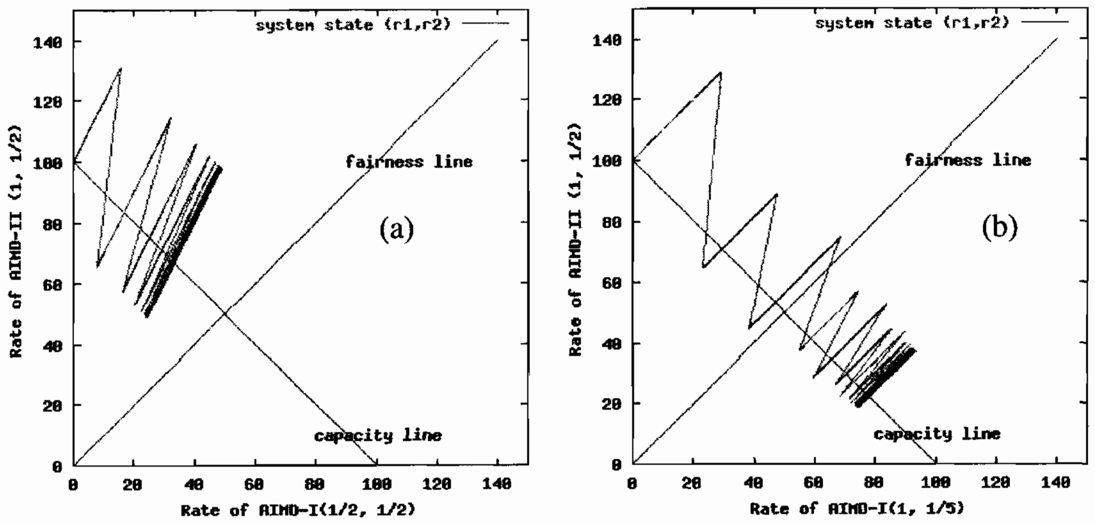


Figure 4.4: Fairness between unfriendly flows.
(a) AIMD(1/2,1/2) and AIMD(1, 1/2), (b) AIMD(1,1/5) and AIMD(1,1/2)

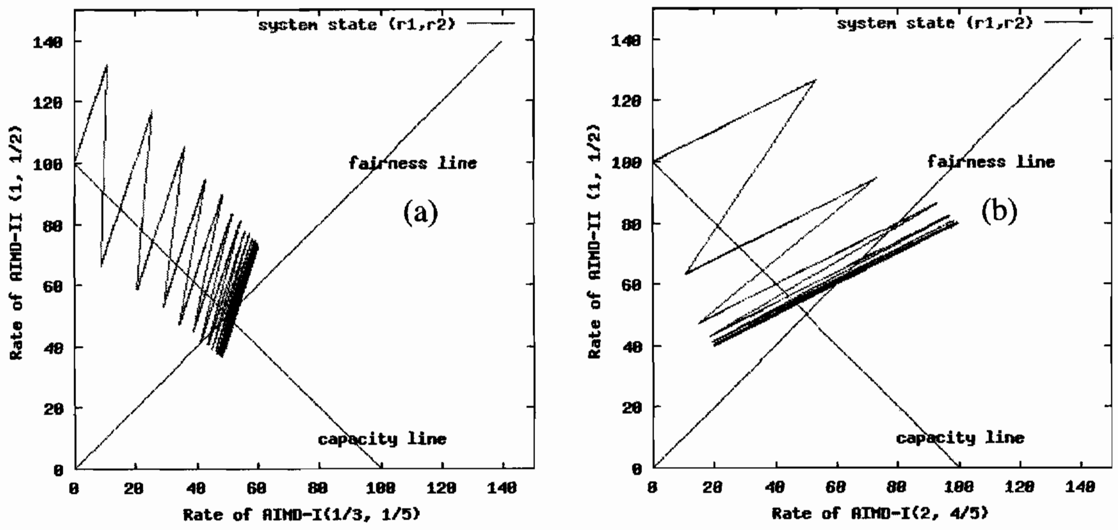


Figure 4.5: Fairness between TCP-friendly flows.
(a) AIMD(1/3,1/5) and AIMD(1,1/2), (b) AIMD(2,4/5) and AIMD(1,1/2)

Figure 4.5 (a) and (b) show two trajectories of systems that use two different sets of TCP-friendly parameters. Here the assumption about the congestion signal is still that it is universal and that all flows perceive it with a common backward delay. We can see that the resulting system states in both systems stabilize around the fairness line, although along different limit cycles.

One interesting aspect of these two trajectories is that the stable states are almost always not on the fairness line except at their centers. This aspect brings back an early issue about the time-scale of fairness. Clearly, even after the system reaches the stable state, the bandwidth share ratio is almost always not fair over small time scales. Only when averaging the share ratio over a period longer than the period of the stable limit cycle, will a TCP-friendly AIMD flow have an average share equal to a normal TCP flow.

With this result, we claim that fairness is only an interesting aspect for flows whose lifetime is longer than the time taken for the system to stabilize and to traverse the stable limit cycle at least once. Thus fairness among AIMD flows should be measured over a time-scale larger than the period of the stable limit cycle T specified in (4.2). This time value is usually many times a flow's RTT. Unfortunately, for most of today's web-traffic, the data content can fit in one or two packets, and thus they last about 3 RTTs (one for SYN and SYN+ACK, one for ACK and Data, and one for FIN and FIN+ACK). HTTP 1.1 has started to use a persistent TCP connection to carry multiple short lived flows as one flow. However, the majority of HTTP flows are still short lived due to the small size of the content transferred. This situation could change in the next few years due to the incremental deployment of multimedia data on the Internet. Until then, fairness is not an interesting aspect among these short-lived web flows.

The period of the limit cycle depends on all participating flows' behavior. We derived the period of the stable limit cycle as in (4.2). This period is related to the parameters of all competing flows. For the details of the derivation, please refer to the derivations of (A1.10) in the Appendix A1.

$$T = \frac{\sum_{j=1}^{N} \frac{\beta_j}{1-\beta_j} r_j}{\sum_{j=1}^{N} \frac{\alpha_j MSS_j}{RTT_j^2}} \qquad (4.2)$$

When all flows have the same α , β , MSS and RTT, the period can be simplified to (4.3).

$$T = \frac{\frac{2\beta}{2-\beta}R}{\alpha \sum_{j=1}^{N} \frac{MSS_i}{RTT_j^2}} + BD + FD \tag{4.3}$$

This result indicates that systems with low feedback delay (*BD*+*FD*) traverse the stable limit cycle fast. So does a system with large α and small β parameters. Low feedback delay, large α parameters, and small β parameters, help to reduce the time-scale lower limit for measuring fairness.

Remember that all the derivations so far are based on the assumption of universal congestion signals. In reality, this assumption may not hold.

4.1.3 Fairness under Non-universal Congestion Signals

In this section, we study the case of non-universal congestion signals, in which not all flows perceive the same congestion signal. Because of the relative complexity of non-universal congestion signals, we first describe the process of congestion signal distribution before discussing the model and results.

Congestion Signal Distribution

In real network systems, packet losses are used as congestion signals, but not all flows get packet losses during a period of congestion. The distribution of congestion signals is not universal among flows but has some randomness. The probability of a flow losing a packet due to buffer overflow depends on the number of packets of that flow arriving at the bottleneck buffer, which is proportional to the flow's sending rate. This behavior covers the majority of real world systems. As advanced techniques such as RED and ECN are deployed, a router process will select which flows will be sent congestion signals to. A simple policy would be to select flows randomly. Again, in this case, the probability of being selected would depend on rate.

Based on this relationship between loss probability and transmission rate, we add a congestion signal distribution process to introduce some randomness into the state-space model. Concretely, we assume that the packet loss events are independent, and the chance of a flow getting a congestion signal is proportional to the amount of data sent by it during the period of congestion[12]. To simplify the discussion, we also assume that all flows use the same packet size (*MSS*).

Our distribution process includes two subsystems, the loss trigger subsystem that determines when a packet loss happens, and the signal distribution subsystem that determines to which flow the lost packet is attributed. The two subsystems include several functions whose return values are all set to zero at the beginning of a congestion.

In the following discussion, we assume that a congestion starts at time t_0, and describe all functions and their meaning one by one.

We first look at the loss trigger subsystem, which uses the loss trigger function L:

$$L(t_0+T)=\frac{\int_{t_0}^{t_0+T}[\sum_{j=1}^{N} r_j(t-FD_j)-R]dt}{MSS} \qquad (4.4),$$

in which t_0 is the start time of a congestion period, T is the time that a congestion has lasted so far, R is the bottleneck link capacity, $r_i(t\text{-}FD_i)$ is the input rate at the bottleneck link from flow i at time t, and FD_i is the flow i's forward delay.

[12] In real networks, the packet loss events are not strictly independent [P99], and flows could use different packet sizes. To verify the result in this section, we conduct real world experiments and present them in Chapter 5. Also, in real networks, a flow's packet loss rate is not exactly proportional to its amount of data transmitted during the congestion when the bottleneck uses tail-drop queue management. However, many recently proposed active queue management techniques, such as RED [FJ93] and BLUE [FKSS99], actually intend to enforce that congestion distribution be proportional to the data rate.

The value of the loss trigger function L is actually the amount[13] of packets lost from the bottleneck since the congestion period started. The factor $\int_{t_0}^{t_0+T}[\sum_{j=1}^{N} r_j(t-FD_j)-R]dt$ is the total amount of data that has been dropped during the period T. Divided by the packet size MSS, it gives us the amount of packets that have been dropped during that period.

The loss trigger function L determines when a packet loss happens. A packet is lost at time t_0, which is the time the congestion started. After that, every time L increases by one packet, it indicates that a new packet is lost.

When the loss trigger function L indicates that a new packet loss happens, the signal distribution subsystem chooses one flow and attributes the packet loss to it. The distribution is based on a per flow probability function $W_i(t+T)$, which is defined as follows:

$$W_i(t_0+T)=\frac{\int_{t_0}^{t_0+T} r_i(t-FD_i)dt}{\sum_{j=1}^{N}[\int_{t_0}^{t_0+T} r_j(t-FD_j)dt]} \qquad (4.5).$$

$W_i(t)$ is the weight used to calculate the chance for flow i to receive a congestion signal. The numerator $\int_{t_0}^{t_0+T} r_i(t-FD_i)dt$ is the total amount of data that has arrived from flow i at the bottleneck during period T. The denominator $\sum_{j=1}^{N}[\int_{t_0}^{t_0+T} r_j(t-FD_j)dt]$ is the total amount of data that has arrived at this bottleneck during the period T. The ratio of these two factors is the portion of the total traffic generated by flow i. We assume the chance of a flow getting a packet loss is proportional to the amount of data the flow sent during the congestion period. Therefore, the chance of a packet loss being allocated to flow i is proportional to the value of $W_i(t+T)$.

The loss distribution chooses one flow to experience a packet loss every time the loss trigger function L indicates a lost packet. The loss distribution process uses a uniform

[13] Notice here, the amount is not necessarily an integer but can include a fraction of one packet.

random generator to generate random values between 0 and 1. The region [0:1] is divided into N contiguous parts. At time t, a flow i would get a $W_i(t)$ size region between [0:1]. If the value generated by the uniform random process happens to be in flow i's region, then flow i loses a packet.

One thing to remember is that the above process just determines the packet loss distribution. The congestion event trigger in the AIMD controller is still under the control of (3.8) described in Chapter 3, which takes RTT into account and limits the congestion signal to at most one per RTT.

System Setup

The congestion distribution process is the only component changed compared to the state space model presented in Chapter 3. After describing the extended congestion distribution process above, now we are ready to describe our target system.

The goal of this study is to check the interaction between the AIMD controller's behaviors and the congestion signal distributions. We first choose a simple setup with only two AIMD flows in the system, and study the fairness between the two flows under non-universal congestion signals. Note two identical flows get the same distribution of congestion signals because they are symmetric with respect to the congestion distribution process. Therefore the interesting case is when the two flows have different AIMD parameters. We start with the case of two different AIMD flows, and at the end of this section, we extend the study to the case with more than two AIMD flows.

We choose a setup such that all flows have the same RTT, and to simplify (4.4) and (4.5), we set the forward delays (*FD*) to be zero so that the backward delays (*BD*) are the same for both flows and are the dominant part of RTT.

4.1.3.1 Fairness between two AIMD flows

We start with two AIMD flows: AIMD-I (1,1/2) and AIMD-II (1/3, 1/5), and then we deal with a wider variety of AIMD flows at the end.

Because of the randomness introduced in the congestion distribution process, the state-space trajectory no longer exhibits a clear limit cycle. We plot a 2-D projection of the system trajectory, with the rate of AIMD-I as one dimension, and the rate of AIMD-II as the other dimension. The trajectory is around the fair share line. However, it's difficult to tell from the figure whether the average sharing ratio between these two dimensions is equal to 1:1 or not.

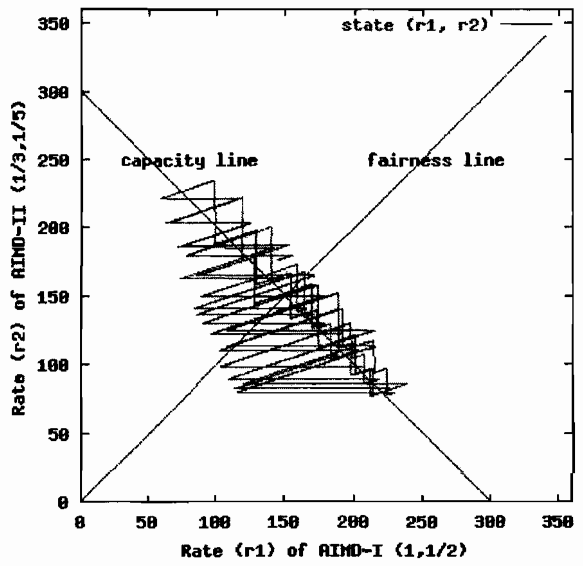


Figure 4.6: State-Space trajectory without a clear limit cycle. two different AIMD flows, non-universal congestion signals

From the trajectory in Figure 4.6 and earlier results, we can see that bandwidth is not evenly shared over short time scales. However, it is unclear whether these flows share bandwidth evenly over long time scales either.

Because of the lack of a visible limit cycle, we are left with two approaches to study the fairness among these flows. The first option is to extend the notion of stable limit cycle to include stochastic stability. The second option is to measure the sharing ratio among competing flows empirically. We are still working on applying stochastic control theory to the stability of bandwidth competitions, therefore we temporarily choose the second option.

As mentioned before, we care only about the fairness at stability. The main difficulty with the empirical approach is that it is not possible to define exactly what constitutes the stable state or when it starts. The process of removing the transient part and letting us focus only on the steady state is called *transient removal*. The way we perform transient removal is to perform long running experiments and truncate the early part of each experiment.

In each experiment using our Simulink model, we set the simulation time to 40 minutes. To measure the average throughput, we truncate the first half of the experiment, and only average the throughput of each flow in the second 20 minutes. Although we have no proof that the system reaches stability within 20 minutes, our experiment observations indicate that 20 minutes is long enough for the flows to adjust to a stable state if the system has stability when the RTT of each flow is only set to 50msec.

The goal of measuring the average throughput is to tell whether the two flows have the same throughput. However, the signal distribution process involves a random generator, and it has an infinite number of seeds. It is not possible to get a perfect estimation of the real average throughput over all seeds. The best we can do is repeat experiments multiple times and get a probabilistic bound for the real average throughput over all seeds. A bound around the measured average is called a *confidence interval*, which indicates the range of the real average. It is a probabilistic bound because the size of the confidence interval relates to the level of confidence we choose, and the confidence level is usually represented by a percentage value. For example, a confidence interval of a 90% confidence indicates that the probability of the real average fit in the confidence interval is 90%. The size of the confidence interval also relates to the number of experiments. Increasing the number of experiments could reduce the size of the confidence interval. If the two confidence intervals of the throughput of the two flows are apart and have no overlap, it assures the two flows have different throughputs with the given confidence level.

Fortunately, it is not necessary to repeat too many samples to get a high confidence probability. In this example, we repeat each study 10 times with different generator seeds, and get a 99% confidence[14] that the average rate of the two flows are different.

Figure 4.7 shows the average throughput measured in each experiment, and the confidence interval for the average throughput of each flow with 99% confidence. The result shows that the two confidence intervals do not overlap, and thus we believe these two flows have different average throughputs in average. This is the case in spite of the choice of α and β according to Equation (2.3) for TCP-friendly flows.

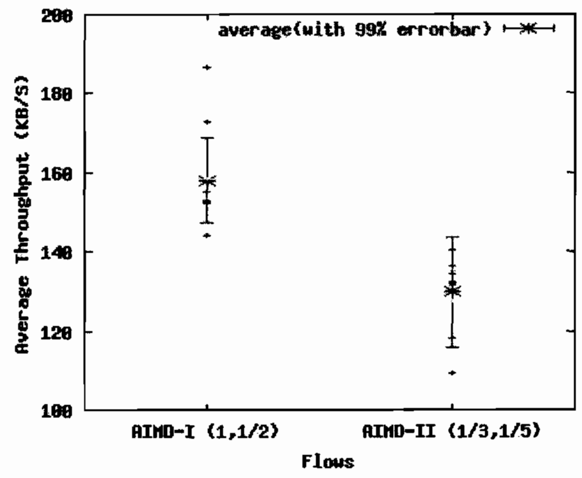


Figure 4.7: Average Throughput of two different AIMD flows
Non-Universal Congestion Signals

4.1.3.2 Fairness between multiple different AIMD flows

In addition to the experiments with two flows, we perform the following experiments with 6 flows just to check the effect on a wide range of AIMD parameters. All 6 flows have different AIMD parameters: (1/3, 1/5) (2/3, 4/11), (1,1/2), (4/3, 8/13), (5/3, 5/7), and

[14] The calculation of confidence interval is available in the Chapter 13 of [J91].

(2, 4/5). We number them from flow-1 to flow-6. All the AIMD parameters of these flows are TCP-friendly according to Equation (2.3).

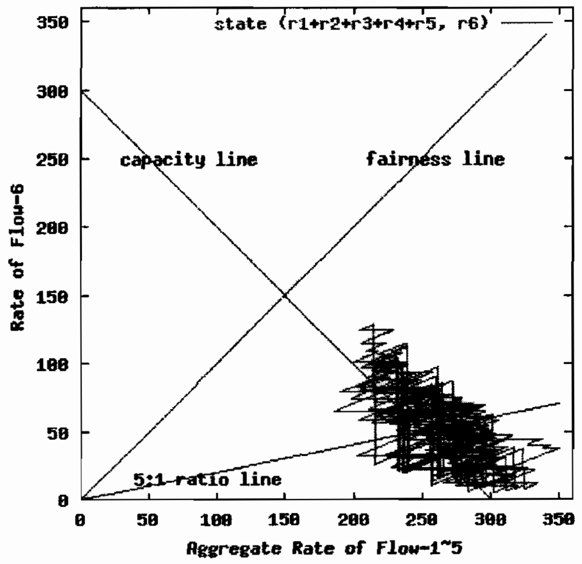


Figure 4.8: State-Space Trajectory without a Clear Limit Cycle
Six different AIMD Flows, Non-universal Congestion Signals

As in section 3.2.5.3, we plot a 2-D projection of the system trajectory shown in Figure 4.8, with the aggregate rate of the flow-1 to flow-5 as one dimension, and the rate of the flow-6 as the other dimension. The trajectory is around the 5:1 ratio line. However, it's hard to tell from the figure whether the average sharing ratio between these two dimensions is equal to 5:1 or not.

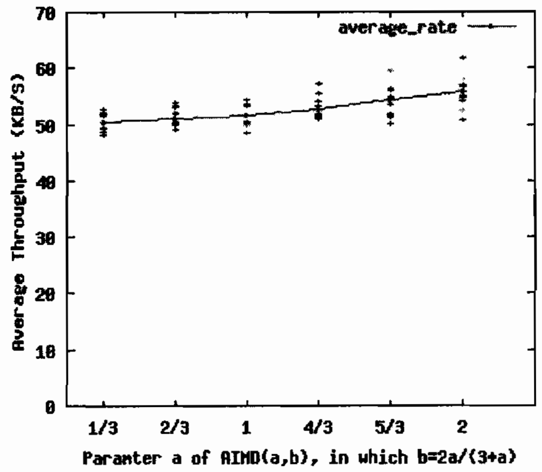


Figure 4.9: Average Throughput versus AIMD Parameters

Since the signal distribution process involves a random number generator, we repeat each study 10 times with different generator seeds. We plot both the average rate of each flow in each run, and also each flow's average rate over multiple runs, in Figure 4.9.

The result shown in Figure 4.9 contradicts our expectation of TCP-friendliness from equation (2.3). We know from section 4.1.2 that flows with different AIMD parameters do not get equal bandwidth share over short time scales, but do share bandwidth evenly over long time scales. However, this claim is under the assumption of universal congestion signals. In the result presented in Figures 4.7 and 4.9, flows with TCP-friendly parameters do not get equal bandwidth share. In particular, flows with larger α get more bandwidth in this example.

According to recent literature [FHP00, YL00], building congestion control according to the equation (2.3) falls into a category of *TCP-friendly congestion control.* Now our study shows that TCP-friendly flows are not friendly to each other in the presence of a non-universal congestion distribution process. With more and more deployment of random drop queue management, we believe non-universal congestion distribution will become dominant. Thus we believe the non-universal congestion distribution process used in our simulations models reality more accurately than a universal congestion distribution process.

We believe the contradiction comes from an unrealistic assumption in the derivation of TCP-friendliness. According to early research on TCP throughput [PFTK98] two flows should get about equal throughput if they have the same average packet loss rate.

The derivation of the TCP-friendly relationship (2.3) is based on the assumption that the congestion signal distribution is independent of the behavior of the congestion control. In reality, this assumption is not true. To explain why the contradiction happens, we plot the congestion perceptions of each flow in the earlier experiments. Here we use two metrics, the packet loss probability of each flow, and the total amount of packet losses over the steady state.

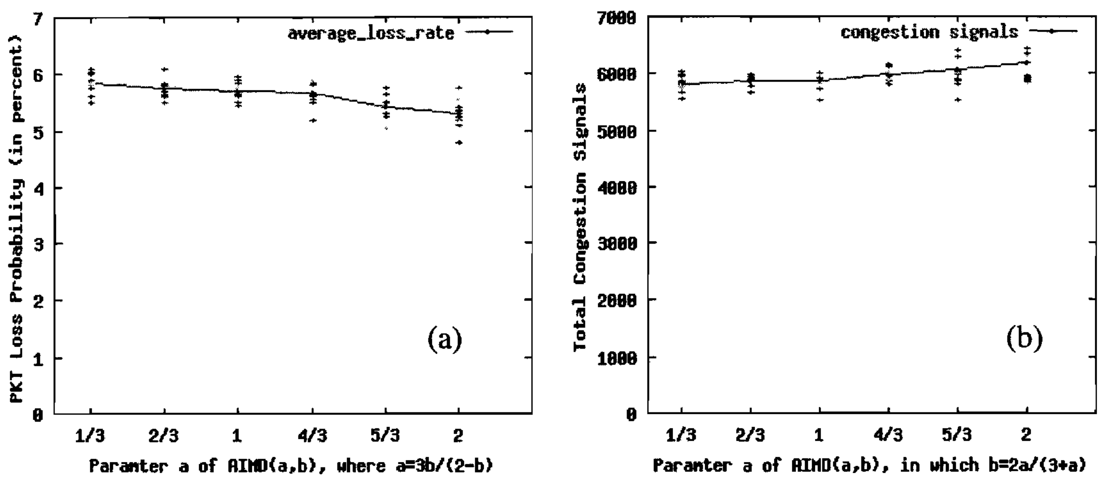


Figure 4.10: Congestion Signals versus AIMD Parameters
(a) Packet loss probability, (b) Total congestion signals over 20 minutes.

Figure 4.10(a) shows the packet loss probability versus the AIMD parameters. As we expected, the perceived loss probability is not the same for all flows. Flows with a small α have high per packet loss rates and thus have lower throughputs (Figure 4.9) in this example. Figure 4.10(b) shows the total number of congestion signals (packet losses) for every flow. Although AIMD flows with larger α have lower per packet loss probability, they have higher throughput. As a result, the AIMD flows with larger alpha end up getting a larger total number of congestion signals. We conclude that the difference in the AIMD parameters makes a difference to their congestion signal perceptions, which in turn affects their average throughput.

4.1.4 Discussion

We conclude the study in this section with the claim that AIMD-based TCP-friendly flows do not necessarily share bandwidth evenly with each other even when they have the same RTT and MSS. Thus they are not friendly to each other. There are several follow up questions related to this result.

The first question is, why they perceive congestion signals differently? The answer is that the congestion signal is distributed based on the flows' transient behavior. Details of this reasoning deserve some further study and is part of our future work.

The second follow up question is: if TCP-friendly AIMD flows do not share bandwidth evenly, how can we predict their bandwidth share ratio? If we can predict the ratio, can we adjust AIMD parameters to achieve a desired ratio?

The experimental results in this section are based on one particular setup. The sharing ratio could be changed when the mixing ratio of AIMD flows and TCP flows changes. However, this study is not to predict the exact ratio of each flow, but to expose the unfairness between TCP-friendly AIMD flows under the exact same environment. Furthermore, adjusting any flow's AIMD parameters could potentially change the occurrence frequency of congestion overall. We don't have a way to predict the precise rate share ratio, which is part of our future work. In the later chapters, we start to study this aspect, and try to make the AIMD parameters tunable to applications so that we can empirically adjust the parameters to check the sharing behavior among flows. That work is still in a very preliminary stage, and a thorough study of predicting the bandwidth sharing dynamics deserves considerable future work.

The studies presented in this section only address a small set of the fairness related problems using a simple model. There are many more issues that are not covered by these studies. We briefly address two additional problems in this section.

First, all study is about TCP-style fairness, but TCP-style of fairness is unfair. The TCP-style fairness is biased to flows with a short round-trip-time, whereas some other scenarios might prefer fairness regardless of RTTs. Thus, we briefly discuss other fairness paradigms in this section.

Second, all study is based on our fluid-based model that does not have a notion of packets, and hence the granularity of measuring any state variable can be arbitrarily small. In real networks, this is simply not true. Data are sent in packets. The size of packets could be a factor that impacts the fairness among flows. In subsection 4.1.4.2, we discuss how packet size might affect our model.

4.1.4.1 Uniform Fairness

Here we make the case that a fairness model other than TCP-style fairness is required. One example scenario is a distributed first-man-shoot game that wants to give players equal advantage no matter what type of network connections they are using, or how far away they are from the server. We call the notion of even bandwidth sharing regardless of RTT, *uniform fairness.*

When assuming universal congestion signals, we can actually tune the fairness of bandwidth sharing to achieve equal bandwidth allocation regardless of RTT by tuning the AIMD parameters. We extend TCP's throughput equation in (2.1) to a general case for an AIMD-based algorithm [LSW01]:

$$r = \sqrt{\frac{\alpha(2-\beta)}{2\beta}} * \frac{MSS}{RTT * \sqrt{p}} \qquad (4.6)$$

Thus, if we adjust all competing flows' α parameters proportionally to the square of RTT, and leave β the same as the default ½, then they can achieve equal bandwidth sharing regardless of their RTTs.

In section 4.1.1, we have shown that a flow (AIMD-I) with 100ms RTT gets four times the throughput of a flow (AIMD-II) with 200ms. When both flows use normal AIMD(1,1/2), the result is shown in Figure 4.11. In Figure 4.12, AIMD-II uses (4,1/2) instead of (1,1/2) as its AIMD parameters. Both figures are produced under the assumption of universal congestion signals.

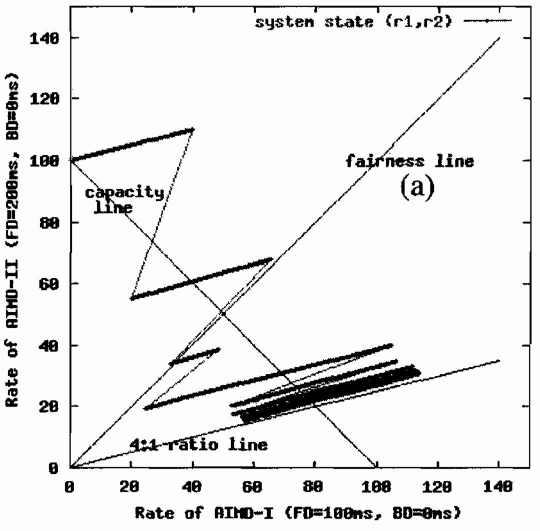


Figure 4.11: Original TCP-style Fairness

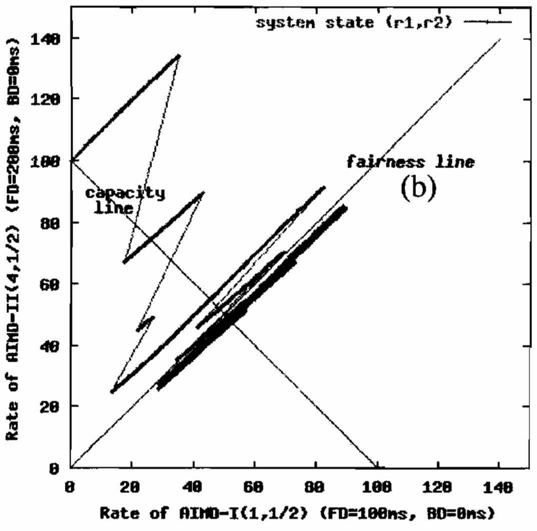


Figure 4.12: Uniform Fairness

Figure 4.11 shows the case where the bandwidth sharing result stabilizes around a point that gives a higher rate for the flow with a small RTT. Figure 4.12 shows the bandwidth sharing result from the same two flows in Figure 4.11 except the one with the longer RTT uses a larger α. The resulting trajectory stabilizes along the fairness line regardless of the RTT differences, and thus uniform fairness is achieved.

The derivation of (4.6) and the above experiments are based on the assumption of universal congestion signals. Under this assumption, we conclude that uniform fairness can be achieved by letting all flows adjust their α to conform to the following simple rule:

$$\frac{\alpha}{RTT^2} = C \qquad (4.7)$$

in which C is an arbitrary positive constant, as long as all the flows use the same C.

However, the assumption of uniform congestion signals is not accurate, as we have shown in 4.1.3. We will study the accuracy of (4.7) to achieve uniform fairness and how to compensate it through real world experiments in later chapters. The compensation could be in the form of adjusting the relationship between α and RTT, and could also be some adjustment to the β parameter.

4.1.4.2 Quantization Effect

The state-space model for AIMD flows is fluid based, and it assumes a flow can send at any available share of the bandwidth. However, TCP and many TCP-friendly congestion control flows do not have sophisticated rate control. They use round-trip-time as the time period over which to control their transmission rates, and they cannot send fractional packets in one round-trip-time. When the bandwidth share for each flow cannot map to an exact number of packets per round-trip-time, some flows round-up to more bandwidth than the ideal share and some flows get less. We call this round-off behavior a *quantization effect* due to packet sizes.

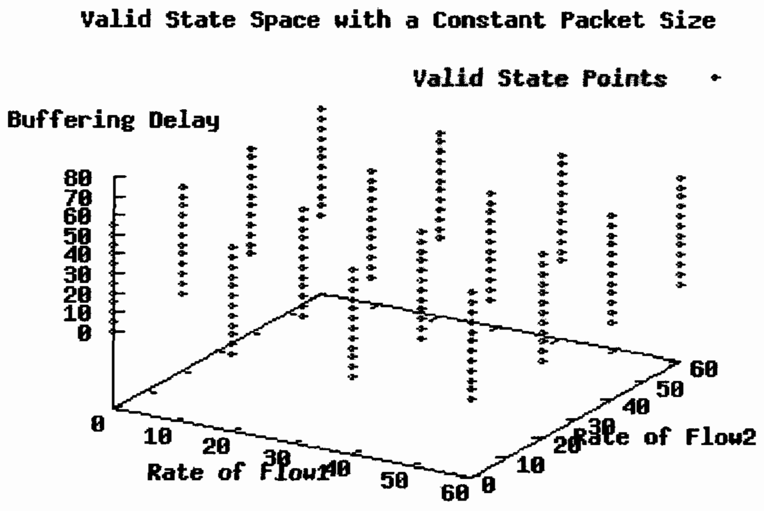


Figure 4.13: A Modified State-space with Only Valid Points

Regarding the state space, this quantization effect limits the valid state spaces to some discrete points rather than the whole non-negative space used in our early work. A new state space for the model is illustrated in Figure 4.13. The system trajectory is no longer an arbitrary trace in the space but a series of transitions between these discrete points.

As one can imagine, the distance between neighboring points is related to the packet size. The state space for a system with a small packet size exhibits denser valid points than one with large packet size. As packet size gets smaller, the valid state space gets closer to the space for the fluid-based model.

To further study the effect of this quantization effect, either experiments in a real network or some extension to the state-space model are required. We plan to do the latter first in our future work.

4.2 Buffering requirement for CBR interactive applications

All the work so far shows that there are oscillations of the system state in a bandwidth competition system. This inherent oscillation during the stable state causes an unavoidable buffering delay for applications that use a constant bit rate (CBR).

4.2.1 Buffering and Adaptations

To study the impact of the dynamics of congestion control on applications, we have to make some assumptions about the application behaviors, because applications could have different preferences regarding the behavior of congestion control. Hence, this section presents the structure of our target application and how adaptation and buffering are used in this application. It then describes the relationships between various adaptation policies and their minimal buffering requirements.

4.2.1.1 Application Structure

Figure 4.14 describes our target application's structure[15]. It includes a data source (e.g., a video camera) and a data sink (e.g., a display) connected through the Internet. The sender side generates data on the fly and sends data to a congestion control protocol through a buffer. Data is transmitted over the Internet under the limit of a congestion control

[15] We assume the application has only one-way traffic. A typical interactive application usually involves two-way traffic, which can be divided to two applications with one-way traffic but with tight dependency on each other.

protocol and is put into a receiver side buffer. The data sink fetches data from the buffer and presents it to users.

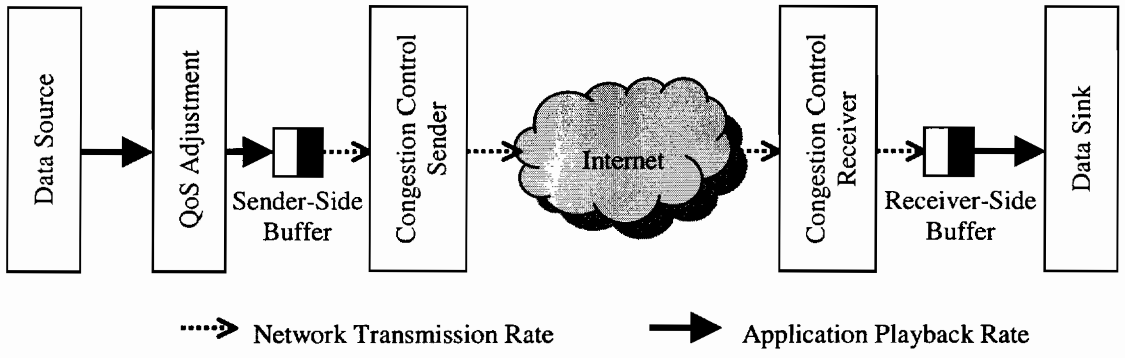

Data Source
QoS Adjustment
Sender-Side Buffer
Congestion Control Sender
Internet
Congestion Control Receiver
Receiver-Side Buffer
Data Sink
Network Transmission Rate
Application Playback Rate

Figure 4.14: A QoS-Adaptive Application over the Internet

The transmission rate over the Internet oscillates over time. To achieve a stable playback quality at the data sink, a receiver-side buffering policy and a sender-side adaptation policy are used in this structure. We assume that a constant playback quality of the application maps to a constant bit rate[16]. Thus, the users' preference of constant playback quality maps to the preference of a constant draining rate from the receiver-side buffer.

The receiver delays the start of playback at the data sink side until enough data has been accumulated in the receiver-side buffer, so that the sink can keep playing even when the network transmission rate drops below the playback rate. As long as the network transmission rate can catch up before the receiver-side buffer reaches empty, the user will not perceive any network rate oscillation. Once the transmission rate is higher than the playback rate the buffer will start to fill again.

Determining what data to send and how to fill the buffer is complex. Applications require smart buffer filling strategies so that all buffered data are useful to compensate for network rate drops in the future. Since buffer management is not the focus of our work, we simply assume that the application can fully utilize all the buffered data. Studies of

[16] In reality, a constant quality could map to a variable bit rate [KW99], which will be more complex but will not invalidate the buffering delay derivation in this work.

smart buffer management approaches can be found in recent research work [FLKP99, FR99, KWLG01, RHE99b].

To reduce the buffering requirements, the target application makes QoS adaptations to adjust sending rate according to the network transmission rate. We assume that the adaptation is fine-grain layer-based, and the application can adapt its rate closely to the network transmission rate. Several research works have shown ways of making fine-grained rate adaptations to the available bandwidth. For example, Jacobs et al. [JE96] adapt encoding parameters according to the available bandwidth; Krasic et al. [KW99] propose a priority-based encoding mechanism and make a scalable rate adjustment for video streams; and more recently, Byers et al. apply a fine-grained rate adaptation [BLM01] to multicast environments.

4.2.1.2 Ideal Adaptation

QoS adaptations are used to compensate for long-term rate variations, and buffering is used to compensate for short-term rate variations. If the congestion control can sustain a constant long-term average, no QoS adaptation is required, and the buffering delay is for short-term variations only. If the congestion control varies the long-term rate, the QoS adaptation compensates for it.

Ideally, QoS adaptation can always find the right long-term average rate of the congestion control. We call this adaptation policy *ideal adaptation*. According to our early study, the steady state of an AIMD congestion control algorithm is a limit cycle in the state space. This limit cycle in the state space maps to a *saw-tooth shape rate behavior* in the time domain. The QoS adaptation is ideal when we assume it knows ahead of time the rate of one saw-tooth in the future. Since it knows one saw-tooth in the future, it can choose the average of the next saw-tooth as its sending rate. Therefore it achieves a stable quality (in the next saw-tooth period) and maximizes the throughput. The buffering requirement for this ideal adaptation policy is the amount of data buffering required to smooth one saw-tooth of the network transmission rate.

Discussing the ideal QoS approach is useful as a benchmark comparison for other approaches, even though it is not totally realistic to implement this ideal QoS adaptation. In reality, there are many QoS adaptation approaches that are close to the ideal adaptation. The discussion of these QoS adaptation approaches is out of the scope of this thesis. Readers can check the QoS adaptation work [KW99, LKW+01] for details of other adaptation policies.

4.2.1.3 Cost for the Ideal Adaptation

This ideal adaptation might not be a preferred adaptation policy by applications. However, as an extreme case of adaptations, it exposes the minimal buffering requirement for maximizing the throughput.

For other "realistic" policies, if they push the application rate to the upper bound of the network transmission rate achievable by congestion control protocols, they require at least the same amount of buffering as this ideal adaptation. This buffering cost is caused by the mechanisms that smooth out the inherent rate oscillation in the congestion control protocols. This buffering is required because the application does not want to oscillate its quality with the rate of congestion control (such as the rate variations within one saw-tooth). This buffering delay could be significant depending on the application's sending rate and round-trip-time. At some point, this buffering delay can become too high for interactive applications. We give a simple derivation for this inherent buffering requirement in Section 4.2.2.

4.2.2 Buffering Requirement for AIMD Congestion Control

In this section, we derive the minimal buffer requirement caused by the rate oscillation during the steady state of an AIMD-based congestion control protocol.

4.2.2.1 Minimal Buffering Requirement

To determine the buffering requirement for smoothing the rate oscillations, we need to describe how the rate of an AIMD-based protocol evolves along time. Figure 4.15 shows an AIMD flow with a playback rate R. For an AIMD flow, the achievable rate in a RTT is its window size divided by the RTT. The window size evolution of an AIMD flow is controlled by the algorithm stated in (2.2): if the window size before a back off is W, the achievable network transmission rate for this flow periodically varies from (1-β)*W / RTT to W/RTT.

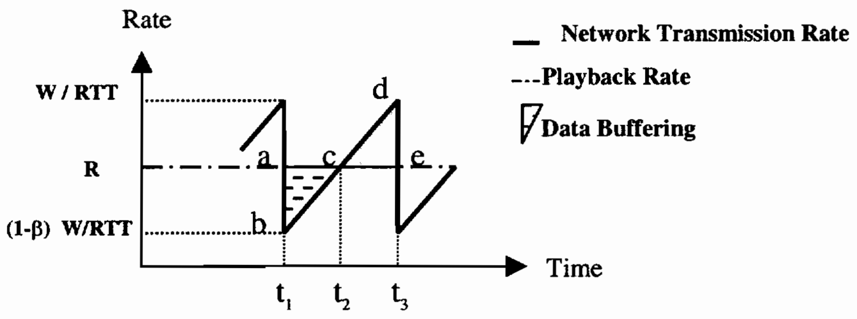


Figure 4.15: Buffering Requirement of an AIMD-based Congestion Control

With an ideal adaptation, the application playback rate is the average of the achievable transmission rate:

$$R = (2\frac{W}{RTT} - \beta\frac{W}{RTT})/2 \qquad (4.8),$$

The application fetches data from the receiver-side buffer at rate R, but the network delivers data to the buffer at the average rate of the saw-tooth shape during its steady limit cycle. Therefore, the data buffering required to smooth the rate oscillations in one saw-tooth is equal to the area of triangle Δabc in Figure 4.15, which is:

$$\Delta abc = \frac{1}{2\alpha MSS} \times (\frac{\beta}{2-\beta})^2 \times R^2 \times RTT^2 \quad (4.9).$$

The details of the derivation are in Appendix A.2.

From this simple derivation, we can see that the buffering requirement is related to the selection of AIMD parameters (α,β). More importantly, this buffering requirement is in proportion to the square of rate and RTT, which is significant for high rate and long RTT applications. This result indicates that interactive applications with tight latency constraints might not want to fully utilize all the available bandwidth in order to avoid this buffering cost. This result also indicates that adjusting the AIMD parameters can change the buffering delay. Early study in this chapter shows that adjusting AIMD parameters can change the bandwidth share ratio between flows. From (4.9), we can see that the adjustment also has an impact on the buffering delay.

With the amount of buffering indicated by (4.9), an application will have a stable playback quality within one saw-tooth period. If the bandwidth share is very stable and saw-tooth shape is evenly repeated along time, then the application keeps a stable quality all the time and utilizes its entire bandwidth share.

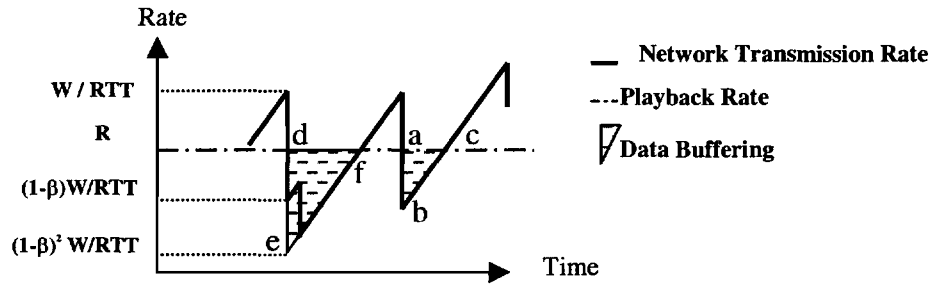


Figure 4.16: Buffering Requirement for Two Closely-spaced Back-offs

However, in the Internet, even a relatively stable bandwidth share would not produce a regularly repeating saw-tooth shape. In the state-space, that means the system stabilizes on a larger time scale than a single limit cycle. In the time domain, the saw-tooth back-offs come closely together for a while, and then spread sparsely for another while. With the ideal adaptation, the application changes its playback quality at every saw-tooth period. If the application prefers a more stable playback quality, it should buffer more data for the rate oscillations caused by closely spaced back-offs, at least to the stability timescale.

Figure 4.16 shows an example of two closely spaced back-offs. If an application wants to keep a stable playback quality when two back-offs happen closely spaced, the buffering requirement would be at most the area of triangle Δdef, which is

$$\Delta def = (3-2\beta)^2 \Delta abc = \frac{1}{2\alpha MSS} \times (\frac{3\beta - 2\beta^2}{2-\beta})^2 \times R^2 \times RTT^2 \qquad (4.10).$$

Similar derivations can be applied to the buffering requirement that is used to smooth more than 2 closely spaced back-offs.

4.2.2.2 Buffering Requirement for AIMD-based TCP-friendly Flows

Early work [FHP00, YL00] has studied how to make AIMD-based congestion control friendly to other TCP traffic in the Internet. A simplified result from the TCP-friendliness study can be expressed as a constraint on its α and β parameters as shown in (2.3). With this α and β relationship, we can refine the buffering requirement to smooth the inherent oscillation of AIMD-based TCP-friendly congestion control as:

$$\Delta abc = \frac{\alpha}{18MSS} \times R^2 \times RTT^2 \qquad (4.11),$$

and the buffering requirement to smooth out two closely spaced backing offs as:

$$\Delta def = (3-2\beta)^2 \Delta abc = (\frac{9-\alpha}{3+\alpha})^2 \times \frac{\alpha}{18MSS} \times R^2 \times RTT^2 \qquad (4.12).$$

Chapter 5

Real World Experiments

The previous two chapters have addressed the dynamic behaviors of bandwidth sharing through a state-space model. While the modeling is important for understanding the impacts of AIMD parameters in theory, real world experiments are important for verifying the modeling result. As discussed in Chapter 3, the state-space modeling approach is based on a few simplifying assumptions, such as viewing flows as continuous fluids and assuming independent packet losses. In the real Internet, flows are delivered in terms of packets, and today, many routers are still using tail-drop queues, with which packet losses show strong inter-dependences.

This chapter describes some experiments with competing flows in a real network environment. The goal of the real-world experiments is to verify the following aspects of the bandwidth sharing behaviors: (1) the timescale of fairness among AIMD flows, (2) the unfairness between TCP-friendly AIMD congestion control protocols, and (3) the possibility of adjusting AIMD parameters to achieve different share ratios other than TCP-style fairness.

To conduct real world experiments of AIMD flows, we need to generate flows that use various AIMD parameters. Unfortunately, no available commercial operating systems support a general AIMD congestion control protocol other than TCP's AIMD(1,1/2). Thus we have to build a general AIMD congestion control protocol ourselves. We call it *adaptive AIMD congestion control.*

5.1 Adaptive AIMD Congestion Control

This section describes the adaptive AIMD congestion control protocol. We first address the motivation for building such a protocol, then we briefly describe the TCP congestion control implementation in the Linux kernel, and how it can be extended to produce an adaptive AIMD congestion control.

5.1.1 Motivation

In addition to the motivation of producing AIMD traffic in a real network environment, the adaptive AIMD congestion control protocol provides a more general and flexible congestion control framework.

Recently, researchers proposed several TCP-friendly congestion control protocols [FHPJ00, TCPF, YL00]. They are claimed to be friendly to TCP based on having the same average throughput over long timescales. Compared to the transmission rate of TCP's additive-increase multiplicative-decrease (AIMD) algorithm, these newly proposed protocols tend to result in a smoother transmission rate, which is good for delay-sensitive applications, such as video conferencing.

The penalty for having a smoother transmission rate than TCP and competing fairly for bandwidth is that these TCP-friendly congestion control protocols respond slower than TCP to bandwidth variations. This limitation comes from the fact that congestion control protocols use data transmissions to detect congestion. In order to quickly probe the spare capacity, the congestion control has to adjust its rate quickly, which leads to a less smooth rate. If it adjusts its rate less frequently in order to preserve a smooth rate, it probes the spare capacity slowly. TCP-friendly protocols make this smoothness improvement by giving up some responsiveness.

Indeed, the tradeoff between responsiveness and smoothness is controllable. TCP-friendly congestion control protocols can have quite different dynamical behaviors

[BBFS01, LSW01, YL00] by tuning their parameters, such as the α and β parameters of an AIMD algorithm. For example, a TCP-friendly AIMD algorithm with a smaller α parameter tends to have a lower responsiveness but a smoother rate than an algorithm with a larger α parameter.

Although the tradeoff is controllable, the current paradigm for building a TCP-friendly congestion control protocol is to choose an algorithm with a fixed set of parameters. This approach is referred to here as *a static protocol.* Using this approach, all applications, at least in the same category, must use the same algorithm. However, a static protocol cannot satisfy the preferences of all applications. A more flexible congestion control protocol is preferred in many scenarios.

We build an adaptive AIMD congestion control protocol in the Linux kernel that exposes the AIMD parameters to user programs through the UNIX socket API. The flexibility of the adaptive AIMD congestion control protocol is shown via experiments presented later in this chapter.

5.1.2 Implementation of Linux Adaptive AIMD

Although TCP has a common protocol specification [APS99, MMFR96], its implementations and their performance vary widely due to TCP's complexity [PAD+99]. Most of the TCP implementations in Unix operating systems are derived from the BSD TCP code, which is very well documented in [WS95]. Our implementation of adaptive AIMD congestion control is based on the Linux TCP, which is implemented independently from the BSD TCP code. Unfortunately, there is no detailed documentation for the Linux TCP implementation. Therefore, we give a brief overview of the implementation of Linux TCP as well as our Adaptive AIMD congestion control protocol based on it[17].

[17] The Linux kernel we used is Linux 2.4.16. The TCP implementation of 2.4.x is significantly different from early Linux TCP implementations in 1.2.x, 2.0.x, and 2.2.x. A Linux 2.4.x kernel has a New Reno style of TCP with SACK option implemented.

5.1.2.1 Kernel Implementation

The Linux TCP implementation uses a *snd_cwnd* variable as the congestion window size for each TCP flow. The variable *snd_cwnd* is in units of packets, and TCP achieves its congestion control by limiting the value of *snd_cwnd*. The Linux TCP is event based, and *snd_cwnd* is updated upon either an acknowledgement (ACK) arrival event or a timeout event. Two algorithms are involved in the *snd_cwnd* updates. The TCP AIMD algorithm is related to the ACK arrival event, and the TCP SlowStart algorithm is triggered by the timeout event.

TCP potentially adjusts the *snd_cwnd* value on the arrival of every ACK, either increasing or decreasing it. During the non-congestion period, Linux TCP increases the *snd_cwnd* by one packet after receiving every CWND number of ACKs, as indicated by the *tcp_cong_avoid* function shown in Figure 5.1.

```
static __inline__ void tcp_cong_avoid(struct tcp_opt *tp)
{
  if (tp->snd_cwnd <= tp->snd_ssthresh) {
        /* Slow Start: Increase cwnd by 1 for every ACK */
        if (tp->snd_cwnd < tp->snd_cwnd_clamp)
                tp->snd_cwnd++;
  } else {
        /* Additive Increase: cwnd = cwnd + 1/cwnd */
        if (tp->snd_cwnd_cnt >= tp->snd_cwnd) {
                if (tp->snd_cwnd < tp->snd_cwnd_clamp)
                        tp->snd_cwnd++;
                tp->snd_cwnd_cnt=0;
        } else
                tp->snd_cwnd_cnt++;
  }
```

Figure 5.1 Slow Start and Additive Increase (from linux/net/ipv4/tcp_input.c)

When a congestion signal is detected, by either an ECN or a triple-duplicate-ACK, the Linux TCP makes the multiplicative decrease in the *snd_cwnd* in two steps. The first step occurs upon the detection of a congestion signal, and involves setting the variable *snd_ssthresh* to be half of the current *snd_cwnd* value. The second step happens when the

Linux TCP detects the end of congestion, by either the end of congestion notification marks (ECN) or the advance of acknowledged sequences (in the triple-duplicate-ACK case). In both cases, the Linux TCP calls the function *tcp_complete_cwr* that sets *snd_cwnd* to *snd_ssthresh*. The code segments of the two steps are shown in Figure 5.2.

```
void tcp_enter_loss(struct sock *sk, int how)
            ... ...

/ * Enter Loss state (by ECN or duplicated Acks)
  * reset ssthresh to half
  */
if (tp->ca_state <= TCP_CA_Disorder ||
   tp->snd_una == tp->high_seq ||
   (tp->ca_state == TCP_CA_Loss && !tp->retransmits)) {
          tp->prior_ssthresh = tcp_current_ssthresh(tp);
          tp->snd_ssthresh = tcp_recalc_ssthresh(tp);
}

            ... ...

static inline __u32 tcp_recalc_ssthresh(struct tcp_opt *tp)
{
          return max(tp->snd_cwnd >> 1U, 2U);

}

            ... ...

/* Change window size to the ssthresh value, which is
 * half of the original window size
 */
static __inline__ void tcp_complete_cwr(struct tcp_opt *tp)
{
          tp->snd_cwnd = min(tp->snd_cwnd, tp->snd_ssthresh);
          tp->snd_cwnd_stamp = tcp_time_stamp;
}
```

Figure 5.2 Multiplicative Decrease (from linux/net/ipv4/tcp_input.c)

Once we understand the TCP implementation in the Linux kernel, the extension to produce adaptive AIMD congestion control is straightforward. Instead of increasing the *snd_cwnd* value by one packet for a window advance, it is increased by α packets[18]. Instead of reducing the *snd_cwnd* by half, it reduces it by β times.

In the implementation of adaptive AIMD, TCP SlowStart and timeout are unchanged. At the transition between SlowStart and AIMD, the congestion window size is still cut by half regardless of what β value users set. Preserving this action unchanged regardless of the AIMD parameters guarantees that all adaptive AIMD flows have the same SlowStart behavior. A more flexible adaptive congestion control approach could include adjusting the SlowStart expanding ratio, or using MIMD with a small expanding ratio at the steady state as in SubTCP [FKSS97].

5.1.2.2 Application Programming Interface

To expose the α and β parameters to user level programs, we add two options: TCP AIMD_ALPHA and AIMD_BETA to the Unix socket API. User programs can adjust the congestion control parameters through the socket option system calls. The APIs of these two socket options are shown in Figure 5.3.

```
int  getsockopt (int sock,
                 int level,          /* SOL_TCP */
                 int option,         /* AIMD_ALPHA or AIMD_BETA */
                 void *value,
                 socklen_t len);

int  setsockopt (int sock,
                 int level,          /* SOL_TCP */
                 int option,         /* AIMD_ALPHA or AIMD_BETA */
                 const void *value,
                 socklen_t len);
```

Figure 5.3: Socket Option API of the Adaptive AIMD

In addition to the per-socket based adaptive AIMD system calls, we also introduce a system wide Adaptive AIMD interface through the Linux Proc file system [Linux]. Users

[18] The congestion window size is no longer an integer value, hence the rate limitation of the congestion control part is now based on its truncated integer value.

with root privilege can read and write the system wide AIMD settings by reading or writing to the two proc files: /proc/sys/net/ipv4/tcp_aimd_alpha and /proc/sys/net/ipv4/tcp_aimd_beta.

To illustrate the effect of the parameter on the window size adjustments, we run a group of adaptive AIMD flows with their parameters adjusted through the socket API, and we capture their *snd_cwnd* migrations by periodically sampling the kernel status. Figure 5.4 shows a snapshot of the *snd_cwnd* migrations of two competing flows: AIMD(3,1/2) and AIMD(1,1/2). It's clear that AIMD(3,1/2) increases its *snd_cwnd* value 3 times faster than AIMD(3,1/2), whereas they back off their *snd_cwnd* in the same ratio. Also we notice that both flows do not back off synchronously, and since AIMD(3,1/2) gains more bandwidth, it experiences more congestion signals than AIMD(1,1/2) during the period shown in Figure 5.4. We will address the impact of this issue in more detail in latter experiments.

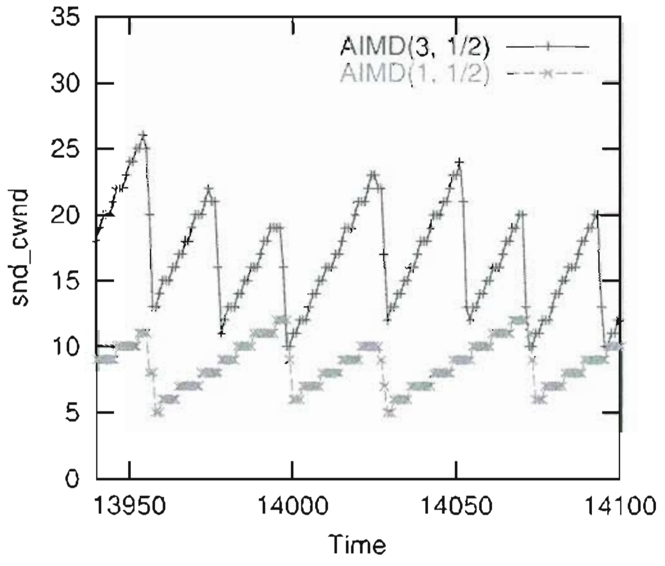


Figure 5.4: Congestion window size of an Adaptive AIMD flow and a normal TCP flow

We plot the congestion window migrations of more mixed flows in Figure 5.5. We plot the congestion window of AIMD(3,1/2), AIMD(1,1/2), AIMD(3,4/5), and AIMD(1,4/5) to show the impact of both parameters to verify the correctness of our implementation.

As expected, AIMD(3,1/2) and AIMD(3,4/5) have the same increment pace, but a different back off ratio. So do AIMD(1,1/2) and AIMD(1, 4/5).

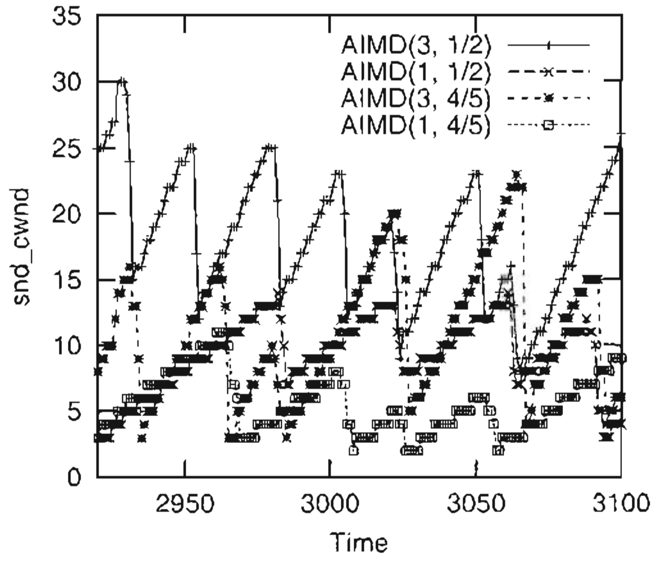


Figure 5.5: Congestion window of more Adaptive AIMD flows

All the results shown in Figure 5.4 and Figure 5.5 indicate that the Linux implementation of the Adaptive AIMD behaves as we expected. With this implementation, we can generate various AIMD flows in our real network test-bed, and conduct experiments on the bandwidth sharing dynamic among AIMD flows.

5.1.2.3 Discussion

The major reason to expose the AIMD control parameters to user space is to measure the effect of AIMD parameters on the bandwidth sharing dynamics. However, this approach potentially lets users produce flows that are more aggressive than TCP since α can be set large and β can be set small. We are aware of this danger. However, we do not think it introduces any more danger than the already available UDP interface. Furthermore, we recommend an alternative user interface, in addition to the above API in Figure 5.3, in which the relationship between α and β can be constrained appropriately, to provide a safer use of the adaptive AIMD congestion control. This alternative interface is called TCP-friendly AIMD congestion control, which only exposes one parameter, such as α, to the user space. Whenever the α parameter is adjusted, the β parameter is adjusted according to some "TCP-friendly" rules, such as (2.3). Some of the experiments in this chapter verify the effect of this "TCP-friendly" parameter adjustment.

5.2 Experiment Setup

With the development of the Linux adaptive AIMD congestion control protocol, we perform a study of the bandwidth sharing behavior among AIMD flows in a real network environment.

The experiments are conducted in a controlled network environment, which is a Linux 2.4 test-bed that simulates a WAN network. In this environment, we can control the number of flows, the position of the bottleneck, and the bottleneck capacity (including rate, delay, and buffer space). All our experiments use a single common bottleneck with FIFO scheduling.

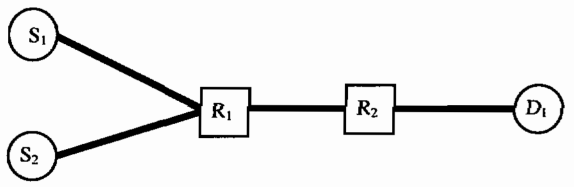


Figure 5.6: Network Topology

The network topology is shown in Figure 5.6. Each node in the figure is a separate Linux machine. Two of the Linux machines, S_1 and S_2, are traffic senders. The machine D_1 is the receiver of all the traffic. The two machines in the middle, R_1 and R_2, act as routers by running NISTNet [NIST], a network emulation program that allows the introduction of additional delay and bandwidth constraints on the network path. Router R_1 is used to introduce additional delay to flows. We control R_1 so that it can introduce different delays to various flows if required. Router R_2 is used as a common bottleneck for all flows, and by default, it uses a tail-drop queue management approach.

In our experiments, we use a flow-generating tool called *flowpair* [FP] to study the impact of the AIMD parameters on performance aspects, such as the flow's throughput. *Flowpair* generates a group of flows from the source machines, S_1 and S_2, to the destination machine D_1. It exposes control knobs of the AIMD parameters for each flow, and indicates the relative progress of each flow by showing the amount of data delivered.

5.3 Experiments and Results

In this section, we describe details of our experiments and their results. The first question we try to answer is: *In what timescales should we measure the throughput in order to judge the inter-flow fairness?* We answer this question by showing the bandwidth share ratio of identical flows in different timescales. The second question we try to answer is: *Do theoretically friendly AIMD flows share bandwidth evenly with TCP flows as predicted by (2.3), or do they share bandwidth unevenly as predicted by the simulation in Chapter 4?* In the experiments, we adjust the AIMD parameters of the competing flows and show the bandwidth share ratio. We also take measurements of the congestion events experienced by competing flows. The third question we try to answer is: *Can we tune the AIMD parameters to produce a different share ratio? For example, can a flow be adjusted to compensate for the TCP RTT bias and achieve the "uniform fairness" behavior introduced in section 4.1.4?* The following sections address these three problems.

5.3.1 Fairness Time-scales

The goal of the experiment in this section is to see the effect of the measurement timescales on bandwidth sharing fairness. In chapter 4, the theoretical study shows that the limit cycle period is a minimum measurement timescale, even under the circumstance of identical AIMD flows. Measurement in a timescale smaller than the limit cycle period would lead to an unfair share ratio.

We measure the bandwidth share of each flow in various timescales. To compare the measured fairness among different timescales, we use the following fairness index:

$$F(T) = \frac{[\sum_{i=1}^{N} x_i(T)]^2}{N \sum_{i=1}^{N} x_i^2(T)} \qquad (5.1),$$

in which *N* is the number of flows, *T* is a measurement timescale, and $x_i(T)$ is the bandwidth share of the *i-th* flow measured in the timescale *T*. The fairness index *F(T)* is a value between *1/N* and 1. When the bandwidth share is extremely unfair, for example only one flow gets all the bandwidth and other flows get nothing, the value of *F(T)* is equal to *1/N*. As the bandwidth shares of flows get closer, *F(T)* increases. And when the bandwidth share is perfectly even among competing flows, *F(T)* is equal to 1. Our experiment is to see how the timescales affect the fairness index.

Measuring the bandwidth share ratio between two competing flows is not simple. The share ratios vary from one experiment to another even with the same setup. Many components of the test-bed use a random process in their control, for example, the Ethernet backoff timer and the random queue drop selection. These random processes, plus the randomness introduced by the CPU scheduler, prevent the system from generating exactly the same packet output at the millisecond level from multiple flows. The behavior of TCP congestion control is sensitive to the timing and order of packet deliveries. To compensate for this randomness and noise, every experiment has to be repeated multiple times, and the mean value and the standard deviations of all measurements are presented.

From the average behaviors, our result shows that measuring in timescales longer than the theoretic limit cycle period can give an approximate fairness, while measuring in timescales shorter than the limit cycle period leads to obvious unfairness.

5.3.1.1 No Perfect Even Share

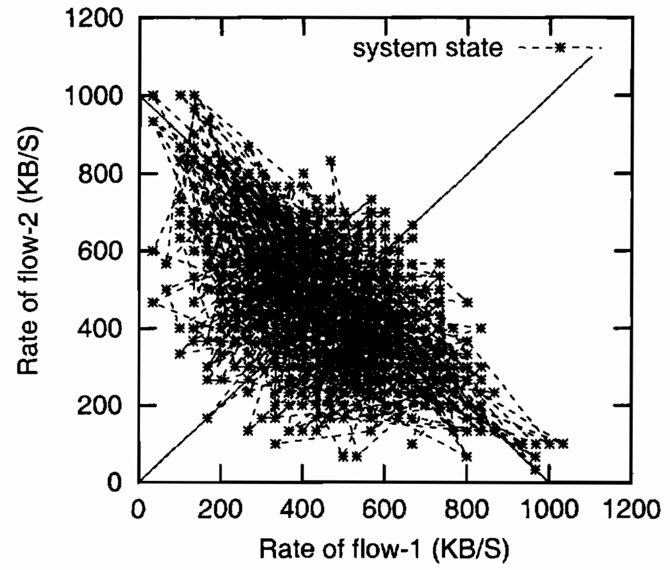


Figure 5.7: The System Trajectory Monitored in a Real Network

We first run only two competing flows, and measure their bandwidth share ratios. In this experiment, the bottleneck link bandwidth is set to 1MB/S, the bottleneck queue size is set to 33 packets, the round-trip-time is 50ms, and the packet size (MSS) is 1500 Bytes. Theoretically, the system stabilizes on a limit cycle, and the limit cycle period is 300ms according to the following equation (5.2). This equation is derived from (4.3) combined with the conditions of $\alpha = 1$ and $\beta=0.5$.

$$T = \frac{\frac{1}{3} \times \frac{R}{N}}{\frac{MSS}{RTT^2}} + \frac{RTT}{2} \qquad (5.2)$$

We plot a part of the system trajectory in Figure 5.7. Because of the packet granularity and other factors discussed in Section 4.1.4, the trajectory is not as clear as the one in Figure 4.1.

Since we don't have the ability to extract out visually the shape of the trajectory, we go ahead to measure the fairness directly using the fairness index in (5.1). All the measurements are started 20 minutes after the flows have started sending data, to truncate the flow startup behavior, as we did in section 4.3.1.1. In this experiment, the bottleneck rate and the bottleneck queue size are carefully chosen to make sure all flows are running in AIMD steady state and thus that we are measuring AIMD flows rather than the TCP slowstart. During the experiments, we monitor the timeout behaviors, and make sure no timeouts happen. We study the impact of timeouts in later experiments in a severe congestion scenario.

The measurement timescales we choose are from 20ms to 2000 seconds, which covers the lifetime of most Internet flows. For each timescale measurement over each time period is repeated multiple times, evenly distributed over the flows' lifetime[19]. In the experiment result, we show 100 samples for each selected timescale.

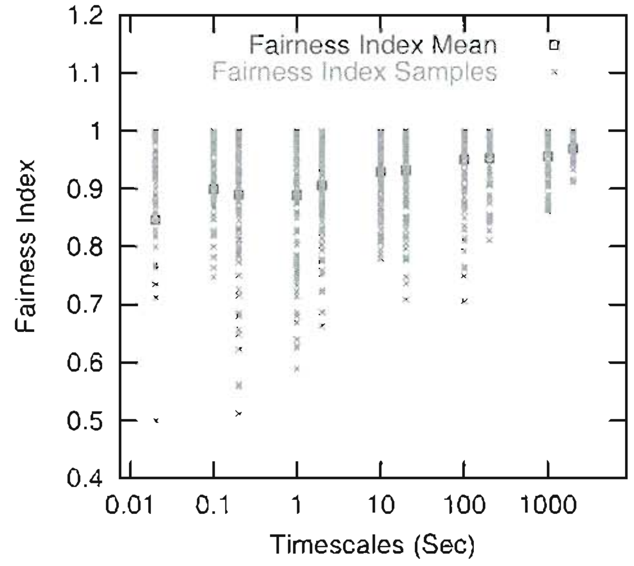


Figure 5.8: Fairness for two identical competing flows in different timescales

The bandwidth share measurement is conducted based on the traffic captured in the output of the router R2. For every selected timescale τ, the number of packets of each flow is counted as $x_i(\tau)$, which is fed into the equation (5.1) to calculate the fairness index $F(\tau)$.

[19] Since all measurements are for one connection, they are not really independent measurements. We think it is still appropriate because the goal of the measurement is to check the effect of measurements of the same object over different time scales.

The result of the fairness index versus various timescales is presented in Figure 5.8, which shows uneven share ratios over all time scales. The fairness index gets closer to 1 as timescales increase. However, the bandwidth shares between the two competing flows are never perfectly equal, even in the timescale of 2000 seconds. This result indicates that the notion of TCP-friendliness is not meaningful if it is defined as exactly even share of bandwidth. In real systems, fairness among competing flows is a notion based on a statistical average, not deterministic one.

The fairness index of the two flows is bounded between 0.5 and 1, which might be too narrow to show the differences resulting from different timescales. We repeat the above experiment twice, with 20 flows and 200 flows respectively. The experimental setup is the same except that the bottleneck rate and the bottleneck queue size increase in proportion to the flow numbers (10MB/S and 330 packets for 20flows, and 100MB/S and 3300 packets for 200 flows). The reason for these adjustments is to keep the same average share for all flows in all three experiments for easy comparison. The results of fairness index versus timescales for 20 and 200 flows are shown in Figures 5.9 and 5.10 respectively.

Figures 5.9 and 5.10 show that the fairness index is not equal to 1 even over very large timescales, as shown by Figure 5.8. When more flows are used, the effective range of the fairness index expands, and thus Figures 5.9 and 5.10 show more unfairness over small timescales, where we expect unfairness.

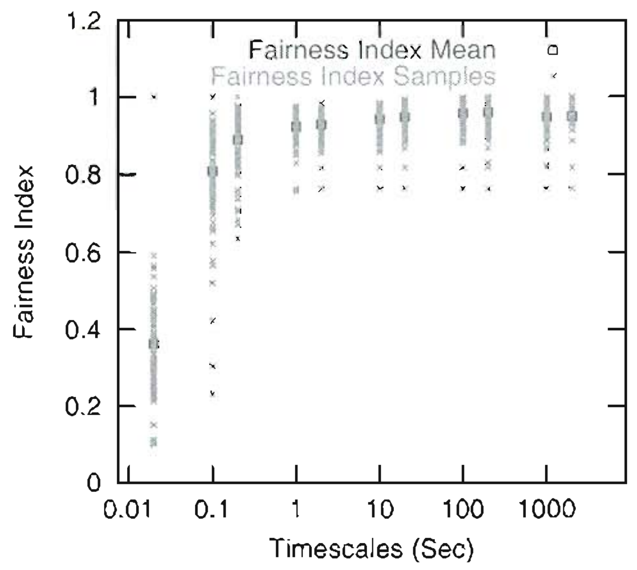


Figure 5.9: Fairness for 20 identical competing flows

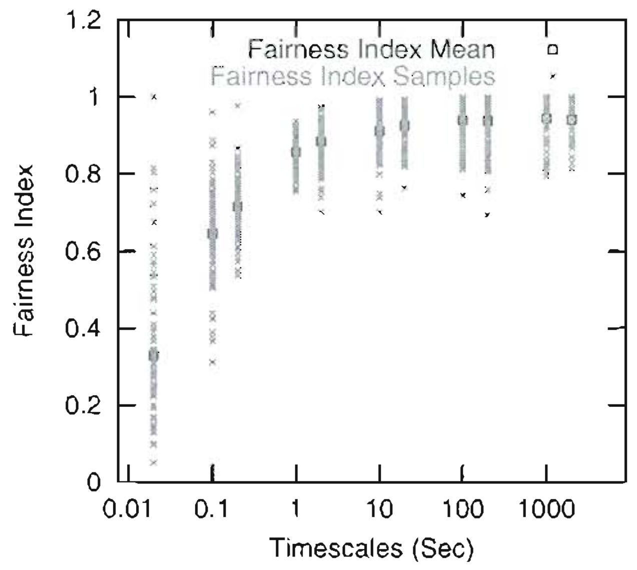


Figure 5.10: Fairness for 200 identical competing flows

5.3.1.2 Fairness over the Theoretical Limit Cycle Period.

Although we know that the theoretical limit cycle doesn't match the real system limit cycle, we still want to check whether the theoretical limit cycle period can be used as a guidance (at least as a lower bound) for choosing the right measurement timescales, because intuitively the real limit cycle period should be longer than the theoretical one.

It is hard to judge what timescale is the cutting line between good and bad measurement timescales, because the fairness index is not 1 even with very large timescales. We decide to compare the fairness index measured in the theoretical limit cycle period to the fairness index measured in the largest timescale (2000 seconds) used in our experiments.

We repeat the experiments of 20 flows with RTTs equal to 5ms, 10ms, 20ms, 50ms, 100ms, 200ms and 500ms respectively. The corresponding theoretical limit cycle periods of these RTTs are 30ms, 60ms, 120ms, 300ms, 600ms, 1200ms, and 3000ms, We plot the mean fairness index measured in the theoretic limit cycle period *T*, as well as the one measured in *T/2*, *T/10*, and 2000 seconds. The result is shown in Figure 5.11.

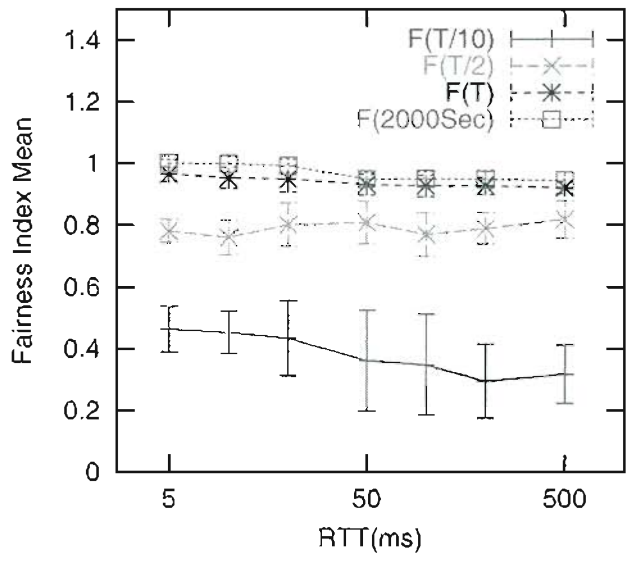


Figure 5.11: Fairness Index versus RTT

Figure 5.11 shows that *F(T)*, the mean fairness measured in the theoretical limit cycle *T*, is very close to the fairness index measured in the maximum timescale we used, whereas the measurements in T/2 and T/10 are much less fair than *F(T)*. Therefore, although the system has a more complex trajectory than the one in our model, the limit cycle period T is still a good guideline for choosing the measurement timescales.

5.3.1.3 Discussion

The results from this section indicate that bandwidth shares among identical flows are not strictly equal even over large timescales. The round-trip-time and the number of flows do not change this result. The exact causes of this uneven sharing behavior are still unclear to us. Recent studies [PA02, V00] of this problem using control theory and complex system theory hint that the bandwidth sharing among packet based flows might be subject to the *butterfly effect*, and thus a small disturbance, such as the end system interrupt timer, can significantly change the system's exact behavior. However, fairness measured in the theoretical limit cycle timescale is still close to fairness in the large timescale. This result indicates, although does not prove, that the average behavior is close to the theoretic predictions.

In the following experiments, we decide to use a fixed timescale, 1000 sec, to measure fairness. This timescale is guaranteed to be larger than the theoretical limit cycle period in all our experiments. Plus, most of today's long lived flows are in the order of tens of minutes, which is about this selected timescale. Friendliness or Fairness is then judged by comparing the bandwidth share ratio measured in this timescale.

5.3.2 Share among "TCP-friendly" AIMD flows

The intention of the experiments in this section is to verify the impact of "TCP-friendly" AIMD parameters on the throughput and the sharing ratio of competing flows.

Recent studies [PFTK98, PFTK99, NST00] of TCP show that a few factors have a significant impact on a flow's throughput. These factors are:

- Number of congestion signals (ECN or triple-dup-ACK) experienced by a flow,

- Number of timeouts experienced by a flow,
- The socket buffer size.

The values of these factors[20], as well as their impacts on a flow's throughput, depend on the experimental scenarios, including the bottleneck rate, RTT and the bottleneck buffer capacity. For example, if the bottleneck's rate-delay product is larger than the sum of the receiver-side socket buffer sizes[21] of all competing flows, then only the flow control of all the competing flows matters. If they have the same socket buffer size, they share bandwidth evenly no matter what congestion control parameters they use. Therefore, for the scenarios where congestions seldom happen, AIMD parameters do not affect the throughputs of competing flows and their bandwidth share ratio.

In this chapter, we only study scenarios where congestion happens at least occasionally. Therefore, we ensure that the socket buffer of any flow is large enough, so that the congestion control protocol determines the bandwidth share ratio.

The impact of AIMD parameters on the bandwidth competition would is through the measured bandwidth share ratio, the link utilization, and the number of congestion signals and timeouts. Here we refer to every group of ECN packets or duplicated ACKs that cause a congestion window back off as *a congestion signal event*. We refer to every TCP retransmission timeout that triggers TCP SlowStart as *a timeout event*. In our measurements, the congestion signals do not include timeouts.

Both congestion signal and timeout events are important to determine a flow's network rate behavior. Timeouts become especially significant in the heavily congested scenarios.

[20] The socket buffer sizes in most systems are static, so their values are not affected by the running scenarios. However, some recent research [SMM98, GKLW02] has proposed automatic socket buffer tuning. Therefore, it is possible that this size is also dynamically changing according to the running scenario.

[21] By default, TCP's socket buffer size is set to 64KB in Linux 2.4. For the typical 1550 Bytes MTU size (1448 Bytes application data), this buffer size is about 45 packets. Because of flow control, TCP's congestion window size will not increase over this buffer size limit. Once TCP's congestion window size reaches 45, the TCP flow control would be the only mechanism that controls the transmission rate. Thus, by default, TCP's maximum congestion window is 45 packets.

However, every timeout event triggers a SlowStart, which is not modeled in our simulation and modeling work in Chapters 3 and 4. Since we want to verify the result from our simulations, and we want to cover as many congestion scenarios as possible, we divide all the experiments into two scenarios: I) *light congestion*, and II) *severe congestion*, so that we can avoid mixing the effect of timeouts and congestion signals. We control the network setup to ensure zero timeouts happen in the light congestion scenario, so that it can be used to verify the simulation results in Chapter 4. The severe congestion scenario is designed to study the impacts of timeouts.

The difference between light and severe congestion is the buffer capacity of the bottleneck router. In the light congestion case, the bottleneck buffer capacity is set to *20*N* packets, where *N* is the number of competing TCP flows. We choose 20 packets as the average window size per flow, because 20 is about half of 45 packets, which is the default socket buffer size. A 20 packet average window gives a flow's congestion control a large range of actions for expanding or contracting without giving control to the flow control. In the severe congestion control case, we set the bottleneck router's buffer capacity to *5*N* packets, so that the congestion control of each flow has only a small range for adjusting its congestion window, and the window size is very close to its initial window size. Timeouts are more likely to happen with a small window size.

Each running scenario is further divided into two sub-cases based on the queue management schemes used in the bottleneck router. The two queue management schemes are a tail drop queue and a random drop queue with ECN support. We discovered that the two queue management approaches correspond to the two different assumptions about congestion signal distribution that we made in our simulations in Chapter 4. The tail-drop queue case corresponds to the assumption of universal-congestion, whereas a random-drop queue corresponds to the non-universal congestion signal assumption.

All flows in this section use the "TCP-friendly" equation (3.1). Because this equation controls the relationship between α and β, a single one of the parameters can determine the flow's behavior. In this section, we choose to use *TCP-friendly AIMD(α)* to represent the "TCP-friendly" AIMD(α,β) flow.

The result of our experiments indicates that the impact of AIMD parameters is different in the light and heavy congestion, and between the two queue management schemes. Here we briefly summarize this result, prior to presenting the details of the experiment results.

- When the congestion is not severe, in the sense that timeouts seldom happen, the theoretical TCP-friendly AIMD flows get bandwidth equal to normal TCP flows under tail-drop queue management at the bottleneck. Under the severe congestion with a random-early drop queue, the theoretical TCP-friendly AIMD flows get bandwidth shares that are not equal to those of normal TCP flows. Specifically, the AIMD flows with a smaller α get less bandwidth than the ones with a larger α. This result matches our simulation result that the TCP-friendly relationship for α and β is accurate only with universal congestion signals (with tail drop queues), and is not accurate with non-universal congestion signal (with random drop queues). With more and more deployment of active queue managements, the latter case will be the dominant one in the future.

- When the congestion is severe in the sense that a lot of timeouts happen, the theoretical TCP-friendly AIMD flows get unequal share compared to a normal TCP flow. AIMD flows with larger α tend to get more bandwidth. Since the buffer capacity is very limited in the severe congestion case, the random drop queue with ECN and the tail drop queue produce the same results. Our theoretical studies do not cover this scenario, since the behavior is dominated by TCP's SlowStart algorithm, which we did not model. This result indicates that timeouts are important when modeling severe congestions, and is an area for future research.

5.3.2.1 Two Flows

We start with a system that has only 2 flows: one normal TCP flow, one TCP-friendly AIMD(α) flow. To show the impact of AIMD parameters, we choose a group of different TCP-friendly AIMD(α) flows and measure their bandwidth sharing behavior with a normal TCP flow. The α parameters we chose for the experiments range from 0.2 to 2.8 with 0.2 as the step increase[22].

We measure the bandwidth share ratio between the TCP flow and the TCP-friendly AIMD(α) flow, the link utilization, the congestion signals, and the timeouts in each running scenario. The bandwidth share ratio is measured over a timescale of 1000 seconds. We do not use the fairness index here because it can not tell us which flow gets more bandwidth, whereas the bandwidth share ratio can. The link utilization is measured as the percentage of the bottleneck rate that represents application level throughput. The number of congestion signals and timeouts are measured in the kernel TCP stack. No matter how many duplicated ACKs or ECN packets it receives, every time a flow backs off its window we count it as one congestion signal event[23]. This behavior corresponds to TCP NewReno, which is the one used in these experiments. Every time a kernel TCP retransmission timeout happens and triggers a SlowStart, we count it as a timeout event. The congestion signals and timeouts are presented in the results by the average number of events in every 10 seconds period[24].

Due to the limited time to perform experiments, we repeated each measurement 20 times. As a result, the experiment takes 6 days for each running scenario. We present the mean and the standard deviation of each set of measurements in the results. The results are presented in Figures 5.12 ~ 5.20.

[22] We choose this range between 0 to 3 because (1) α is required to be larger than zero to do additive increase, (2) α can not be larger than 3 since otherwise it causes a negative β according to (2.3), where $0<\beta<1$ is required.

[23] TCP can trigger a back off by mistake when reordering happens in the network. Therefore, the measured congestion events are not necessarily the exact number of congestion signals as defined in Chapter 4. However, we believe the reordering rarely happens in our test-bed. Therefore, we believe using the kernel state is an accurate reflection of the flow's congestion experience.

Scenario I: Light Congestion

In this section, we present the results in the light congestion scenario in two cases: the case with a tail drop queue, and the case with a random drop queue. This separates experiments according to the different assumptions used in the early studies in Chapter 4. The tail-drop queue experiment corresponds to the universal congestion signal assumption, whereas the random drop queue corresponds to the non-universal congestion signal assumption.

(a) Tail drop queue

The bandwidth share ratio between the two TCP-friendly flows is equal when the bottleneck router uses a tail drop queue. The TCP-friendly AIMD(α) flow's α parameter varies from 0.2 to 2.8 across 14 experiments. Adjusting the α parameter does not change the bandwidth share ratio. This result is shown in Figure 5.12 (a), which plots the ratio of achieved average throughputs over 1000 seconds.

The link utilization for these experiments, shown in Figure 5.13 (a), decreases slightly as the α parameter increases. The link utilization is not 100% for several reasons. First, retransmissions do not contribute to application throughput. Second, the bottleneck buffer occasionally runs out of packets due to the slowdown of both flows. These events occur in the experiments due to the congestion signals, which are shown in Figure 5.14 (a). As the congestion signals increase, both factors (retransmissions and buffer underflows) increase, and thus the link utilization degrades slightly. However, compared to the studies presented later in 5.3.3, where unfriendly AIMD congestion control flows are used, this link utilization degradation is small.

[24] This representation shows the events per time. Another popular way of represent the congestion experience is using per packet event possibility, which can be derived from the event per time combined with the throughput measurement.

Although the bandwidth share ratio is unchanged, the total number of congestion signals experienced by the two flows is not. The Figure 5.14 (a) shows the congestion signals experienced by each flow over the same time period. Clearly, as the α parameter of the TCP-friendly AIMD(α) flow increases, the congestion signals experienced by both flows increase. We believe this increment of congestion signals is due to the increment of flow aggressiveness, because a flow with a larger α parameter can fill the same amount of buffer faster than a flow with a smaller α.

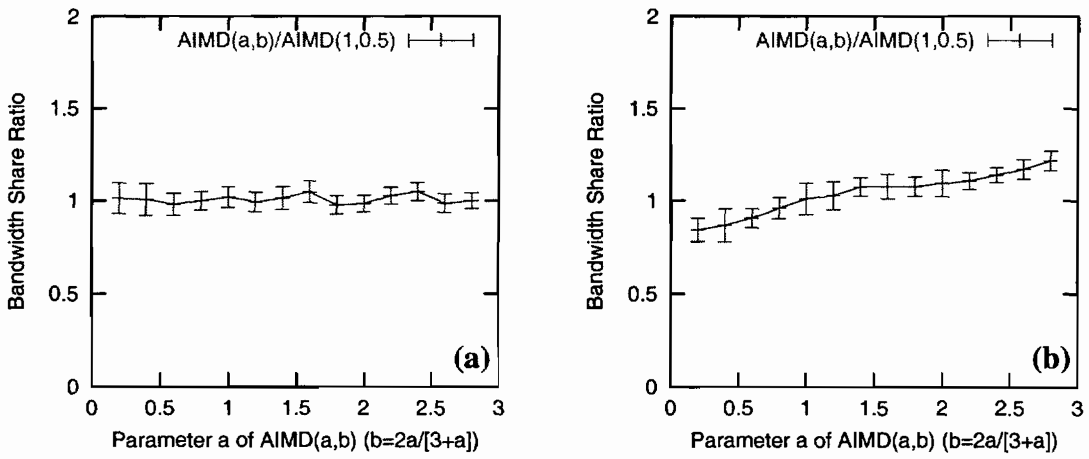


Figure 5.12: Bandwidth Share Ratio of TCP-friendly Flows during a Light Congestion.
(a) Tail-drop Queue, (b) Random-drop Queue

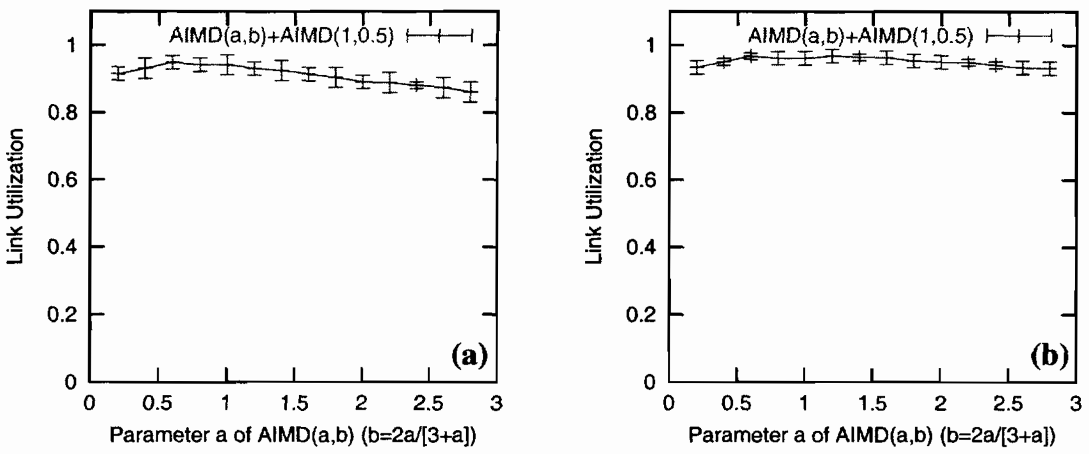


Figure 5.13: Link Utilization during a Light Congestion.
(a) Tail-drop Queue, (b) Random-drop Queue

We also measured the number of timeouts experienced by the two flows. Zero timeouts happen for these two flows during this experiment; therefore, no timeout figures are presented here. The timeout values under various network conditions (including this one) are plotted in Figure 5.19.

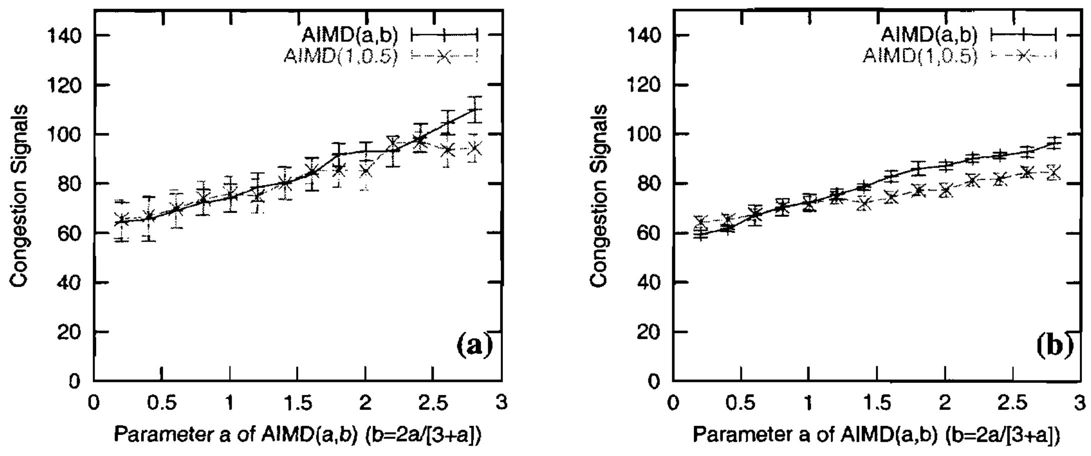


Figure 5.14: Congestion Signals Experienced by TCP-friendly Flows during a Light Congestion. (a) Tail-drop Queue, (b) Random-drop Queue

(b) Random Drop Queue (DRD + ECN)

The bandwidth share ratio between the two TCP-friendly flows is no longer equal when the bottleneck router uses a random drop queue. In our experiment, a derivative random drop (DRD) queue with explicit congestion notification (ECN) is enabled. This result is shown in the Figure 5.12 (b), which plots the ratio of the two flows' achieved average throughput over 1000 seconds. The flow that has a larger α parameter acquires slightly more bandwidth than the one with a smaller α parameter.

The link utilization in the random drop queue case, shown in Figure 5.13(b), is higher than the link utilization in the tail drop queue case, shown in Figure 5.13(a). This is the result of random early ECN marks by the DRD queue management, so that the two competing flows behave less synchronously and keep filling the bottleneck buffer without overflowing it.

The measurement of the number of congestion signals shows a counter-intuitive result. Intuitively, a random early drop queue with ECN sends early notifications back to a sender before the queue is filled up, thus it causes more back offs to the sender than a tail-drop queue does. The experiment results in Figure 5.14, however, show that the random drop queue case has fewer congestion signals than the case with a tail drop queue. The congestion window trace in the kernel indicates that the random drop case has most of its back offs spaced apart and typically only one of the two flows back offs for a congestion, whereas the tail-drop case has more contiguous back offs, and typically both flows back off for a congestion.

The other aspects of the measurement of congestion signals in the random drop queue case are similar to the tail-drop case. As in the case of tail-drop, the total number of congestion signals experienced by the two flows increases as the parameter α increases. The Figure 5.14 (b) shows the congestion signals experienced by each flow over the same time period. As the α parameter of the TCP-friendly AIMD(α) flow increases, the congestion signals experienced by both flows increase. No timeouts happen for these two flows during this experiment.

The bandwidth sharing ratio results are different between the cases with a tail-drop queue and with a random drop queue. The reason is that a tail-drop queue typically causes packet losses to the two flows together, i.e. tail-drop could be modeled using a universal congestion signal in this case. The random-drop queue has a higher chance of marking only one flow, and thus has more non-universal congestion signals. The TCP-friendly parameter relationship (2.3) is derived with the assumption of universal congestion signals, which is close to the result from a tail-drop queue. Thus, the TCP-friendly AIMD(α) flow is more “friendly” to the TCP flow in the tail-drop queue case than in the random-drop queue case. In the random drop queue case, a flow’s congestion experience is more related to its transient transmission rate. This relationship is similar to the assumption used in our Simulink simulations in section 4.1.3, and the result shows a similar impact of the AIMD parameters.

Scenario-II: Severe Congestion

In the case of severe congestion, two major factors, both congestion signals and timeouts, play important roles in determining the bandwidth share ratio. In our early simulations, we did not model the timeout events. Experiments in this section actually show that timeouts are no longer absent during severe congestion, but rather significant. This result indicates that further improvement to the model to include timeouts is desired, especially to study severely congested networks.

(a) Tail drop queue

The bandwidth share ratio is no longer fair in the tail drop queue case. Figure 5.15 (a) shows the average throughput ratio between a TCP-friendly AIMD(α) flow and a normal TCP flow, which diverges from the even-share. The flow that has a larger α parameter acquires slightly more bandwidth than the one with a smaller α.

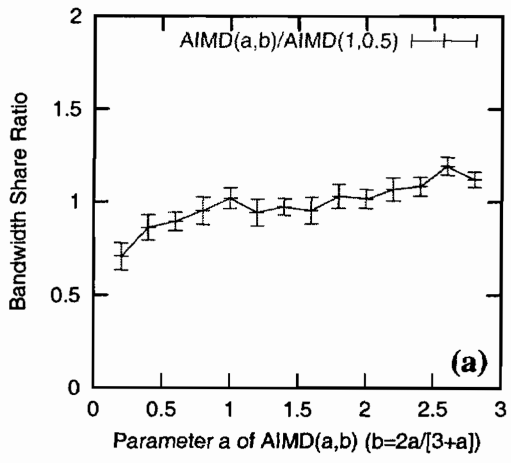
The link utilization, shown in Figure 5.16(a), clearly decreases as the α parameter increases. The reason for this utilization decrease can be found in Figure 5.17 (a) and 5.18 (a), in which congestion signals and timeouts are shown respectively. Both timeouts and congestion signals happen in the experiments, and the total of each of them increase as α increases. One more thing to notice is that, as the α parameters differ far from 1, the congestion experience (both congestion signals and timeouts) of the AIMD(α) flow differs far from the TCP flow's experience. This indicates that as flows' parameters differ far away from each other, so do their congestion experiences.

The experiment result indicates that timeouts are significant. Figure 5.17 (a) and Figure 5.18 (a) present the congestion signals and the timeouts respectively. Figure 5.17(a) indicates that the flow with a larger α tends to get more congestion signals than the one with a smaller α. In the α = 2.8 case, the flow AIMD(2.8) gets about 20% more congestion signals than the flow AIMD(1.0, 0.5). If there were no timeout difference, the flow AIMD(2.8) would get less bandwidth than AIMD(1.0, 0.5). On the contrary, the flow AIMD(2.8) gets about 10% more bandwidth than AIMD(1.0, 0.5). We believe the change is caused by the behavior of timeouts. As shown in Figure 5.17(a), the flow AIMD(1.0, 0.5) gets about twice the number of timeouts as the flow AIMD(2.8). The amount of timeouts not only balances the difference in congestion signals, but also causes AIMD (1,0.5) to get more bandwidth. Therefore, we conclude that timeouts play an important role in determining the bandwidth share ratio. A small number of timeouts can significantly reduce a flow's ability to compete for bandwidth. One possible reason for the more aggressive flow getting fewer timeouts is that it sends more packets in bursts, and therefore is more likely to get duplicated ACKs rather than waiting for timeouts for a given lost packet.

(b) Random Drop Queue (DRD + ECN)

In the severe congestion scenario, the experiment result with a random drop queue exhibits a similar bandwidth share ratio as the case with a tail drop queue. The bandwidth share ratio, the link utilization, the congestion signals, and the experienced timeouts are shown in Figure 5.15 (b), 5.16 (b), 5.17 (b) and 5.18 (b) respectively.

Similar to the tail-drop queue case, timeouts play a significant role in determining the bandwidth share ratios. Most of the aspects shown in this experiment are very similar to the case with tail-drop queue. The reason is that the queue in the severe congestion scenario is small compared to the path bandwidth-delay product, and the early congestion notifications are actually not early enough to prevent the queue from filling up and dropping packets.



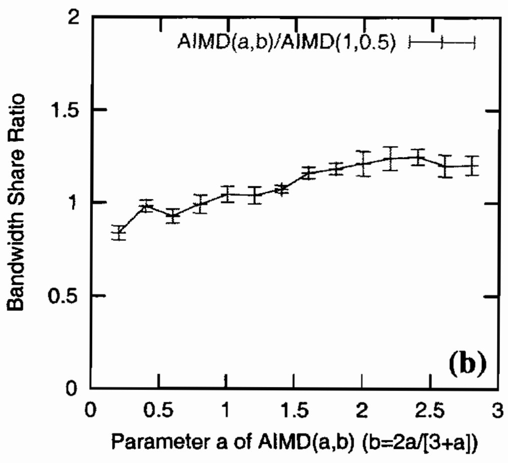
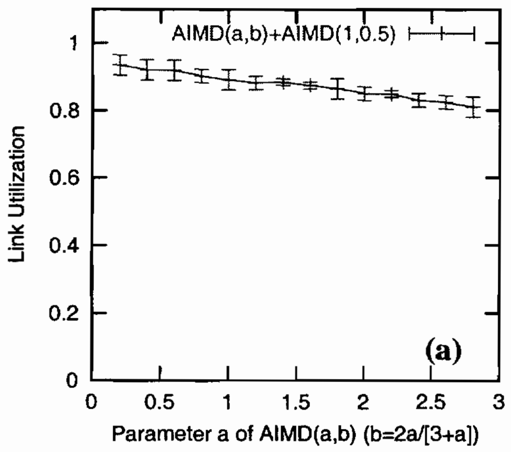


Figure 5.15: Bandwidth Share Ratio of TCP-friendly Flows during a Severe Congestion. (a) Tail-drop Queue, (b) Random-drop Queue



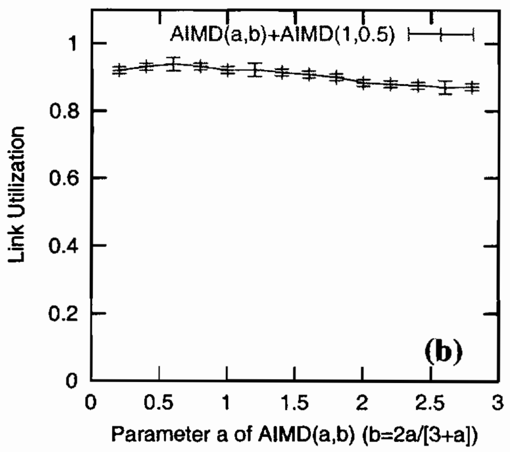
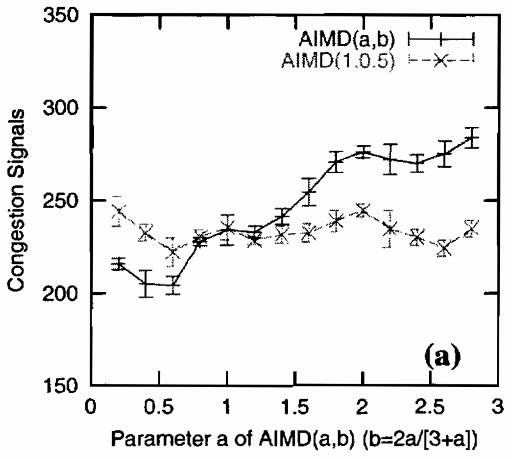


Figure 5.16: Link Utilizations of TCP-friendly Flows during a Severe Congestion. (a)Tail-drop Queue, (b) Random-drop Queue



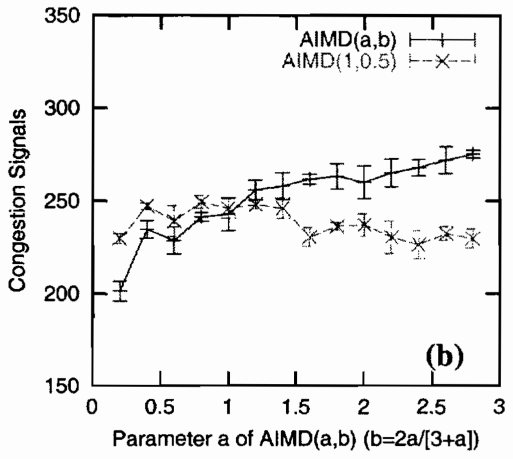


Figure 5.17: Congestion Signals Experienced by TCP-friendly Flows during a Severe Congestion. (a) Tail-drop Queue, (b) Random-drop Queue

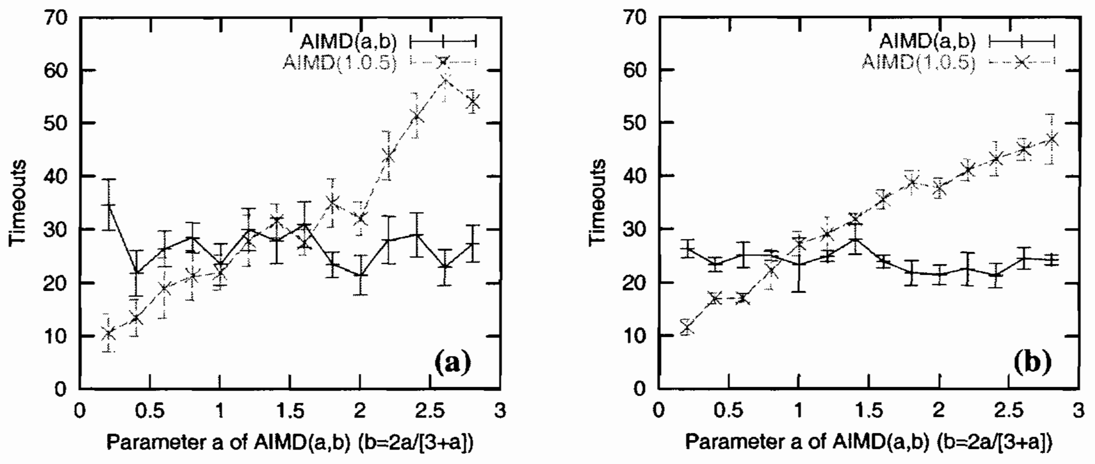
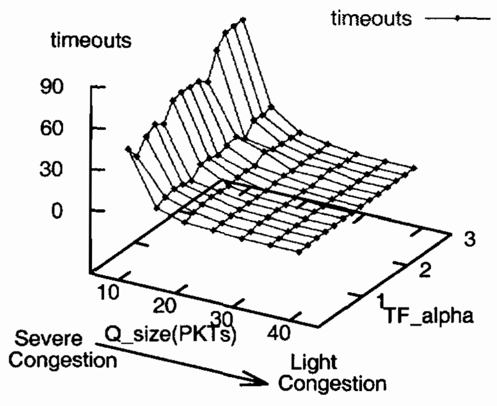


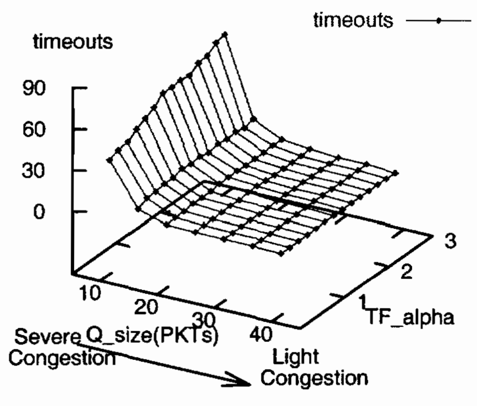
Figure 5.18 Timeouts Experienced by TCP-friendly Flows during a Severe Congestion. (a) Tail-drop Queue, (b) Random-drop Queue

To illustrate the relationship between the congestion degree (light to severe) and the timeouts, we plot the number of timeouts versus a few experiment parameters from severe congestions to light congestions. The degree of congestion is controlled by the bottleneck queuing capacity. In our experiment, the bottleneck queue size varies from 10 packets to 40 packets. The degree of congestion changes from severe to light as the queue size changes.

The resulting timeouts are shown in Figure 5.19 with (a) a tail drop queue case and (b) a random drop queue case. In both cases, we can see that timeouts rarely happen when congestion is not severe, as in Scenario-I. The number of timeouts increases dramatically as the bottleneck buffer becomes smaller. Another point is that more timeouts happen when the TCP-friendly AIMD (α) uses a large α parameter. This result indicates that increasing the α parameter of an AIMD flow in the severe congestion case could cause more congestion to the whole system, even if the AIMD parameters are adjusted in a TCP-friendly way.



(a)



(b)

Figure 5.19: Timeouts versus congestion levels.
(a) Tail-drop Queue, (b) Random-drop Queue

5.3.2.2 Many Flows

We extend the earlier experiments with only 2 flows to an experiment with 14 flows. The 14 flows are all different TCP-friendly AIMD(α) flows with their α parameters evenly distributed between 0.2 and 2.8 with an interval of 0.2.

Due to time limitations, we only study the case with light congestion. Since the number of competing flows increases from 2 to 14, the bottleneck queuing capacity for light congestion is adjusted to 280 packets accordingly. We divide the experiments into the cases of using a tail-drop queue and a random-drop queue. The bandwidth share ratios of all competing flows are presented in Figure 5.20 (a) and (b) for the two queue management cases. The result ratio is measured by comparing the throughput of a flow to the average throughput of the AIMD(1,0.5) flow over the 20 rounds of experiments.

The result of many flows is essentially the same as the result of 2 flows. The bandwidth share ratio is about the same for all 14 flows in the case with a tail drop queue. The bandwidth share ratio of the same 14 flows is clearly different in the case of a random drop queue.

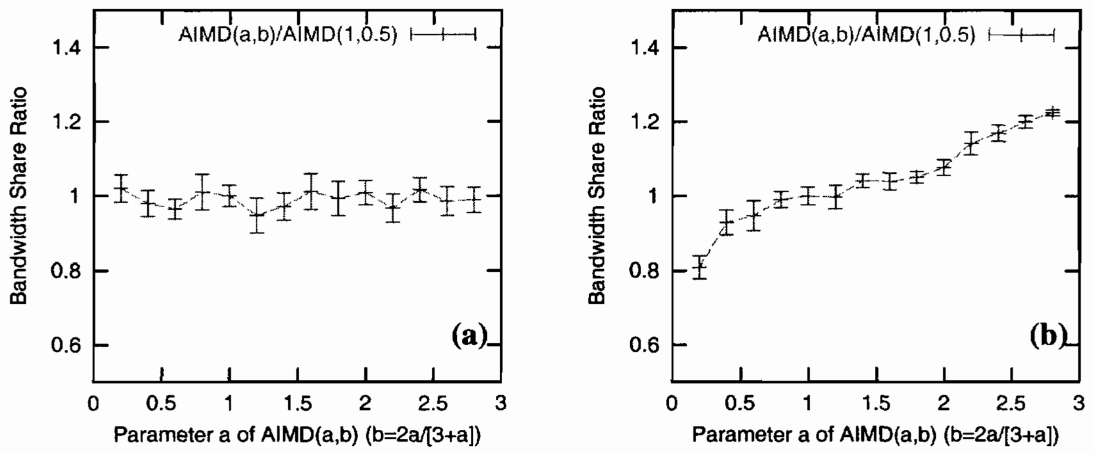


Figure 5.20: Share Ratio of 14 TCP-Friendly Flows during a Light Congestion.
(a) Tail-drop Queue, (b) Random-drop Queue

5.3.2.3 Discussion

The results in this subsection show that the "TCP-friendly" AIMD flows that follow the Equation (3.1) do not necessarily share bandwidth equally with normal TCP flows. There are two reasons for this unfairness. One is the difference in congestion experiences, and the other is timeouts. The former causes unfairness because the Equation (3.1) is derived based on the assumption of universal congestion signals, and differences on congestion experience diverge away from this assumption as active queue management is deployed in routers. The latter causes unfairness because timeouts are not taken into account in (3.1).

Adjusting the AIMD parameter of an AIMD(α,β) flow to achieve an even share ratio to a TCP flow is not easy. The impact of AIMD parameters on bandwidth sharing is not static but changes based on running scenarios, such as the degree of congestion and the queue management scheme used. Typically the congestion control protocol does not know the conditions beforehand, and the running scenario may change from time to time. Therefore, it is difficult to design an AIMD protocol that shares bandwidth evenly in all conditions.

Indeed, it might not be important for all flows to achieve a perfectly even share with each other. As shown in the section 5.2.1, the share ratio is not perfect even among identical TCP flows. Therefore, we move our focus onto adjusting the AIMD parameters to achieve other share ratios, rather than trying to fine tune the AIMD parameter relationship for "perfect fairness".

5.3.3 Share among "Un-Friendly" AIMD Flows

The last section has shown that the TCP-friendly AIMD flows do not always get equal bandwidth with normal TCP flows, especially under the situation of a random drop queue or under severe congestion. This section discusses the cases of the bandwidth share ratio of AIMD flows using "non-TCP-friendly" parameters, to check how to achieve a better fairness condition or other share ratios in various scenarios.

The way we conduct our study is to fix one of the AIMD parameters, and check the impact of the other AIMD parameter on the bandwidth share ratio. We first look at the case with a fixed β parameter, and check the impact of the α parameter on the share ratio. The results are presented in section 5.3.3.1. Then in section 5.3.3.2, we look at the case with a fixed α parameter, and check the impact of the β parameter.

5.3.3.1 Impact of α parameter

For simplicity, we choose only one fixed $\beta = 0.5$ in this experiment. The experiments are again split into two scenarios: light congestion and severe congestion.

Scenario-I: Light Congestion

In each scenario, we have two cases based on the bottleneck queue management schemes: tail-drop queue or random drop queue.

(a) Tail-drop Queue

Figure 5.21(a) shows the share ratio between AIMD(α,0.5) and AIMD(1,0.5) in the light congestion scenario. The result indicates that increasing an AIMD flow's α parameter does increase its aggressiveness, thus gaining more bandwidth than the flows with a smaller α parameter.

In our theoretical study of the behavior of AIMD flows, we derived the average throughput of an AIMD flow, which is shown in (A2.5). If we assume two AIMD flows have the same congestion signals, we can get the ratio of the throughput of two flows from this throughput equation:

$$\frac{R_{AIMD(\alpha_1,\beta_1)}}{R_{AIMD(\alpha_2,\beta_2)}} = \sqrt{\frac{\alpha_1}{\alpha_2} \times \frac{(2-\beta_1)/\beta_1}{(2-\beta_2)/\beta_2}} \tag{5.3}$$

With a fixed β parameter, the theoretic result indicates that the share ratio between AIMD(α, 0.5) and AIMD(1,0.5) should be proportional to the square root of the α parameter if the loss rate is the same. The share ratio in the light congestion scenario, shown in Figure 5.21 (a), matches this theoretic prediction well when α is small. To make a comparison between the bandwidth share ratio presented in this experiment to the one of TCP-friendly AIMD flows, we enlarge the part of Figure 5.21 where the range is 0<α<3, and present it in Figure 5.22. Figure 5.22(a) shows that the bandwidth share ratio matches the prediction well in the light congestion scenario with the tail drop queue.

As shown in Figure 5.21(a), the bandwidth share ratio diverges from the prediction when α becomes larger than 3. The ratio even decreases when the α parameter is more than 10. The reason is that congestion experiences are no longer the same for the AIMD(1, 0.5) and AIMD(α, 0.5) when α is large. With a large α, a group of packets (here more than 10) are sent to the network immediately every time the AIMD flow expands its window. This large burst can potentially cause many packet losses and reduces the effective bandwidth obtained by the AIMD(α, 0.5) flow. Thus the bandwidth share ratio can not keep up with the predicted value.

The result of the link utilization in this case is presented in Figure 5.23(a). The link utilization decreases significantly as α gets larger. The reason for the utilization decrement is also because of the packet losses caused by the burst.

From the experiment result, we can also see that the total number of congestion signals, shown in Figure 5.24(a), also increases with the increment of the α parameter and the aggressiveness.

The results indicate that additional mechanisms, such as self-clocking and pacing, are required for deploying AIMD(α,β) flows that have a large α to smooth out the big burst caused by large α parameters. Without any additional mechanisms to smooth out the burst, it is better not to use them to produce different bandwidth share ratios, because they cause high congestion signal rates and reduce link utilization too much.

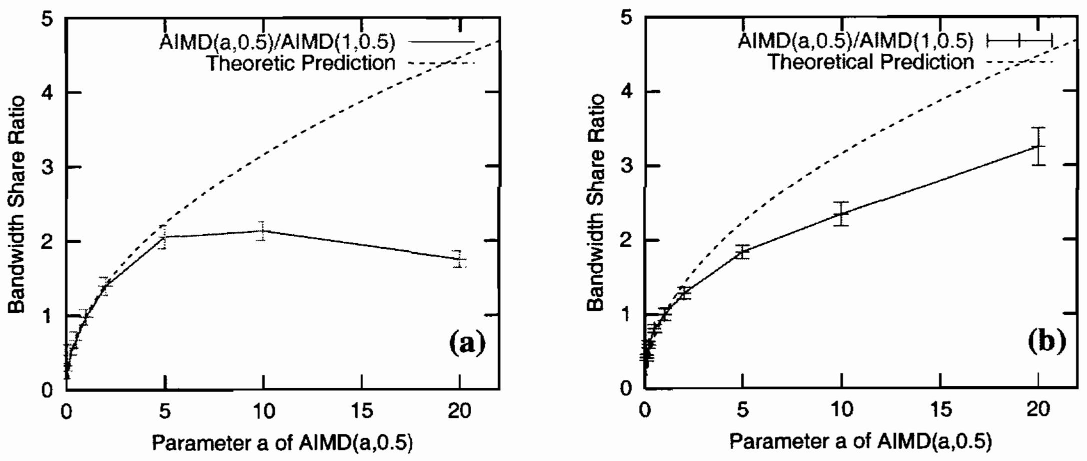


Figure 5.21: Share Ratio Impact of α Parameter. (β = 0.5. Light Congestion.)
(a) Tail-drop Queue, (b) Random-drop Queue

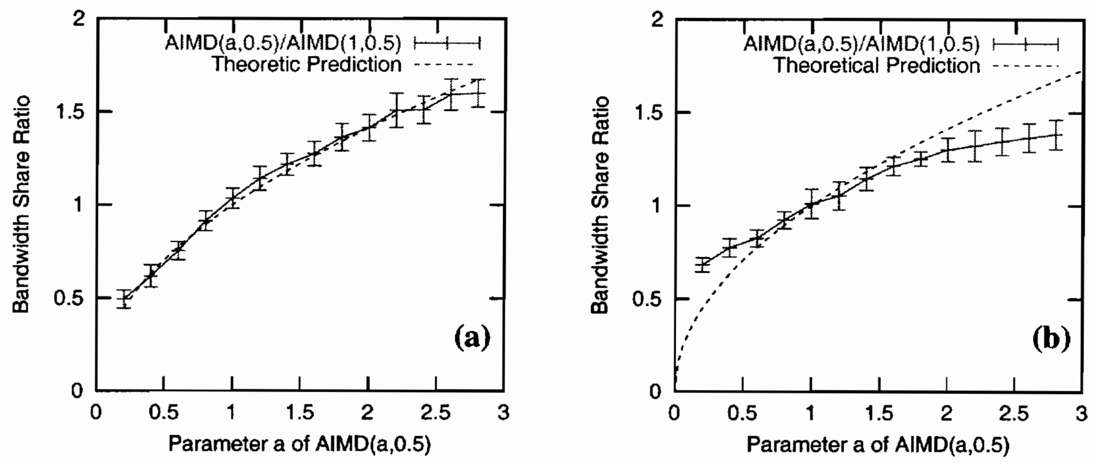


Figure 5.22: Share Ratio Impact of small α Parameter. (β = 0.5. Light Congestion)
(a) Tail-drop Queue, (b) Random-drop Queue

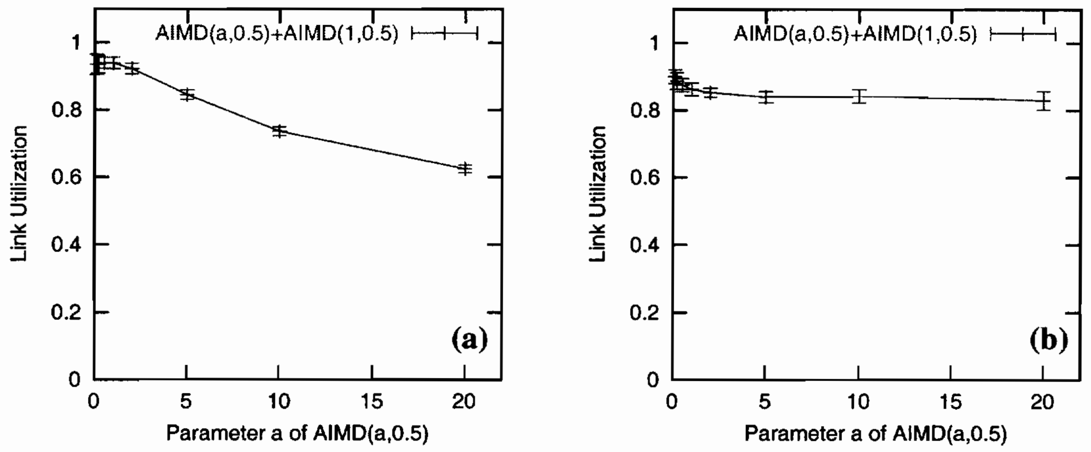


Figure 5.23: Utilization Impact of α Parameter. (β = 0.5. Light Congestion.)
(a) Tail-drop Queue, (b) Random-drop Queue

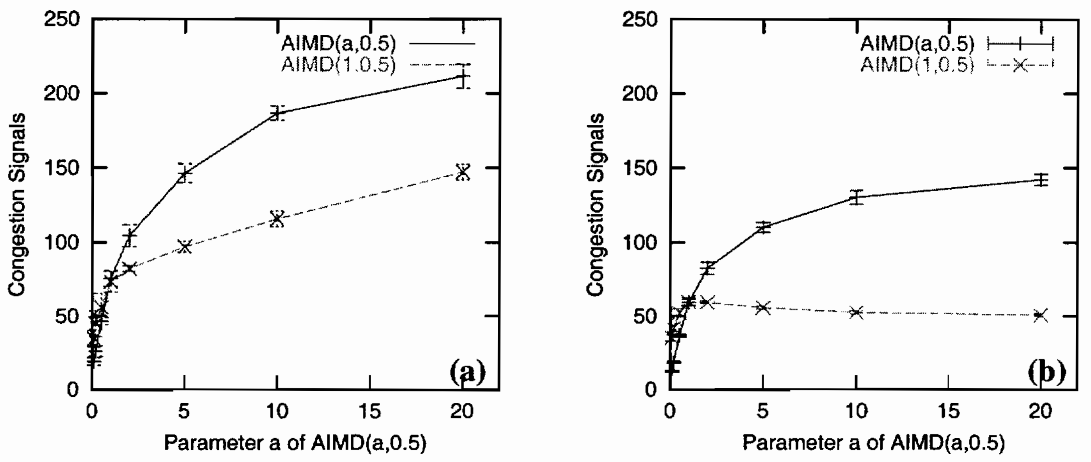


Figure 5.24: Congestion Signal Impact of α Parameter. (β = 0.5. Light Congestion)
(a) Tail-drop Queue, (b) Random-drop Queue

(b) Random Drop Queue (DRD + ECN)

The result of light congestion with a random drop queue is presented in Figure 5.21(b) ~ 5.24(b). The general trend of bandwidth share ratio, the congestion signals, and the utilities in this case is similar to the one in the tail-drop queue case. The bandwidth share ratio increases as the α parameter increases, but the number of congestion signals increases too.

Compared to the tail-drop queue case, the bandwidth share ratio in the random drop queue case is less close to the prediction when α is small ($0<\alpha<3$), as shown in Figure 5.22 (b). As α becomes larger in Figure 5.21(b), the bandwidth share ratio diverges away from the prediction but not as significantly as in the tail-drop queue case. The reason is that a random drop queue keeps the average queue fill-level low, so that it has a better tolerance for the burst than the tail drop queue case.

The result in this experiment indicates that a higher bandwidth share ratio can be achieved in the random drop queue case than in the tail drop case. In the random drop case, the AIMD(α, 0.5)/AIMD(1,0.5) ratio is up to 3 and possible higher with a large α, while the range in the tail-drop queue case the ratio is up to 2 and is likely to have a smaller than 2 ratio with a large α.

Scenario-II: Severe Congestion

This section briefly explains the bandwidth share experiments for non-TCP-friendly congestion control protocols under severe congestion.

(a) Tail-drop Queue

The results of the tail-drop queue case are shown in Figure 5.25(a) ~ 5.28(a). The bandwidth share ratio still matches well with the predication when α is small ($0<\alpha<3$).

Once α gets larger, the share ratio of AIMD(α,0.5)/AIMD(1,0.5), shown in Figure 5.25(a), and the link utilization, shown in Figure 5.26(a), both decrease significantly.

Similar to the light congestion scenario, the total number of congestion signals, shown in Figure 5.27(a), increases as α increases. Differing from the light congestion scenario, the number of congestion signals of flow AIMD(1,0.5) decreases when α is large. The reason can be found in Figure 5.28(a), which shows the number of timeouts. When α is large (α>3), the number of timeouts is significantly different for the two flows, and AIMD(1, 0.5) gets many more timeouts than AIMD(α, 0.5).

(b) Random Drop Queue (DRD + ECN)

The results of the random drop queue case are presented in Figure 5.25(b) ~ 5.28(b). Similar to the tail-drop queue case, the bandwidth share ratio, shown in Figure 5.25(b), diverges from the prediction when α becomes larger. The link utilization is low, and the total number of congestion signals for both flows increase when α becomes larger. The bandwidth share ratio with a random drop queue is better than the case with a tail drop queue in the sense that the share ratio does not decrease when α becomes larger.

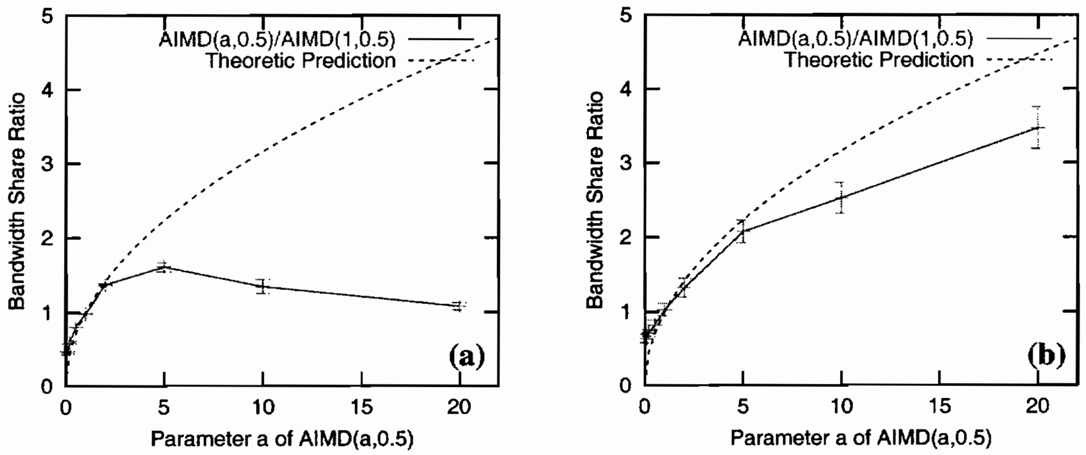
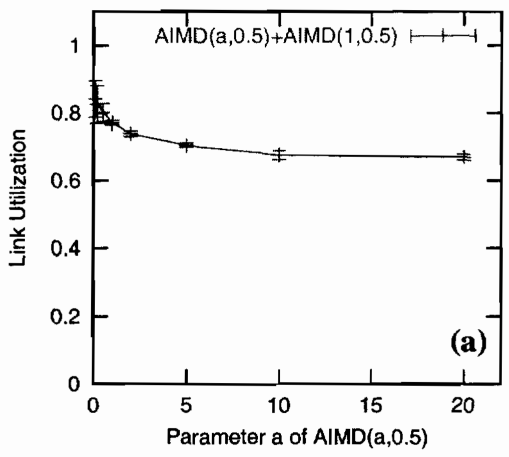


Figure 5.25: Share Ratio Impact of α Parameter. (β = 0.5. Severe Congestion.)
(a) Tail-drop Queue, (b) Random-drop Queue



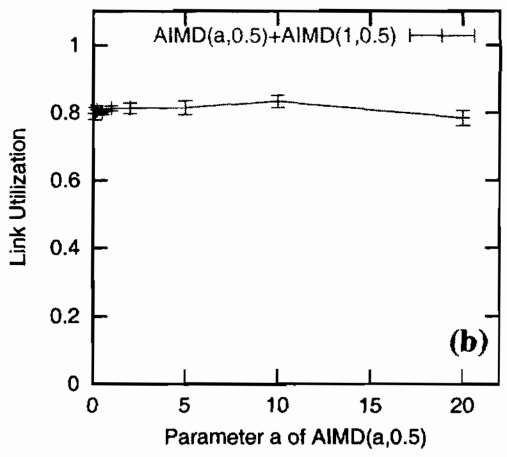
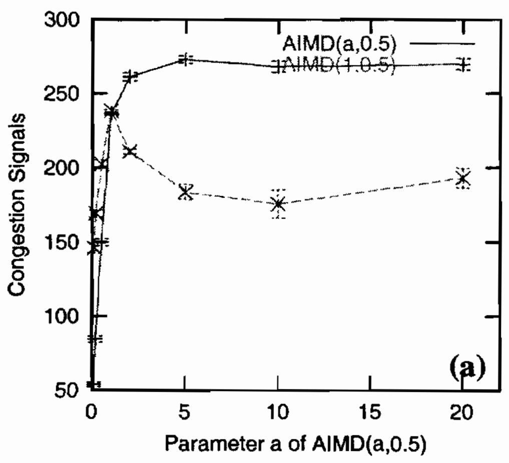


Figure 5.26: Utilization Impact of α Parameter. (β = 0.5. Severe Congestion.)
(a) Tail-drop Queue, (b) Random-drop Queue



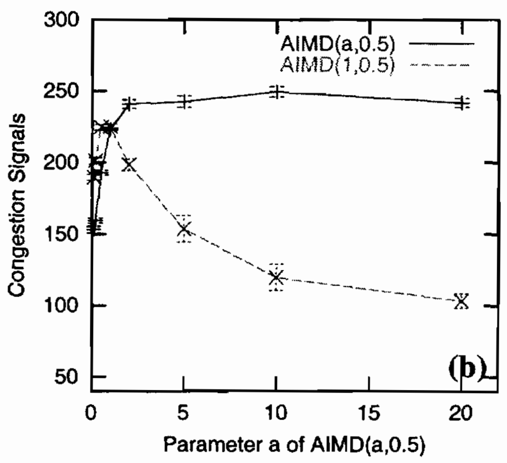
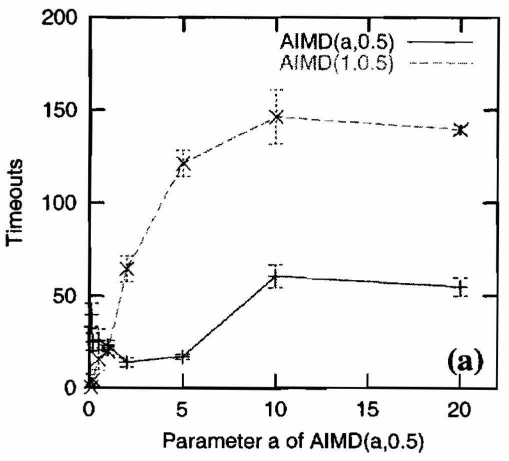


Figure 5.27: Congestion Signal Impact of α Parameter. (β = 0.5. Severe Congestion.)
(a) Tail-drop Queue, (b) Random-drop Queue



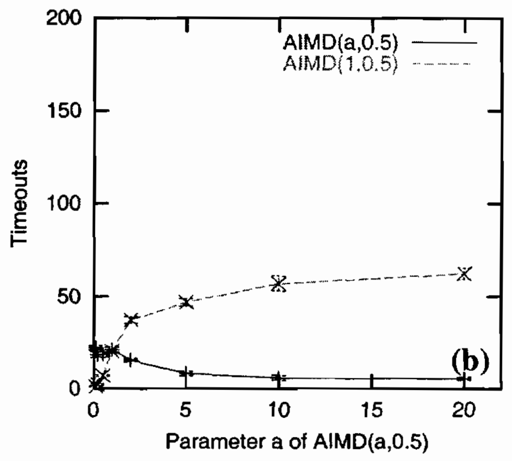


Figure 5.28: Timeout Impact of α Parameter. (β = 0.5. Severe Congestion)
(a) Tail-drop Queue, (b) Random-drop Queue

5.3.3.2 Impact of β parameter

In this experiment, we fix the α parameter, and vary the β parameter to check its impact. The range of valid β parameters is between 0 and 1. For simplicity, we choose a fixed α =1, and choose a group of AIMD(1, β) flows with the β parameter ranging from 0.1 to 0.9 with an interval of 0.1 between each flow. The experiment is to compare the bandwidth ratio between AIMD(1,β) with AIMD(1,0.5) when they are competing for the same bottleneck. The experiment is divided into the light and severe congestion scenario as before.

Scenario-I: Light Congestion

(a) Tail-drop Queue

As shown in the Figure 5.29 (a), the AIMD(1, β) gets less bandwidth as the β parameter increases from 0 to 1. According to (5.3), the bandwidth share ratio between AIMD(1, β) and AIMD(1,0.5) should be proportional to $\sqrt{(2-\beta)/3\beta}$ under the assumption of universal congestion signals. The result shown in Figure 5.29 (a) shows that the measured ratio matches the theoretical prediction well in the tail drop queue case. A congestion signal difference between the two flows can be seen in Figure 5.31(a) when β is small. The link utilization, shown in Figure 5.30(a), is high and increases slightly when β varies from 0 to 1.

From this result, we know that controlling the β parameter can control the share ratio between AIMD(1, β) and AIMD(1, 0.5) in a small range, from 0.5 to 2 times. For the case of a tail drop queue, we can predict the share ratio using (5.3).

(b) Random Drop Queue (DRD + ECN)

The bandwidth share ratio between AIMD(1, β) and AIMD(1, 0.5), shown in Figure 5.32(b), does not match the predicted ratio in the tail-drop queue case. The range of the achieved share ratio is from 0.8 to 1.5, and is smaller than the tail-drop queue case. The number of congestion signals and timeouts in the random drop queue is smaller than the tail-drop case. Other than that, the results are similar to those of the tail-drop queue case.

From this result, we can see that adjusting β can change the bandwidth share ratio, but the achievable ratio range is very limited.

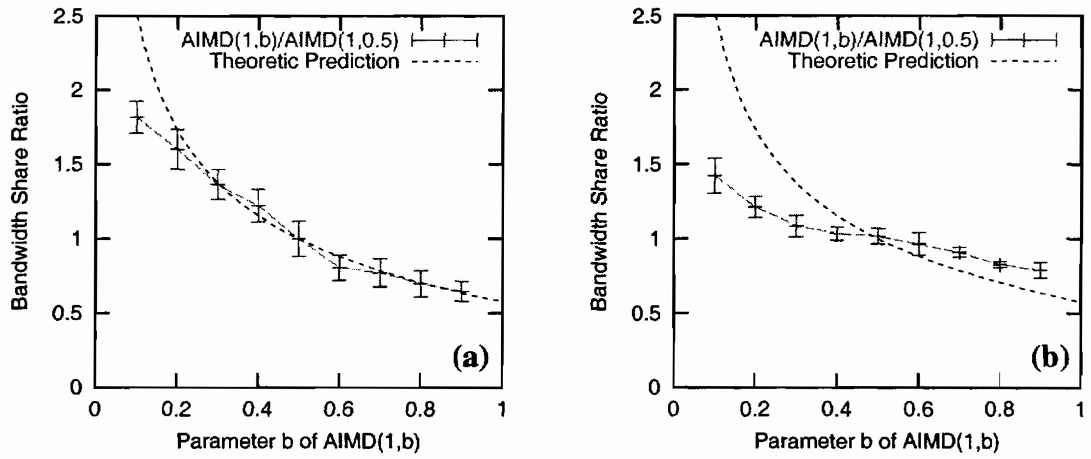


Figure 5.29: Share Ratio Impact of the β Parameter. (α = 1. Light Congestion)
(a) Tail-drop Queue, (b) Random-drop Queue

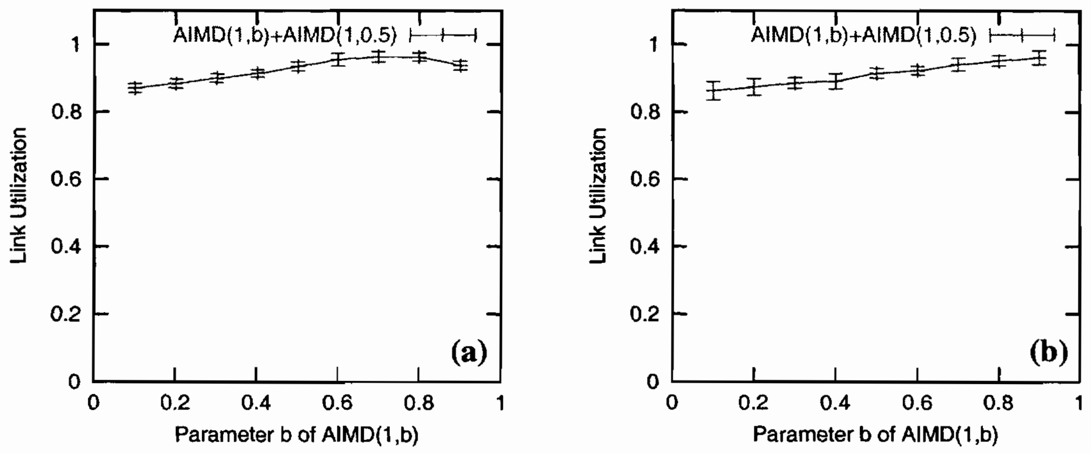


Figure 5.30: Utilization Impact of the β Parameter. (α = 1. Light Congestion.)
(a) Tail-drop Queue, (b) Random-drop Queue

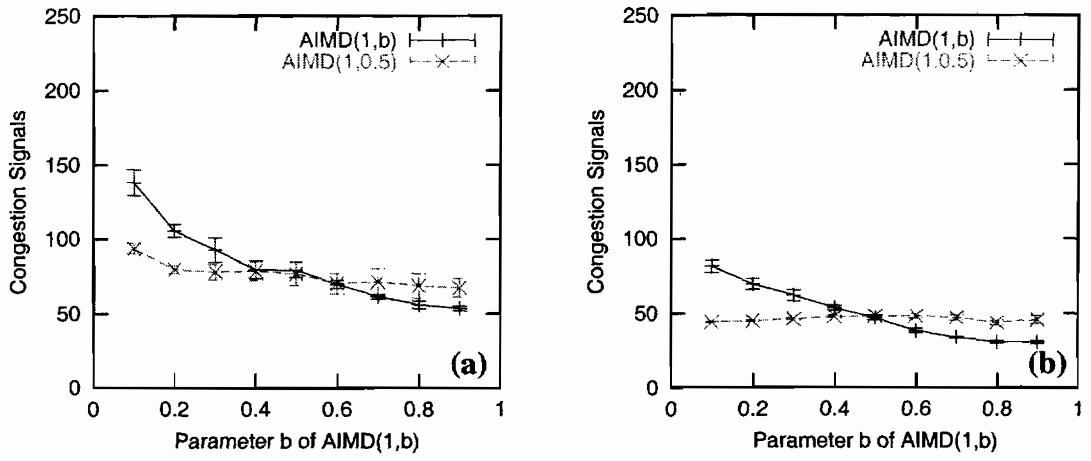


Figure 5.31: Congestion Signal Impact of the β Parameter. (α = 1. Light Congestion.)
(a) Tail-drop Queue, (b) Random-drop Queue

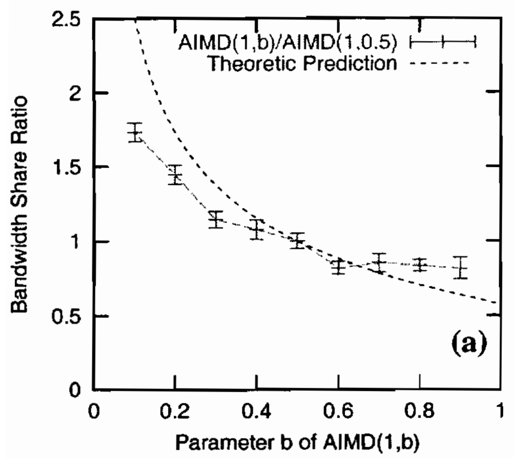
Scenario-II: Severe Congestion

(a) Tail Drop Queue

In the severe congestion scenario, the bandwidth share ratio between AIMD(1,β) and AIMD(1,0.5) is still close to the theoretical prediction when using a tail drop queue. It diverges more compared to the light congestion scenario because the two flows' congestion signals and timeouts, shown in Figure 5.34 (a) and 5.35 (a), differ more in this scenario. Other than that, the result is similar to the light congestion scenario.

(b) Random Drop Queue

Compared to the case with a tail drop queue or the light congestion scenario, the bandwidth share ratio between AIMD(1,β) and AIMD(1,0.5) is less close to the theoretical prediction. The difference is large when β is far from 0.5. The achieved share ratio between AIMD(1,β) and AIMD(1,0.5) is in a small range between 1 to 1.5. Other aspects, such as the link utilization, the congestion signals, and the timeouts are similar to the tail-drop queue case, except the number of congestion signals and timeouts are slightly smaller than the tail drop case.



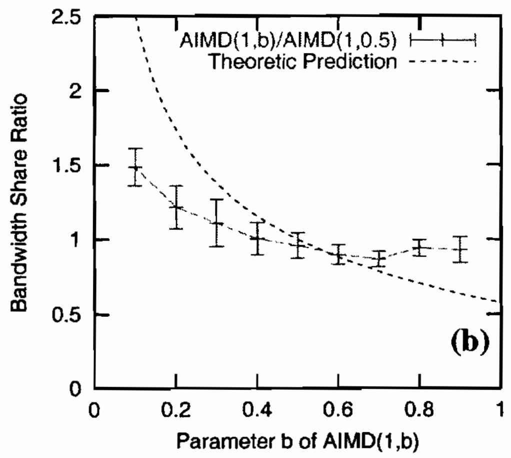
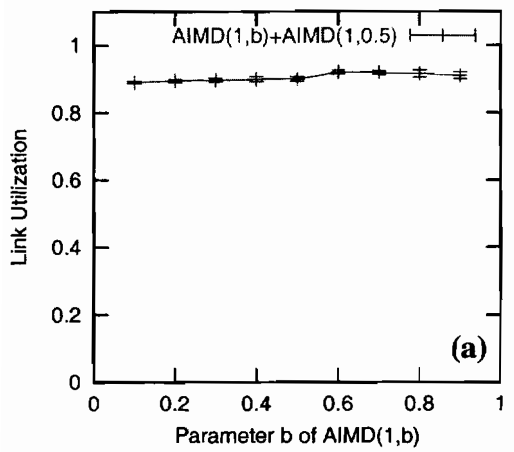


Figure 5.32: Share Ratio Impact of the β Parameter. (α = 1. Severe Congestion.)
(a) Tail-drop Queue, (b) Random-drop Queue



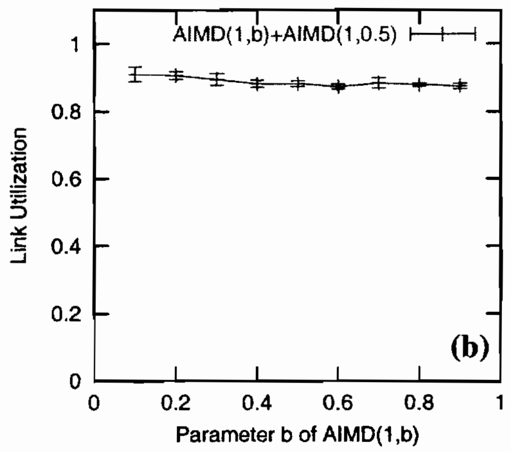
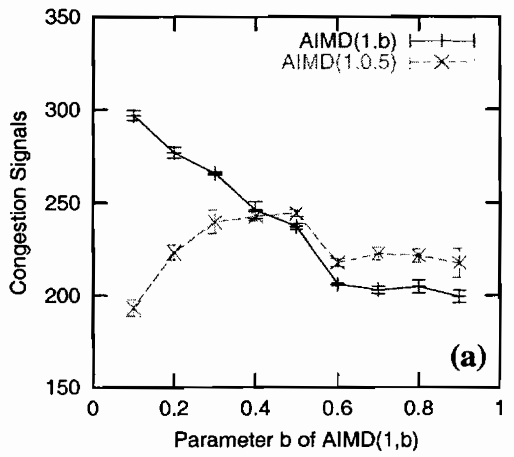


Figure 5.33: Utilization Impact of the β Parameter. (α = 1. Severe Congestion.)
(a) Tail-drop Queue, (b) Random-drop Queue



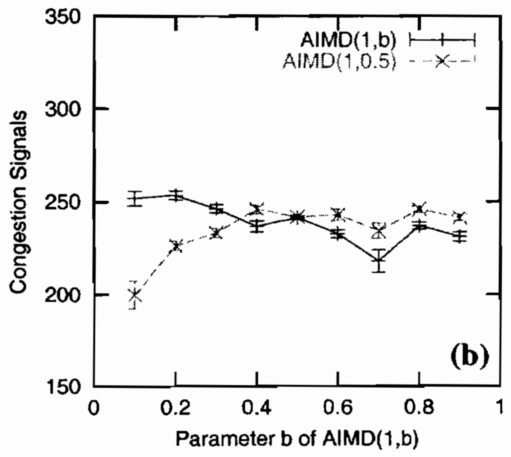


Figure 5.34: Congestion Signal Impact of the β Parameter. (α = 1. Severe Congestion.)
(a) Tail-drop Queue, (b) Random-drop Queue

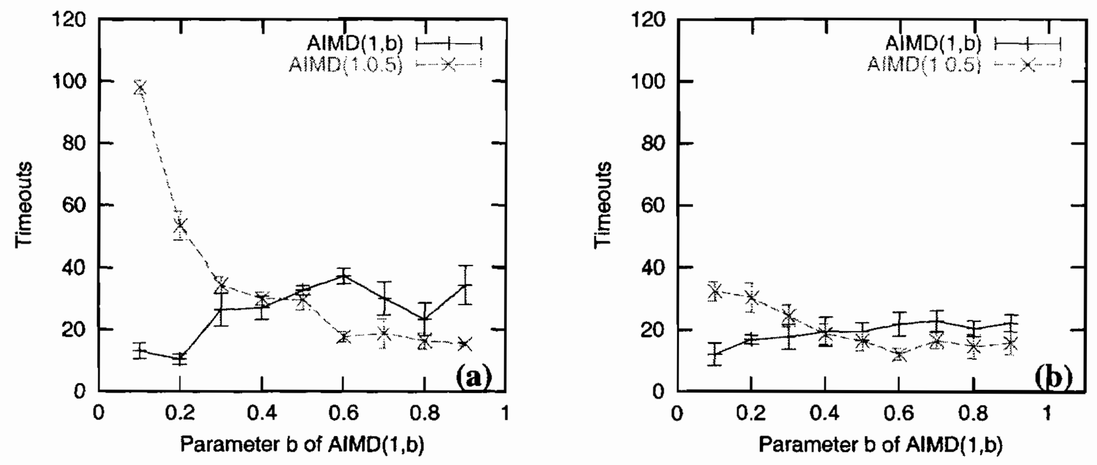


Figure 5.35: Timeout Impact of the β Parameter. (α = 1. Severe Congestion.)
(a) Tail-drop Queue, (b) Random-drop Queue

With the experiment results of the sharing ratio between "unfriendly" AIMD flows, we conclude the following points. In the tail drop case, the bandwidth share ratio between unfriendly AIMD(α,β) flows and normal TCP flows matches the predicted behavior when the parameters of the AIMD(α,β) flows are close to (1, 0.5). For a range of 0<α<3 and 0<β<1, the bandwidth share ratio is very close to the theoretic prediction. For a very large α, or a very small β, the changes of α and β become less effective for changing the share ratio. The reason for this reduced effectiveness and the divergence form the theoretical prediction is because congestion experiences are different between a TCP and an AIMD(α,β) flow with a large α or a small β. For the random drop queue case, the bandwidth share ratio does not exactly match to the predicted ratio, but tuning the α and β parameters can still achieve various share ratios, if the ratios are not too large.

5.3.4 Other Fairness Ratio

In this section, we address the issue of how to create some share ratios other than the TCP-style fairness. We phrase the question as, how to adjust the AIMD parameters to achieve a N-to-one share ratio between flows?

We can use the result of the last section to answer this question when all flows have the same RTT and the bottleneck uses a tail-drop queue. Compared to AIMD(1,0.5), we know that a ratio of 0.25 to 3 can be achieved by tuning the α or β parameter without adding many congestion signals and timeouts. A wider range of share ratio might be achieved by tuning both parameters.

While saying that different sharing ratios can be achieved, it is worth mentioning two constraints. First, the achieved share ratio between two flows is only effective when the two flows encounter congestions. If there were no congestion, then the two flows would have a 1:1 throughput ratio. Second, the sharing ratio that can be achieved by tuning the parameter is coarse and is dependent on running scenarios. Our study of the TCP-friendly AIMD flows indicates that the queue management scheme and the bottleneck buffer spaces change the sharing ratio.

With the awareness of these constraints, we go ahead to tune the α and β parameters for different share ratios. As an example of achieving different ratios, we adjust the parameters to achieve a uniform fairness that shares bandwidth fairly regardless of the RTT. This result is shown in the next section.

5.3.4.1 Different RTT

In this section, we address the issue of the effect of different round-trip-times on bandwidth share, and how to compensate the RTT difference using the adaptive AIMD congestion control.

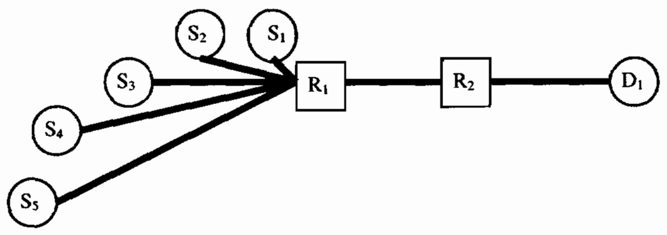


Figure 5.36: Network Topology

First, we perform experiments to check the effect of RTT on identical AIMD flows. We reconfigure our experiment topology, so that we can create AIMD flows with different

RTTs. Figure 5.36 shows the new topology, in which flows to D_1 have different RTTs depending on which source they are from (S_1 to S_5).

In this network topology, we set the bottleneck link rate to 3MB/S, with a bottleneck queue size of 50 packets. To check the impact of RTT on bandwidth share ratio, we run 2 identical flows with different RTTs. The RTTs we choose in our experiments vary from 25ms to 100ms.

We first conduct the bandwidth share measurement for AIMD(1, 0.5) for different RTTs. We use the throughput of the flow with RTT=50ms as the reference, and present the normalized bandwidth share ratios for server RTTs in Figure 5.37 (a) and (b), for the cases of tail drop queue and random drop queue respectively. Conventionally, TCP throughput equations, such as (2.1), indicate that a TCP flow's throughput is inversely proportional to RTT, but these equations are based on the assumption that congestion signals are independent of the RTT. In this experiment, the flows that we monitored are the only contributors to the congestion signals, and congestion signals are not static but change when RTTs change. According to the measured result in Figure 5.37, the bandwidth shares obtained by flows are inversely proportional to the square of RTT.

To show that we can compensate the bandwidth share bias of different RTTs by adjusting the AIMD parameters, we redo the experiments with flows that adjust their α parameters according to their RTT. We conduct experiments that only tune the α parameters due to time limitations. Experiments that tune the β, or both α and β parameters can be conducted in a similar way.

We conduct two sets of experiments for tuning the α parameter. One set uses the policy of tuning α to be proportional to RTT; the other one uses the policy of tuning α to be proportional to RTT^2. If the congestion signals are independent of the flow's RTT, then the first policy should compensate the RTT difference, according to (2.1). Otherwise, if the congestion signals are closely dependent on the AIMD flows' parameters, the second policy should compensate the RTT difference effectively according to the experiments in section 5.3.3.

We set the RTT of the reference flow AIMD(1, 0.5) to 50ms. Correspondingly, other flows' AIMD parameters are set to AIMD(RTT/50, 0.5) and AIMD([RTT/50]^2, 0.5) for the two policies respectively.

Our results indicates that the second RTT compensation policy (adjusting α parameters quadratic to RTT) can achieve an uniform fairness when α is in the range of $0<\alpha<3$, but is less effective when α is large ($\alpha>3$). The experiment results for both policies are presented in Figure 5.37 for the cases of (a) tail drop queue and (b) random drop queue.

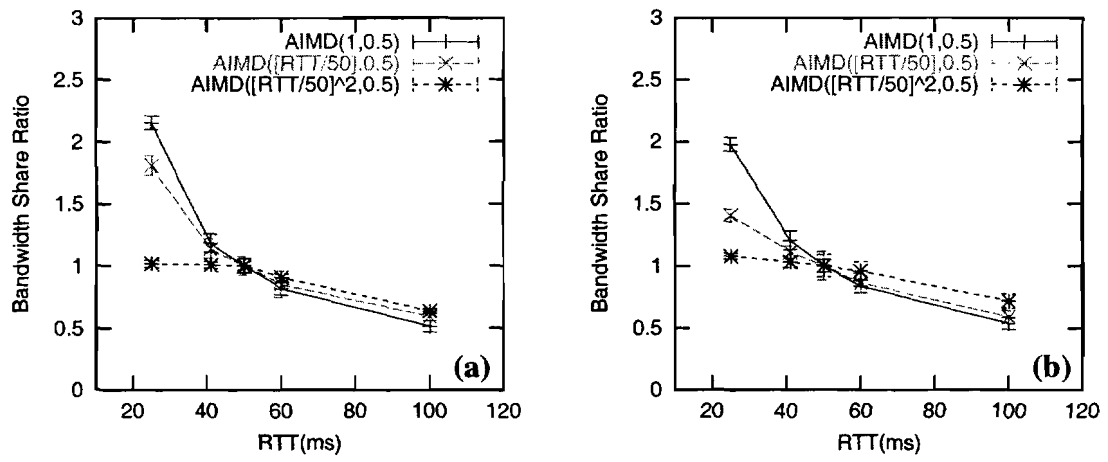


Figure 5.37: Adjusting AIMD parameters for RTT compensations
(a) Tail-drop Queue, (b) Random-drop Queue.

The first policy's compensation(adjusting α proportional to RTT) is less effective in the tail drop case than the random drop case. Figure 5.37(a) shows that short RTT flows still get significantly higher bandwidth shares than long RTT flows in the tail drop queue case. The short RTT flows' advantage on bandwidth ratio is reduced in the random drop queue case with this policy, although the share ratio is still not close to uniform fairness. The improvement in the share ratio fairness in the random drop queue is probably because the number of congestion signals is less than in the tail drop queue case, and they are more evenly distributed to all the competing flows.

We split the result of the second policy (adjusting α quadratic to RTT) into two parts: one is the compensation for flows with small RTTs, the other is for flows with large RTTs. The policy works well for the compensation of flows with small RTTs. Basically, the compensation is achieved by using a small α to reduce the flow aggressiveness on bandwidth competition. Results from section 5.3.3 show that the share ratio of AIMD(α, 0.5) versus AIMD(1, 0.5) matches well in the small α case especially in the tail drop case. In the experiments in this section, the second policy works well in both cases for small α, and the tail-drop case does a little better than the random drop case. For the second part when a large α is required to compensate for long RTT flows, the uniform fairness is not achieved, although the bias against long RTT flows is reduced. The reason for this behavior can be explained with the result in section 5.3.3, in which large α AIMD flows are shown to be less effective in producing the predicted share ratio, and cause high congestion signals and low link utilizations. We believe mechanisms that smooth out the bursts of large α flows could help improve the effectiveness of this compensation policy.

In summary, from the results in this subsection, we conclude that uniform fairness can be achieved by adjusting the AIMD parameters. Other fairness ratios can also be achieved, but the ratio must not be too large. Otherwise, it will require α or β parameters to be set to a value that could cause many more congestion signals and timeouts, which degrades the link utilization. Achieving the preferred share ratio requires a careful selection of α and β parameters.

5.4 Summary

In this chapter, we conducted experiments to investigate various aspects of the bandwidth sharing behavior among competing AIMD flows. These aspects are the timescales for fairness, the bandwidth share ratio among theoretical TCP-friendly AIMD flows, and the share ratio of un-friendly AIMD flows.

- For the notion of fairness, our results show that perfect fairness in terms of exactly even bandwidth shares does not exist with identical TCP flows. Therefore it is not feasible to define friendliness as sharing bandwidth exactly equally. Friendliness should be more properly defined as follows: Flow-X is friendly to Flow-Y when statistically Flow-X will not get more bandwidth than Flow-Y. We also study the right measurement timescales for fairness. Our result shows that for identical TCP flows, larger timescales lead to a better fairness index, but even on a very large timescale, the flows will not achieve even share. The theoretical limit cycle of the system provides guidance for the minimum timescales for measuring average throughput in order to test inter-flow fairness.

- Theoretical TCP-friendly AIMD flows share bandwidth fairly with TCP flows in the tail-drop queue case under light congestion, because tail-drop queues tend to produce universal congestion signals. With a random drop queue, or with severe congestion, theoretical TCP-friendly AIMD flows do not share bandwidth fairly with TCP flows, and the TCP-friendly AIMD flows with a larger α get more bandwidth than those with a smaller α.

- The traditional TCP-friendly fairness and the share ratio prediction in (5.3) are derived based on the assumption of a universal congestion experience. The congestion perception actually varies as the AIMD parameter changes. Furthermore, the different queue management schemes on the bottleneck router can change a flow's congestion perception. When the parameters of AIMD flows are close, they are more likely to experience the same congestion signal, thus the prediction is more accurate at that time. Otherwise, the share ratio diverges from the prediction.

- The bandwidth share ratio of un-friendly AIMD flows is close to the theoretic prediction when their parameters are close. For example, compared to AIMD(1,0.5), AIMD(α, β) flows that have a very large α or a very small β tend to create more congestion signals and timeouts, and thus degrade link utilization.

By carefully tuning the α and β parameters, AIMD flows can achieve different share ratios.

The study of this chapter can be extended in many ways. We only investigated a small group of α and β parameters; one further study could try to cover all combinations of α and β parameters, and study bandwidth share behavior with TCP and other AIMD flows. Another potential work is to look at the share ratio when groups of different AIMD flows compete with each other, and the number of flows in each group are not necessarily equal.

On a final note, most of the experimental results, such as the result of the bandwidth share ratio of TCP-friendly flows with a random drop queue in 5.3.2 and the result of the impact of α and β on the share ratio of unfriendly AIMD flows in 5.3.3, match the theoretical study and simulation results from the Simulink simulation in Chapter 4. Hence, these results verify the results of the analytic model presented earlier in this thesis. At the same time, results from 5.3.3 and 5.3.4 also show the impact of different congestion signal perception on flows. A mathematical model that quantifies the relationship between the AIMD parameters and the experienced congestion signals and timeouts could help polish and complement the analytical model presented here.

Chapter 6

Related Work

Different aspects of congestion control behavior have been extensively studied during the last decade. These studies can be classified as theoretical either modeling, simulation, or system works. Theoretical modeling focuses on the key factors and algorithms of congestion control and predicts their behavior under various network parameters. A theoretical modeling work make general statements about congestion control behavior by making simplifying assumptions about the target system and the environment if operates in. Low-level details that distinguish one implementation of an algorithm from another are often ignore. In contrast, system works tend to focus on the details, such as the performance of a particular protocol implementation in a particular environment. Simulation studies are usually a compromise between these approaches. In this chapter we first discuss related theoretical research, then move on to discuss simulation studies and system works.

6.1 Theoretical Studies

In this section, we first summarize the previous theoretical studies on modeling the bandwidth sharing dynamics, and then a few theoretical models for TCP congestion control. Since the theoretical study in this thesis uses the feedback control theory, we also summarize some feedback control related work in computer systems.

6.1.1 Mathematical Modeling of the Bandwidth Sharing Dynamics

The dynamics of bandwidth sharing have been a research focus for more than two decades. It has attracted more researchers in recent years because of the development of multimedia applications in the Internet, which, due to their real-time requirement, are sensitive to various aspects of the transmission rate. The common goal of these modeling works is to verify dynamic aspects, such as link utilization and the fairness, of various congestion control algorithms. Chiu and Jain [CJ89] study the bandwidth sharing dynamics of a few algorithms using the phase plot approach, and conclude that the AIMD algorithm is the best one in terms of a good fairness and a fast convergence to fairness. Their study assumes synchronized congestion signals. Recently, Misra et al. [MGT00, CHM+02] use a stochastic fluid model for *N* homogenous TCP-controlled sources, and a *PI* control model for the active queue management. They approximate the aggregated TCP behavior with linear control functions. Their result shows that the model can predict the long-term aggregated rate behavior accurately. Similarly, the work by Baccelli and Hong using a max-plus algebra [BH00] presents an analytical study of the average throughput and system stability by linearizing the TCP congestion control.

The state space model in this thesis is also an analytical study of the bandwidth sharing dynamics among congestion controlled flows. The dynamic aspects include stability, fairness, and throughput over various timescales. The technique used in the state space model is similar to the one used in [CJ89], but we extend it to asynchronous congestion signals with different feedback delays. This state space model differs from other modeling work by its nonlinear approach. In the study of stability, it predicts the position of the system trajectory and its convergence rather than using the classic stability theory for linear systems. Compared to the linear approach, the advantage of our approach is that it lets us predict the per-flow behaviors in the short-term, and thus lets us study the interaction between the AIMD parameters and their congestion experiences. In contrast, the linear models [BH00, MGT00, CHM+02] are only good at predicting the aggregate behavior of many flows over large timescales.

6.1.2 Models for the TCP Congestion Control

Recently researchers have proposed a significant number of analytic models to characterize TCP performance in terms of round-trip-time and packet loss rate [MSM97, PFTK98, PFT99, FF99, YR99, and NST00]. They share the same goals as our TCP modeling work. First, to achieve a better understanding of how TCP performance is affected by various network parameters, and second to help the design of new TCP-friendly congestion control protocols.

These TCP models focus on characterizing bulk transfer throughput with TCP. The results of these models are often TCP throughput equations with round-trip-time and packet loss rate as inputs. However, they differ from each other by their assumptions and simplifications of TCP congestion stages. Mathis et al. [MSM97, FF99] focus on TCP steady state and ignore timeout. Padhye et al. [PFTK98, PFT99] adjust the result of Mathis et al. by taking timeouts into account. Yeon et al. [YR99] extend Mathis's throughput equation into a differentiated services network.

Our state space control model for AIMD TCP congestion control differs from these models in two major ways. First, all existing models focus on the long-term average throughput as the TCP performance metric, while our model is used to address their short term rates and buffering delays. Cardwell et al. derived a latency model [NST00] for TCP. However, the latency they defined in the model is the average time to deliver a certain amount of data in one TCP connection, which is more of a goodput metric than a metric for delay constraints. Second, our control model exposes the impact of AIMD parameters on the congestion experience, while all existing models assume the same congestion experience for all flows independent of their AIMD parameters.

6.1.3 Feedback Control Analysis of Computer Systems

The modeling work in this thesis is influenced by some previous work on feedback control systems in networks, operating systems, and adaptive applications.

In network research, the TCP congestion control protocol is heavily studied using control-theoretical approaches [JK88, RA02, S90]. In addition, linear feedback control was used to design other congestion control protocols. For example, Keshav applied linear feedback control and fuzzy logic control to design the packet pair congestion control [K91], assuming networks all perform fair queueing scheduling. He describes the congestion control with discrete differential equations, and proves packet pair congestion control is stable.

In operating system research, feedback control analysis has been applied to CPU resource management. Real-rate adaptive CPU schedulers [SGG+98] use feedback mechanisms to adjust CPU allocation so that processes maintain their application-specified progress rate. A similar idea is applied to the resource management of end-host network interface bandwidth [LWM+01]. Feedback control is also used to manage other OS resources [SMM98, RMP+99]. For example, Semke et al. use feedback control to automatically tune the memory allocation for TCP flows in one system to achieve high overall throughput.

In adaptive application research, especially in research on multimedia applications, feedback control analysis has been widely used. Multimedia applications have periodic resource requirements with tight jitter bounds. However, the resource requirement of the application varies from time to time, and at the same time the available resource varies because it is usually shared among multiple applications. This periodic requirement and the variance of both requirements and resources make multimedia applications ideal for feedback based control analysis. For example, multimedia applications, such as distributed streaming media players [C97, JE96, WKC+97], have been built based on feedback control systems that perform application-level QoS-adaptation.

The theoretical study in this thesis is also applying control theory to adaptive congestion control. The model for the AIMD TCP is based on control theory. It uses a hybrid control system model presented in the literature, rather than a linear control model, and hence differs from the traditional feedback control work listed above, which are mainly linear feedback control systems.

6.2 Network Simulations

While network technologies keep evolving, many aspects of the network behaviors are poorly understood. Due to the network's complexity, simulation plays a vital role in attempting to characterize how different facets of the network behave, and how a few parameter adjustments might affect the network's properties.

A number of network simulations have been developed in order to study various aspects of network behaviors. According to the detail produced in the simulations, these simulations can be divided into three levels: packet level, flow level and aggregate level.

Packet and aggregate level simulations have been developed for years and are widely used in the research community today. The most widely used network simulation is Network Simulator [NS], which is a packet level network simulator. NS is a multi-protocol simulator that implements most of the protocols used in today's Internet. NS models reality at the packet level. Simulated nodes and links process packets according to their headers except the packets do not need to carry real payloads. A packet level simulation enables us to study the protocol details, such as the hand-shakes of a connection setup, the reactions to packet loss and retransmissions etc. Ideally, it also supports the study of the aggregate behavior of many flows by simulating all of their packets. However, simulating a large number of flows over a large timescale using NS is extremely CPU intensive. Both the experiment setup and the result collections are very time consuming.

Aggregate level simulations approximate the behaviors of a group of flows with a mathematical model, such as the one by Misra et al. [MGT00]. Aggregate level simulations model a group of flows as one fluid, and the simulation complexity does not increase as the number of flows increases. Therefore, it is more efficient to predict the aggregated behavior of a large number of flows in a large timescale. On the other hand, aggregate level simulation can not provide as many details as packet level simulations do.

The network simulation in this thesis is a flow level simulation. It describes each flow as a fluid according to the differential equations in the state-space model. We choose to model the system behavior at the flow level because our research focus is the bandwidth sharing behaviors of each individual flow. Ignoring the packet level details dramatically reduces the number of states in the state-space model, so that we can describe the system trajectory in the state space. It's hard to capture the flow level dynamics with NS simulation because of the noise and randomness introduced by packet level details. Our flow level simulation differs from the aggregate level simulation by the fact that we still keep the information about per flow rate behavior, and thus can study the interactions between a flow's AIMD parameters and its congestion experiences. The aggregate level simulations target only the global average behaviors of many flows, and ignore the per-flow behavior.

6.3 System Works: TCP-friendly Congestion Control Protocols

To the study of the bandwidth sharing dynamics, the most closely related system work is the research on building TCP-friendly congestion control protocols. TCP-friendliness is proposed for new transport protocols to incorporate proper end-to-end rate adjustments that are "friendly" to TCP traffic and hence achieve the "fairness" that TCP flows achieve now. However, research on TCP-friendliness so far has not produced a clear result about how closely new end-to-end rate management should mimic TCP's behavior in order for it to be considered TCP-friendly. The major intuitive notion now for TCP-friendliness is that the new protocol should obtain "in average" the same bandwidth as a TCP flow along the same path. Note, however that its short-term transmission rate might be different from TCP's. We have shown in this thesis that even identical TCP flows won't share bandwidth evenly over short timescales.

Although the notion of TCP-friendliness is vague, numerous TCP-friendly congestion control protocols [BMP94, CPW98, FF99, FHPJ00, LPK+00, LRC99, PKTK99, RHE99a, ROY00, TCPF, and YL00] have been proposed for multimedia streaming applications. These TCP-friendly protocols can be divided into two categories, AIMD-based, and Equation-based TCP-friendly congestion control. AIMD-based TCP-friendly congestion control [LPK+00, RHE99, YL00] uses TCP's AIMD algorithm directly. Hence, it linearly probes available bandwidth and multiplicatively reacts to congestion, but decouples reliability control from congestion control. Some AIMD-based protocols [LPK+00, YL00] statically adjust the parameters of the AIMD algorithm that controls the pace of increment and decrement to get the desired rate behavior, for example, to produce a smoother rate. Equation-based TCP-friendly congestion control protocols [FF99, FHPJ00, LRC99, PKTK99] are based on TCP throughput equations derived from TCP models [MSM97, PFTK98, PFT99, FF99, YR99, and NST00] that have been mentioned in section 6.1. These models relate a TCP flow's throughput to its RTT and packet loss rate. Instead of doing AIMD style probing, equation-based protocols monitor RTT and packet loss rate, and use a throughput equation to determine the rate that a flow should sustain. They then directly limit the output rate of the flow to this target rate. The advantage of the Equation-based approach is that it can produce a smoothed output rate. However, its TCP-friendliness is based on the accuracy of the throughput equation it uses, and the time scale of this is still under intense research.

The study of bandwidth dynamics in this thesis is limited to AIMD algorithms, and we have shown that congestion experiences, such as congestion signals and timeouts, vary as AIMD parameters change. However, the study of sharing behavior of non-AIMD flows is equally important. Readers can imagine more significant changes of congestion experiences when the competing flows are between TCP and non-AIMD flows. Study of the dynamic behaviors of these protocols is one of our areas of future work.

We end this section with a briefly summary of two non-AIMD TCP-friendly congestion control algorithms: the *Binomial congestion control* algorithm and the *TCP-Friendly Rate Control* (TFRC) algorithm.

Binomial Congestion Control Algorithm

AIMD-based algorithms make the window adjustment proportional to the current window size. Binomial algorithms [BB01] generalize AIMD-based algorithms by extending the adjustment with nonlinear order of the current window size.

A binomial congestion control algorithm uses the following control rules:

$$\begin{array}{lll} \textit{Increase:} & W(t+RTT) = W(t) + \alpha / W^k(t) & \alpha > 0 \\ \textit{Decrease:} & W(t+\delta) \leftarrow W(t) - \beta W^l(t) & 0 < \beta < 1 \end{array} \quad (6.4)$$

The notations used in (6.4) are same as those in (2.2) except for the introduction of two other parameters *k* and *l*. W is still the window size, and RTT is the round-trip-time. As with an AIMD-based algorithm, a binomial congestion control protocol also must constrain its parameters to be TCP-friendly. In addition to the same constraint (2.2) on α and β, it requires $k+l=1$.

In Binomial algorithms, α and β have the same effect on the smoothness and responsiveness as in AIMD algorithms. Since parameters *k* and *l* are limited by $k+l=1$, their impacts can be shown on one of them: a smaller value of *l* leads to a smoother rate and a low responsiveness.

TCP-Friendly Rate-based Congestion Control (TFRC)

TFRC is an equation-based congestion control protocol. Instead of adjusting the transmission rate based on individual congestion signals, TFRC responds to the average time interval of congestion signals.

In order to make a rate-based congestion control TCP-friendly, it uses TCP throughput equations, such as (2.1), to determine its transmission rate as a function of packet loss rate, packet size, and round-trip-time. A typical rate-based TCP-friendly congestion control protocol is TFRC [FHPJ00, HPFW01]. In general, TFRC's congestion control mechanism works as follows:

1) The receiver measures the loss event rate and feeds this information back to the sender.
2) The sender also uses the feedback messages to measure the RTT.
3) The loss event rate and RTT are then fed into TFRC's throughput equation, giving the acceptable transmission rate.
4) The sender then adjusts its rate to the calculated rate.

The dynamics of TFRC are sensitive to how the measurements are performed and applied. An important parameter of TFRC is the number of history events used to calculate the lose rate. Generally, TFRC(*k*) refers to the protocol that only uses the most recent *k* events to calculate the loss rate. Since TFRC's average throughput is controlled by the equation, changing *k* won't affect its TCP-friendliness. Most recent work [FHPJ00, HPFW01] recommends TFRC(6). Generally, using a larger *k* would make TFRC smoother but less responsive.

TFRC is proposed to produce a smooth rate behavior rather than TCP, such that TFRC can server better for multimedia applications than TCP. TFRC achieves its smoothness by trading its responsiveness. This trade-off can be controlled by the parameter *k*. Similar smoothness and responsiveness trade-off can be done by tuning the AIMD parameters as shown in this thesis.

Chapter 7

Conclusions and Future Work

We conclude this dissertation with a summary of the thesis work, and then enumerate a few research problems that can be addressed in future work.

7.1 Summary of Contributions

This dissertation presented a state-space model and some real world experiments for studying the bandwidth competition between flows using AIMD-based congestion control protocols.

We modeled the bandwidth competition among competing flows by using their rate and the bottleneck queue fill-level as the key state variables in the state-space model. In this model, we defined the dynamic stability as a limit cycle of the system state trajectory. Early research has shown that the bandwidth competition of a system that has multiple AIMD-based flows is stable under synchronized congestion signals. In this dissertation, we showed that the system is stable even with asynchronous universal congestion signals. We built simulations based on the state-space model. Both the model and simulation only covered the AIMD algorithm used in the steady state of the TCP congestion control protocol. They did not cover the MIMD-based SlowStart algorithm. We used simulations to study the fairness of bandwidth competition among AIMD flows. While the TCP-friendly constraint on AIMD parameters is derived under the assumption of synchronous congestion signals, our simulations with random congestion signals that are proportional to the transient transmission rates of flows showed unfriendliness among "TCP-friendly" AIMD flows.

We compared the output from the simulations to the results from real-world experiments of bandwidth competition. The real-world experiments were conducted with an adaptive AIMD congestion control protocol that was built by us to produce AIMD flows with various parameters. AIMD parameter values dominate the behavior of congestion control protocols when congestion is light. The experiment result with a random drop queue showed that "TCP-friendly" AIMD flows do not share bandwidth evenly. The more the AIMD parameters of two flows differ, the more unevenness occurred in the bandwidth shares. This experiment result matched the output from our simulation. In the experiment with a tail drop queue, the "TCP-friendly" AIMD flows are more likely to get even shares. The reason is that a tail-drop queue causes universal congestion signals to both competing flows, and thus matches the assumption of the derivation of AIMD parameter constraint.

Our real-world experiments also covered the case of severe congestions, and the results showed that timeouts play an important role in determining the bandwidth share. Flows with different AIMD parameters experience different numbers of timeout events. Flows with a large α and a large β got more timeouts on average. This result indicates that an extension to the model and simulation with MIMD algorithms and the events that switch between AIMD and MIMD algorithms are very important to fully capture the bandwidth competition in various real-world conditions.

In addition to building the state-space model and verifing the model with real-world experiments, the dissertation also made the following contributions:

We investigated the possibility of achieving different fairness paradigms by tuning AIMD parameters. In particular, we showed that by tuning AIMD parameter, we can achieve uniform fairness that is un-biased to flows with different RTTs. However, our experiment results also showed that the range of the achieved share ratio is limited. When the preferred share ratio between a TCP and an AIMD flow is more than two times, the link utilization begins to degrade and eventually the technique becomes in-effective.

7.2 Future Work

In this section, we identify several future research problems that follows on from this dissertation.

7.2.1 Stochastic Stability

The study of stability in the state-space model relies on a precise description of the stable position of the system states in the state space. This method works well for the case with universal congestion signals. However, with non-universal congestion signals caused by contiguous random events, the system state won't stay on the theoretical limit cycle, but will reside in a region around it, during the steady state. With the introduction of such random events, the study based on the state-space model can only be performed using simulations.

Both simulations and real-world experiments indicate that system states after disturbances are still converging to the theoretical limit cycle in the case with universal congestion signals. The system is not unstable in the sense that the state does not become unbounded. The notion of stochastic stability in the control theory is exactly for this situation that the system state is in a region rather than a singular point or a limit cycle during the steady state. Therefore, we expect to extend the bandwidth competition model presented in this thesis to a stochastic state-space model. Some of the bottleneck queue management schemes, such as random drop queue, can be naturally modeled as a stochastic process. The bandwidth fairness monitored in experiments also shows stochastic behavior. How to capture and predict these behaviors accurately from a theoretical model is one of the future research directions identified in this dissertation.

7.2.2 Dynamic Congestion Control Adaptations

One goal of the research in this dissertation is to provide a flexible congestion control protocol that can fit to a wide range of applications. The adaptive AIMD congestion control protocol provides a framework for building various congestion control protocols. How to adjust the parameters is up to the requirements of applications, however, the adjustment is likely to be dynamic for applications that are running over heterogeneous networks. How to dynamically adapt the AIMD parameters or even switch to different congestion control algorithms, such as MIMD, is still under investigate. Ideally, a feedback control loop would be constructed with the application exposing some information to a controller, which would conduct adjustments based on the monitored result. We would like to extend the framework in this thesis to build this automatic control for a few application examples so that they can work well in a wide range of network conditions.

Bibliography

[AP99] Mark Allman, Vern Paxson. "On Estimating End-to-End Network Path Properties", In *Proceeding of ACM SIGCOMM 1999*, pp.263-274, 1999.

[APS99] Mark Allman, Vern Paxson, and Wright Stevens. "TCP Congestion Control", Request for Comments 2581, Internet Engineering Task Force, 1999. Available at http://www.ietf.org/rfc/rfc2581. Viewed: October 2002.

[B91] William L.Brogan. *Modern Control Theory.* Published by Prentice-Hall. 3rd edition. 1991.

[BB01] D. Bansal and H. Balakrishnan. "Binomial Congestion Control Algorithms", In *Proceedings of IEEE INFOCOM 2001*, v.2, pp.631-640, 2001.

[BCC+98] 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, L. Zhang. "Recommendations on Queue Management and Congestion Avoidance in the Internet". Request for Comments2309, Internet Engineering Task Force, 1998. Available at http://www.ietf.org/rfc/rfc2309. Viewed: October 2002.

[BBFS01] Deepak Bansal, Hari Balakrishnan, Sally Floyd and Scott Shenker. "Dynamic Behavior of Slowly-Responsive Congestion Control Algorithm". In *Proceedings of ACM SIGCOMM 2001*, pp.263-274, 2001.

[BH00] Francois Baccelli, Dohy Hong. "TCP is Max-Plus Linear". In *Proceedings of ACM SIGCOMM 2000*, pp.219-230, 2000.

[BLM01] John Byers, Michael Luby, and Michael Mitzenmacher. "Fine-Grained Layered Multicast", In *Proceedings of IEEE INFOCOM 2001*, v.2, pp.1143-1151, 2001.

[BMP94] L. S. Brakmo, S. W. O'Malley, and L. Peterson. "TCP Vegas: New Techniques for Congestion Detection and Avoidance". In *Proceedings of ACM SIGCOMM 1994*, pp.24-35, 1994.

[C86] Peter A.Cook. *Nonlinear Dynamical Systems.* Published by Prentice-Hall. 1986.

[C97] Shanwei Cen. "A Software Feedback Toolkit and its Application in Adaptive Multimedia Systems". *Ph.D Thesis Dissertation.* Oregon Graduate Institute, 1997.

[CJ89] Dah-Ming Chiu and Raj Jain. "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks". *Computer Networks and ISDN Systems* v.17, pp.1-14, 1989.

[CPW98] Shanwei Cen, Calton Pu, and Jonathan Walpole. "Flow and Congestion Control for Internet Streaming Applications". In *Proceedings of Multimedia Computing and Networking*, pp.250-254, 1998.

[FF99] Sally Floyd, and Kevin Fall. "Promoting the Use of End-to-End Congestion Control in the Internet" *IEEE/ACM Transactions on Networking*, August 1999. Available at http://www.aciri.org/floyd/papers.html. Viewed: August 2000.

[FH99] S. Floyd, and T. Henderson. "The newReno Modification to TCP's Fast Recovery Algorithm", Request for Comments2582, Internet Engineering Task Force, 1999. Available at http://www.ietf.org/rfc/rfc2582. Viewed: October 2002.

[FHP00] Sally Floyd, Mark Handley, and Jitendra Padhye. "A comparison of equation-based congestion control and AIMD-based congestion control." Under submission. Available at http://www.aciri.org/tfrc. Viewed: August 2000.

[FHPJ00] Sally Floyd, Mark Handley, Jitendra Padhye, and Jorg Widmer. "Equation-based Congestion Control for Unicast Applications." In *Proceedings of ACM SIGCOMM 2000*, pp.43-56, August 2000.

[FJ93] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance", *IEEE/ACM Transactions on Networking*, v.1, pp.397-413, August 1993.

[FKSS97] Wu-chang Feng, Dilip D. Kandlur, Debanjan Saha, and Kang G.Shin. "Understanding TCP Dynamics in an Integrated Services Internet". In *Proceeding of the 7th Internation Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 97)*, pp.279-290, 1997.

[FKSS99] W. Feng, D. Kandlur, D. Saha, K. Shin, "Blue: A New Class of Active Queue Management Algorithms" U. Michigan CSE-TR-387-99, April 1999.

[FLKP99] Wu-chi Feng, Ming Liu, B. Krishnaswami, A. Prabhudev. "A Priority-Based Technique for the Delivery of Stored Video Across Best-Effort Networks" in *Proceedings of Multimedia Computing and Networking (MMCN) 1999*. Available at: http://www.cis.ohio-state.edu/~wuchi/pubs.html. Viewed: October 2002.

[FP] FlowPair: A flow aggressiveness test tool. FlowPair's web page available at http://tcpradar.sourceforge.net. Viewed: August 2002.

[FR99] Wu-chi Feng, J. Rexford. "Performance Evaluation of Smoothing Algorithms for Transmitting Prerecorded Variable-Bit-Rate Video" in *IEEE Transactions on Multimedia*, v.1, num.3, pp.302-331, 1999.

[GKLW02] A.Goel, C. Krasic, K. Li, and J. Walpole. "Supporting Low Latency TCP-Based Media Systems", in *Proceedings of the Tenth International Workshop on Quality of Service*. Available at http://www.cse.ogi.edu/~ashvin. Viewed: October 2002.

[J91] Raj Jain. *The Art of Computer Systems Performance Analysis.* published by John Wiley & Sons Inc, 1991.

[JE96] S. Jacobs and A. Eleftheriadis. "Providing Video Services over Networks without Quality of Sevice Guarantees". In *Proceedings of World Wide Web Consortium Workshop on Real-time Multimedia and the Web*, 1996. Available at http://www.ee.columbia.edu/~eleft/papers/iwqos97.ps. Viewed: October 202.

[JK88] Van Jacobson, and Michael J. Karels. "Congestion Avoidance and Control". In *Proceedings of ACM SIGCOMM 1988*, pp.314-329, 1988.

[K91] Srinivasan Keshav. "A Control-Theoretic Approach to Flow Control". In *Proceedings of ACM SIGCOMM 1991*, pp.3-16, 1991.

[KWLG01] Charles Krasic, Jonathan Walpole, Kang Li, and Ashvin Goel. "The Case for Streaming Multimedia with TCP". in the 8th International Workshop on Interactive Distributed Multimedia Systems 2001, Lancaster, UK, September 2001. Also published in Springer Verlag`s *Lecture Notes in Computer Science Series LNCS* 2158, pp. 213-218, 2001.

[KW99] Charles Krasic and Jonathan Walpole. "QoS Scalability for Streamed Media Delivery", OGI CSE Technical Report CSE-99-11, 1999.

[Linux] Linux Kernel Source. Available at http://www.linux-kernel.org. Viewed: August 2002.

[LKW+01] Kang Li, Charles Krasic, Jonathan Walpole, Molly Shor, and Calton Pu, "The Minimal Buffering Requirements of Congestion Controlled Interactive Multimedia Applications", in the 8th International Workshop on Interactive Distributed Multimedia Systems 2001, Lancaster, UK, September 2001. Also published in Springer Verlag`s *Lecture Notes in Computer Science Series LNCS* 2158, pp.181-192, 2001.

[LPK+00] K.W. Lee, R. Puri, T. Kim, K. Ramchandran and V. Bharghavan, "An Integrated Source Coding and Congestion Control Framework for Video Streaming in the Internet". In *Proceedings of IEEE INFOCOM 2000*, v.2, pp.747-756, 2000.

[LRC99] L. Vicisano, L. Rizzo, and J. Crowcroft. "TCP-like Congestion Control for Layered Multicast Data Transfer". In *Proceedings of IEEE INFOCOM 1999*, v.3, pp.996-1003, 1999.

[LSW01] Kang Li, Molly H. Shor, and Jonathan Walpole. "Modeling the Effect of Short-term Rate Variations on TCP-Friendly Congestion Behavior". In *Proceedings of American Control Conference 2001*, v.4, pp.3006-3012, 2001.

[LWM+01] Kang Li, Jonathan Walpole, Dylan McNamee, Calton Pu and David Steere. "A Rate-Matching Packet Scheduler for Real-Rate Applications". In *Proceedings of Multimedia Computing and Networking (MMCN) 2001.* pp.49-61, January 2001.

[MATLAB] *Matlab 5.3 (Release 11): The language of Technical Computing*, by MathWorks, Inc, 1999.

[MGT00] Vishal Misra, Wei-Bo Gong, Don Towsley. "A Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED". In *Proceedings of ACM SIGCOMM 2000.* pp.151-160, 2000.

[MMFR96]Matthew Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. "TCP Selective Acknowledgement Options". Request for Comments2018, Internet Engineering Task Force, 1996. Available at http://www.ietf.org/rfc/rfc2018. Viewed: October 2002.

[MSM97] Matthew Mathis, Jeffrey Semke, and Jamshid Mahdavi. "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm". *ACM Computer Communication Review*, v.27, num.3, pp.67-82, 1997.

[NIST] NISTNet: a Network Emulation Package. NISTNet's web page is available at http://snad.ncsl.nist.gov/itg/nistnet/. Viewed: August 2002.

[NS] ns: UCB/LBNL/VINT Network Simulator (Version 2). NS version 2 is available at http://www-mash.cs.berkeley.edu/ns/ns.html. Viewed: August 2002.

[NST00] Neal Cardwell, Stefan Savage, and Thomas Anderson. "Modeling TCP Latency". In *Proceedings of IEEE INFOCOM 2000*, v.3, pp.1742-1751, 2000.

[P99] V. Paxson, "End-to-End Internet Packet Dynamics", *IEEE/ACM Transactions on Networking*, v.7, num.3, pp.277-292, June 1999.

[PAD+99] V. Paxon, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke, and B. Volz. "Known TCP Implementation Problems", Request for Comments 2525, Internet Engineering Task Force, 1999. Available at http://www.ietf.org/rfc/rfc2525. Viewed: October 2002.

[PF01] Jitendra Padhye, and Sally Floyd. "On Inferring TCP Behavior". *In Proceedings of ACM SIGCOMM 2001*, pp.287-298, 2001.

[PFT99] Jitendra Padhye, Victor Firoiu, and Don Towsley. "A Stochastic Model of TCP Reno Congestion Avoidance and Control". *Technical Report CMPSCI 99-02*, University of Massachusetts.

[PFTK98] Jitendra Padhye, Victor Firoiu, Don Towsley and Jim Kurose. "Modeling TCP Throughput: A Simple Model and its Empirical Validation". In *Proceedings of ACM SIGCOMM 1998*, pp.303-314, 1998.

[PKTK99] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. "A model based TCP-friendly rate control protocol". In *Proceedings of the Ninth International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV) 1999*, pp.137-151, 1999.

[RA02] Priya Ranjan, Eyad H.Abed, "Bifurcation Analysis of TCP-RED Dynamics", in *Proceedings of the American Control Conference*, 2002. Available at: http://www.glue.umd.edu/~priya/home/mywork.html. Viewed: October 2002.

[RFB01] K. Ramakrishnan, S. Floyd, and D. Black. "The Addition of Explicit Congestion Notification (ECN) to IP", Request for Comments 3168, Internet Engineering Task Force, 2001. Available at http://www.ietf.org/rfc/rfc3168. Viewed: October 2002.

[RHE99a] R. Rejaie, M. Handley, and D. Estrin. "An End-to-End Rate-Based Congestion Control Mechanism for Realtime Streams in the Internet". In *Proceedings of IEEE INFOCOM 1999*, v.3, pp.1337-1345, 1999.

[RHE99b] R. Rejaie, M. Handley, and D. Estrin. "Quality Adaptation for Congestion Controlled Video Playback over the Internet". In *Proceedings of ACM SIGCOMM 1999,* pp.189-200, 1999.

[RMP+99] Dan Revel, Dylan McNamee, Calton Pu, David Steere, and Jonathan Walpole. "Feedback Based Dynamic Proportion Allocation for Disk I/O". *OGI Technical Report CSE-99-001.*

[ROY00] Injong Rhee, Volkan Ozdemir, and Yung Yi. "TEAR: TCP emulation at receivers - flow control for multimedia streaming". Technical Report is available at http://www.csc.ncsu.edu/eos/users/r/rhee/WWW/export/tear_page. Viewed: August 2001.

[S90] Scott Shenker. "A Theoretical Analysis of Feedback Flow Control". In *Proceedings of ACM SIGCOMM 1990*, pp.156-165, 1990.

[SIMU] State-space based MATLAB Simulink Simulation. Available at http://www.cse.ogi.edu/~kangli/simulink.html. Viewed: August 2002.

[SLW+00] Molly H. Shor, Kang Li, Jonathan Walpole, David C. Steere, and Calton Pu. "Appication of Control Theory to Modeling and Analysis of Computer Systems". In *Proceedings of the Japan-USA-Vietnam Workshop on Research and Education in Systems, Computation and Control Engineering,* HoChiMinh City, Vietnam, June 7-9, 2000. Available at: http://www.cse.ogi.edu/~kangli. Viewed: October 2002.

[SMM98] Jeffrey Semke, Jamshid Mahdavi, and Matthew Mathis. "Automatic TCP Buffering Tuning". In *Proceedings of ACM SIGCOMM 1998*, pp. 315-323, 1998.

[SGG+98] David C. Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu and Jonathan Walpole. "A Feedback-driven Proportion Allocator for Real-Rate Scheduling", In *ACM Operating System Review, Proceedings of Operating Systems Design and Implementation (OSDI), Special Issue-Winter 1998*, Page 145-228, 1998.

[TCPF] The TCP-Friendly Website. Web page is available at http://www.psc.edu/networking/tcp_friendly.html. Viewed: August 2002.

[V00] A. Veres, "The Chaotic Nature of TCP Congestion Control", in *Proceedings of IEEE INFOCOM 2000*, v.3, pp.1715-1723, 2000.

[WKC+97] Jonathan Walpole, Rainer Koster, Shanwei Cen, Crispin Cowan, David Maier, Dylan McNamee, Calton Pu, David Steere and Liujin Yu, "A Player for Adaptive MPEG Video Streaming Over The Internet," in *Proceedings 26th Applied Imagery Pattern Recognition Workshop AIPR-97*, SPIE, Washington DC, October 15-17, 1997.

[WLG97] R.Wegenkittl, H.Loffelmann, and E.Groller, "Visualizing Dynamical Systems of Higher Dimensions," in IEEE Visualization 97 Conference in Phoenix, AZ. Software related to "Visualization of Complex Dynamical Systems" is available on http://www.cg.tuwien.ac.at/research/vis/dynsys. Viewed: September 2001.

[WS95] Gary R. Wright, and W. Richard Stevens, "*TCP/IP Illustrated Volume 2, The Implementation*", Addison-Wesley Publishing Company, 1995.

[YL00] Yang Yang, and Simon Lam. "General AIMD Congestion Control" In *Proceedings of International Conference on Network Protocols (ICNP) 2000*, Osaka, Japan, Nov 2000. Available at http://www.cs.utexas.edu/users/lam/NRL/TechReports. Viewed: September 2001.

[YR99] Ikjun Yeom, and A. L. Narasimha Reddy. "Modeling TCP Behavior in a Differentiated Services Network". *TAMU ECE Technical Report*, May 1999. Available at http://ee.tamu.edu/~reddy/papers/index.html. Viewed: September 2000.

Appendix A1

Proof of Dynamic Stability

Theorem-1:

When multiple AIMD flows compete for a constant available bandwidth R, the system state, under the assumption of synchronized back off (with a common backward delay BD and a common forward delay FD), and packet conservation, converges to a limit cycle that passes through the point $P = [r_1, r_2, \cdots, r_N, B]^T$, *in which*

$$r_i = \frac{2\alpha_i(1-\beta_i)}{\beta_i} \times \frac{MSS_i}{RTT_i^2} \times \frac{R'}{\sum_{j=1}^{N} \frac{\alpha_j(2-\beta_j)MSS_j}{\beta_j RTT_j^2}} \qquad \text{(A1.1),}$$

where

$$R' = R + \frac{BD}{2} \times \sum_{j=1}^{N} \frac{\alpha_j MSS_j}{RTT_j^2} + \frac{FD}{2} \times \sum_{j=1}^{N} \frac{\alpha_j MSS_j}{RTT_j^2} \qquad \text{(A1.2).}$$

Here, BD is the backward feedback delay, FD is the forward delay, MSS_i is the packet size of flow i, RTT_i is the round-trip-time of flow i, (α_i,β_i) is the AIMD parameter, and these parameters must satisfy the following constraint:

$$\sum_{i=1}^{N} \left(\frac{1-\beta_i}{\beta_i} \times \frac{\alpha_i MSS_i}{RTT_i^2}\right) < \frac{R}{(BD + FD)} \qquad \textit{(A1.3)}$$

When all the flows have the same AIMD parameters, MSS, and RTT, they get equal bandwidth share.

We argue in Theorem-1 that the given limit cycle is stable. The proof of Theorem-1 is in 3 steps. First, we prove that system trajectories starting at any point would have a series of cross points with the plane *fl=B* in the state-space. Second, we prove the trajectory starting from the point stated in Theorem-1 comes back to the same point, and thus it is a limit cycle. Third, we prove the crossing points of a trajectory with the plane converge to the point *P* in the Theorem-1. We present each step as follows:

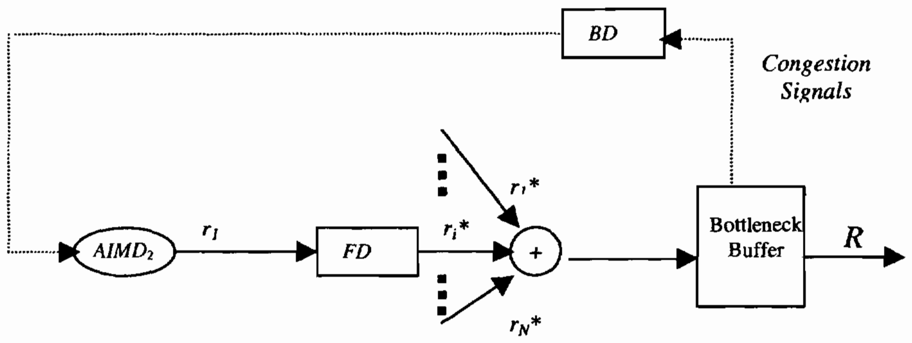


Figure A1.1: A Flow's Rates and Delay

Step-I

In this step, we prove that, for any starting state $X=[r_1,r_2,\cdots,r_N,fl]^T$, the system trajectory starting from X intersects plane *fl=B* again and again.

Figure A1.1 shows a single flow within a bandwidth sharing system (Figure 3.3). Because of the forward delay *FD*, the AIMD flow *i*'s input rate r_i' to the bottleneck buffer is not exactly the output rate r_i of the flow *i*, but r_i delayed by a time interval *FD*.

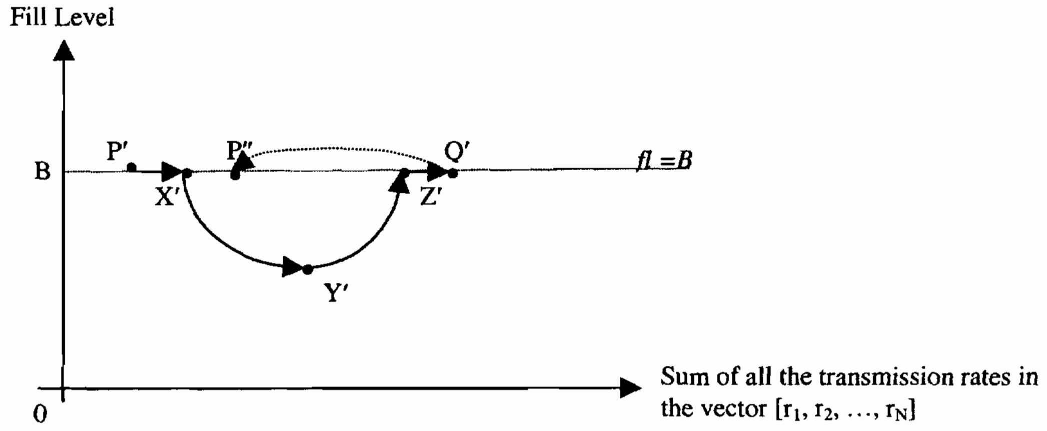

Fill Level
B
P'
X'
P''
Q'
Z'
Y'
fl =B
0
Sum of all the transmission rates in the vector [r₁, r₂, ..., r_N]

Figure A1.2: The System State Migration as Multiple Rounds. (A dotted line indicates the state transition between two points on fl=B)

As the result of the forward and backward delays, the system state migration in the state space is illustrated in Figure A1.2. In this figure, we assume that i) the system is started from a point P' in the plane $fl=B$ and, in addition, the sum of the rate of all flows $\sum_{i=1}^{N} r_{i\_P'}$ is less than the bottleneck link rate R. Here $r_{i\_P'}$ is the output rate from flow i at point P'; ii) before these rate values propagate to the buffer, the rate $\sum_{i=1}^{N} r^*_{i\_P'}$ is larger than R. Therefore, the buffer fill-level keeps at B before the rate $\sum_{i=1}^{N} r_{i\_P'}$ propagates to the buffer. This period is FD, and the system state is indicated by the trajectory between P' and X'.

After this period, the buffer fill-level starts to decrease because the input rate to the buffer is now $\sum_{i=1}^{N} r^*_{i\_X'} = \sum_{i=1}^{N} r_{i\_P'}$, and is smaller than R according to the assumption. Since the rate increment per flow $\frac{\alpha_i \times MSS_i}{RTT_i^2}$ is a positive value, and always larger than a non-zero constant $\frac{\alpha_{\min} \times MSS_{\min}}{RTT_{\max}^2}$, in which $\alpha_{min}=\min[\alpha_1, \alpha_2, ..., \alpha_N]$, $MSS_{min}=\min[MSS_1, MSS_2, ..., MSS_N]$, and $RTT_{max}=FD+BD+B/R$. Thus we know that the input rate to the buffer keeps increasing, and after a while, the input rate to the buffer is equal to R.[1] In Figure A1.2, this system state is indicated by point Y'. The time takes the system to migrate from X' to Y' is $(R-\sum_{i=1}^{N} r_{i\_P'}) / \sum_{i=1}^{N} \frac{\alpha_i \times MSS_i}{RTT_i^2}$.

After the system state passes point Y', the buffer fill-level starts increasing until the fill-level reaches its limit B. The fill-level increasing period $Y'Z'$ is symmetric to the fill-level decreasing period $X'Y'$, and thus takes the same time period $(R-\sum_{i=1}^{N} r_{i\_P'}) / \sum_{i=1}^{N} \frac{\alpha_i \times MSS_i}{RTT_i^2}$.

[1] Notice here that the total input rate to the buffer is $\sum_{i=1}^{N} r^*_{i\_Y'}$ $=R$, however, the total output rate of the flows is $\sum_{i=1}^{N} r_{i\_Y'}$ $>R$, because the input rate to the buffer $\sum^{N} r^*_{i\_Y'}$ is a delayed value of $\sum_{i=1}^{N} r_{i\_Y'}$.

A buffer overflow event is generated at point *Z'*. After the backward delay *BD*, the congestion signal is propagated to all the AIMD flows, and then causes a system state jump from *Q'* to *P"*. At state *P"*, the system is back to a similar condition to point *P'*. Point *P"* is similar to point *P'* because i) the sum of the output rate $\sum_{i=1}^{N} r_{i\_P''}$ is guaranteed to be less than *R*. Otherwise all the flows rate will keeps backing off until the total rate is less than the bottleneck, according to the packet conservation rule of the congestion control; and ii) the input rate to the buffer $\sum_{i=1}^{N} r^*_{i\_P''}$ is still higher than R before the flows' rates at *P"* propagate to buffer. Since Point *P"* is similar to point *P'*, the system keeps repeating the above process again.

As further proof, we divide the system migration along a trajectory into multiple rounds. The system state from any point would go across the plane *fl=B* followed by a state-jump. We let the state-jump be the end of each round. Thus, each round starts with a state just after a backing off upon a congestion signal, and ends with the state jump on the next congestion signal.

Step-II

To verify that the trajectory starting from the point *P* in Theorem-1 is a limit cycle, we need to prove that the trajectory comes back to *P*.

From the parameter constraint (A1.3), we can have

$$\frac{1}{2}(BD+FD)\sum_{i=1}^{N}\frac{\alpha_i MSS_i}{RTT_i^2} < R'\times\sum_{i=1}^{N}\frac{\alpha_i MSS_i}{RTT_i^2} / \sum_{i=1}^{N}\left(\frac{2-\beta_i}{\beta_i}\times\frac{\alpha_i MSS_i}{RTT_i^2}\right) \qquad \text{(A1.5)}$$

From the definition of point *P* in (A1.1), we can have

$$\sum_{i=1}^{N} r_i = \sum_{i=1}^{N}\left(\frac{2\alpha_i(1-\beta_i)}{\beta_i}\times\frac{MSS_i}{RTT_i^2}\right)\times\frac{R'}{\sum_{j=1}^{N}\frac{\alpha_j(2-\beta_j)MSS_j}{\beta_j RTT_j^2}}\frac{\beta_i}{1-\beta_i} \qquad \text{(A1.6)},$$

and combined with the definition of *R'* in (A1.2), we have

$$\sum_{i=1}^{N} r_i + \sum_{i=1}^{N} \frac{\beta_i}{1-\beta_i} r_i = R' = R + \frac{1}{2}(BD+FD)\sum_{i=1}^{N}\frac{\alpha_i MSS_i}{RTT_i^2} \qquad \text{(A1.7).}$$

From (A1.5) and (A1.7), we know that $\sum_{i=1}^{N} r_i < R$. Therefore, for the state of point *P*, we have $\sum_{i=1}^{N} r_{i\_P} < R$ and thus, according to (3.8), although point P is on the plane *fl=B*, there are no immediate state transitions from point P. According to Step-I, we know eventually, the system state will come back to plane *fl=B* again and again. Let us assume the very next state on plane *fl=B* before backing off is $Q=[r_{1\_Q}, r_{2\_Q}, \ldots, r_{N\_Q}]^T$. According to the state migration procedure in Step-I, the time period for the system migrate from *P* to *Q* is

$$T_{P\_Q} = FD + 2\frac{R - \sum_{i=1}^{N} r_i}{\sum_{i=1}^{N}\frac{\alpha_i MSS_i}{RTT_i^2}} + BD \qquad \text{(A1.8)}$$

And the position of point *Q* can be calculated based on (A1.1), and (A1.8)

$$r_{i\_Q} = r_i + T_{P\_Q} \times \frac{\alpha_i MSS_i}{RTT_i^2} = \frac{2\alpha_i}{\beta_i} \times \frac{MSS_i}{RTT_i^2} \times \frac{R'}{\sum_{j=1}^{N}\frac{\alpha_j(2-\beta_j)MSS_j}{\beta_j RTT_j^2}} = \frac{r_i}{1-\beta_i} \qquad \text{(A1.9)}$$

According to (A1.9), after the state jump happened on point *Q*, the state after the jump is exactly point *P*. Therefore, the specific trajectory described in the Theorem-1 is a closed and passing through point *P*. This proves that the system trajectory with *P* on it is a limit cycle. The period of the limit cycle once is equal to T_{P_Q}, which can be written as

$$T = \frac{\sum_{j=1}^{N}\frac{r_j}{1-\beta_j} - \sum_{j=1}^{N} r_j}{\sum_{j=1}^{N}\frac{\alpha_i MSS_i}{RTT_j^2}} = \frac{\sum_{j=1}^{N}\frac{\beta_j}{1-\beta_j} r_j}{\sum_{j=1}^{N}\frac{\alpha_i MSS_i}{RTT_j^2}} \qquad \text{(A1.10).}$$

If all flows have the same α parameters and β parameters, we have

$$T = \frac{\frac{2\beta}{2-\beta} R}{\alpha \sum_{j=1}^{N}\frac{MSS_i}{RTT_j^2}} + BD + FD \qquad \text{(A1.11)}$$

Step-III

As defined in Step-I, each round starts with a state just after a backing off upon a congestion signal, and ends with a state jump on the next congestion signal. We compare the distance of the system state at the beginning of every round to the point P in Theorem-1. If the distance is converging to zero, it indicates that the system state vector is approaching the limit cycle. Furthermore, we know if the system state ever reaches the above limit cycle, it will stay on it, unless there is noise to cause the system state to leave the limit cycle again.

We assume that the system states of the flows start from a random initial condition $P' = [r_1 + \Delta_1', r_2 + \Delta_2', \cdots, r_N + \Delta_N', B]^T$, and the system states arrive at $P'' = [r_1 + \Delta_1'', r_2 + \Delta_2'', \cdots, r_N + \Delta_N'', B]^T$ after one more round. The position of the system is represented in a way to emphasize the distance from the one state to the point $P = [r_1, r_2, \cdots, r_N, B]^T$. If we can prove that the distance Δ_i' on each dimension becomes smaller and smaller, then we know that the system state gets closer to P.

The way we prove the distance becoming small is to prove the ratio $\frac{\sum_{i=1}^{N}(\Delta_i'')^2}{\sum_{i=1}^{N}(\Delta_i')^2}$ is always less than a constant that is smaller than one. To prove this, we first derive the following relationship between Δ_i' and Δ_i''.

To derive the relationship, we have

$$r_i + \Delta_i'' = (1-\beta_i)[r_i + \Delta_i' + \frac{\alpha_i MSS_i}{RTT_i^2} \times \frac{2}{\sum_{j=1}^{N} \frac{\alpha_i MSS_i}{RTT_j^2}} (R' - \sum_{j=1}^{N}(r_j + \Delta_j'))] \qquad \text{(A1.12).}$$

and since $R' = \sum_{j=1}^{N} \frac{2-\beta_j}{1-\beta_j}(r_j + \Delta_j')$, we have

$$r_i + \Delta_i'' = (1-\beta_i)[r_i + \Delta_i' + \frac{\alpha_i MSS_i}{RTT_i^2} \times \frac{1}{\sum_{j=1}^{N} \frac{\alpha_i MSS_i}{RTT_j^2}} \sum_{j=1}^{N} (\frac{\beta_j}{1-\beta_j} r_j - 2\Delta'_j)] \qquad \text{(A1.13).}$$

Since Δ_i' and Δ_i'' are independent of r_i, we have

$$\Delta_i'' = (1-\beta_i)[\Delta_i' - \frac{\alpha_i MSS_i}{RTT_i^2} \times \frac{2}{\sum_{j=1}^{N} \frac{\alpha_j MSS_j}{RTT_j^2}} \sum_{j=1}^{N} \Delta'_j] \qquad \text{(A1.14).}$$

Let $A_i = \frac{\alpha_i MSS_i}{RTT_i^2}$, and rewrite the above equation as

$$\Delta_i'' = (1-\beta_i)[\Delta_i' - \frac{2A_i}{\sum_{j=1}^{N} A_j} \sum_{j=1}^{N} \Delta_j'] = (\beta_i - 1)[\frac{\sum_{j=1}^{N} 2A_i\Delta_j' - \sum_{j=1}^{N} A_j\Delta_i'}{\sum_{j=1}^{N} A_j}] \qquad \text{(A1.15).}$$

Because $0 < \beta < 1$, we can have the following relationship from (A1.15):

$$\begin{aligned} \sum_{i=1}^{N} (\Delta_i'')^2 &= \sum_{i=1}^{N} \{(\beta_i - 1)^2 [\frac{\sum_{j=1}^{N} (2A_i\Delta_j' - A_j\Delta_i')}{\sum_{j=1}^{N} A_j}]^2\} \\ &\le (\min[\beta_1, \beta_2, ..., \beta_N] - 1)^2 \sum_{i=1}^{N} [\frac{\sum_{j=1}^{N} (2A_i\Delta_j' - A_j\Delta_i')}{\sum_{j=1}^{N} A_j}]^2 \\ &< \sum_{i=1}^{N} [\frac{\sum_{j=1}^{N} (2A_i\Delta_j' - A_j\Delta_i')}{\sum_{j=1}^{N} A_j}]^2 = \sum_{j=1}^{N} (\Delta_j')^2 \end{aligned} \qquad \text{(A1.16).}$$

Thus we proved that $\frac{\sum_{i=1}^{N} (\Delta_i'')^2}{\sum_{i=1}^{N} (\Delta_i')^2} \le (\min[\beta_1, \beta_2, ..., \beta_N] - 1)^2 < 1$ when $0 < \beta < 1$, which indicates that the distance to the limit cycle becomes smaller as time goes by. Therefore, we know that the product of that formula with the next formula, etc., will result in the product of various numbers that are all less than one. As time goes to infinity, the distance to the limit cycle approaches zero. Thus, the limit cycle is stable.

Appendix A2

GAIMD Throughput and Buffer Requirement

A.2.1 Average Throughput of a GAIMD Congestion Control

For a flow that uses GAIMD algorithm described in (2.2), we can estimate its throughput given its packet loss probability *p*.

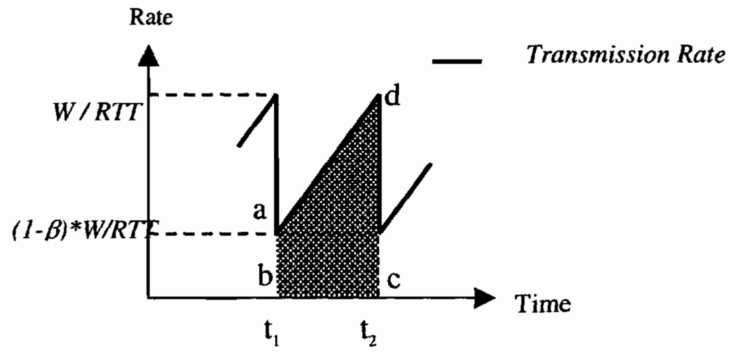


Figure A2.1: Throughput Derivation for an AIMD flow

Figure A2.1 illustrates the flow's rate oscillations along the time, assuming the flow's packet losses are evenly distributed along the time. Because of these periodic packet losses, the flow's congestion window shows a saw-tooth pattern. We assume the flow's congestion window size reaches *W* upon the arriving of every packet loss event, and backing off to (1-β)*W* after the event. Thus, the flow's rate keeps oscillating between *W/RTT* and (1-β)*W/RTT*.

Since a GAIMD flow increase its window size by α**MSS* per *RTT*, the time for the flow's congestion window increasing from β*W* to *W* can be derived by:

$$t_2 - t_1 = \frac{\beta W}{\alpha MSS} RTT \qquad \text{(A2.1).}$$

The total amount of data sent out during this time (t_2-t_1) is indicated by the area of the shaded region *abcd,* which can be derived as:

$$Area(abcd) = (1-\beta)W + ((1-\beta)W + \alpha MSS) + ((1-\beta)W + 2\alpha MSS) + ... + W = \frac{2\beta-\beta^2}{2\alpha MSS}W^2 \quad \text{(A2.2).}$$

Since one packet of every *Area(abcd)/MSS* amount of packets is lost and the packet loss probability is *p*, we can have

$$\frac{1}{p} = \frac{Area(abcd)}{MSS} = \frac{2\beta-\beta^2}{2\alpha MSS^2}W^2 \quad \text{(A2.3).}$$

Thus, we have

$$W = \frac{\sqrt{\frac{2\alpha}{(2\beta-\beta^2)}} \times MSS}{\sqrt{p}} \quad \text{(A2.4).}$$

Finally, the average throughput of the flow can be derived by:

$$\overline{R} = \frac{Area(abcd)}{t_2 - t_1} = \frac{(2-\beta)W}{2RTT} = \sqrt{\frac{\alpha}{2} \times \frac{2-\beta}{\beta}} \times \frac{MSS}{RTT\sqrt{p}} \quad \text{(A2.5).}$$

For TCP with AIMD parameter α=1 and β=½, its throughput can be expressed as

$$\overline{R} = \frac{\sqrt{3/2} \times MSS}{RTT\sqrt{p}} \quad \text{(A2.6).}$$

If an GAIMD flow wants to have the same average throughput as TCP when they share the same *RTT,* packet size *MSS*, and packet losses rate p, the GAIMD flow's α and β parameters have to satisfy the following equations:

$$\sqrt{\frac{\alpha}{2} \times \frac{(2-\beta)}{\beta}} = \sqrt{\frac{3}{2}} \quad \text{(A2.7),}$$

which can be further simplified as

$$\alpha = \frac{3\beta}{2-\beta} \quad \text{(A2.8).}$$

A.2.2 Buffer Requirement of a GAIMD Congestion Control

The rate of a GAIMD flow varies because of its way probing bandwidth and making congestion avoidance. Once the transmission rate is lower than the receiver play out rate, users will perceive the transmission rate oscillations unless there is receiver side data buffering. Receiver side buffering is a popular way to tolerant this rate oscillation. The amount of receiver-side buffering is needed for the transmission rate to catch up the playing out rate.

Figure A2.1 shows a GAIMD flow with a playing out rate R. We assume the GAIMD flow's transmission rate periodically varies from (1-β) * W / RTT to W/RTT.

Since the playing out rate can not be higher than the average the transmission rate, (Otherwise, it will run out the receiver side buffer), the playing out rate is limited by:

$$R = (\frac{W}{RTT} + (1-\beta)\frac{W}{RTT})/2 \qquad \text{(A2.9).}$$

With this playing out rate, the required data buffering size to avoid receiver side buffer underflow is equal to the area of triangle abc in Figure 2.1, which is:

$$\Delta abc = \frac{1}{2}[t_2 - t_1][R-(1-\beta)\frac{W}{RTT}] = \frac{1}{2}[\frac{\beta W}{2\alpha MSS}RTT][\frac{2-\beta}{2}\times\frac{W}{RTT}-(1-\beta)\frac{W}{RTT}] = \frac{\beta^2}{8\alpha MSS}W^2 \qquad \text{(A2.10).}$$

Since playback rate R is equal to the average of the transmission rate, we have $W = \frac{2}{2-\beta} R \times RTT$, and the receiver-side buffering is

$$\Delta abc = \frac{1}{2\alpha MSS}\times(\frac{\beta}{2-\beta})^2 \times R^2 \times RTT^2 \qquad \text{(A2.11).}$$

Biographical Note

Kang Li received his B.S. from the computer science department in Tsinghua University, Beijing China, 1995. He joined Oregon Graduate Institute in 1997. His research interests include network congestion control, network measurement, operating systems, and multimedia applications.