An Adaptive Packet Size Approach to TCP Congestion Control

by

Steven Wilson

B.Sc., Dalhousie University, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

October 2008

© Steven Wilson, 2008

Abstract

This thesis investigates the ability for TCP, the dominant transport protocol used in the Internet, to fairly allocate bandwidth to all competing sources under conditions of severe congestion. The conditions necessary for such congestion are quite common, and we have observed that the TCP congestion control algorithm does a poor job of balancing the bandwidth fairly among competing flows. We present a potential solution to this problem by using a variable maximum segment size in TCP. We show the benefits gained by doing this along with some of the problems encountered in the course of this work.

Contents

Abstract ii

Contents iii

List of Figures v

Acknowledgments vi

Chapter 1 Introduction 1

1.1 TCP Service Model 2

1.2 Packet Size and Throughput 2

1.3 TCP and Fair Sharing 4

1.4 Large Packets Considered Harmful 6

1.5 Thesis Outline 7

Chapter 2 Overview of TCP 9

2.1 Basic TCP 9

2.1.1 TCP Header 10

2.1.2 Reliable Delivery 10

2.1.3 Flow Control 12

2.2 TCP Congestion Control 13

2.2.1 Additive Increase/Multiplicative Decrease (AIMD) 14

2.2.2 Slow Start 15

2.3 Chapter Summary 16

Chapter 3 Literature Review **17**

3.1 LTTCP and Variable Packet Size 17

3.2 TFRC and Variable Packet Size 18

Chapter 4 Experimental Design and Analysis of Variable Packet Size in TCP **20**

4.1 Experimental Design 20

4.2 Fairness 21

4.2.1 The Max/Min Metric 22

4.2.2 The Jain Index 23

4.2.3 Cumulative Distribution Function 23

4.3 Phase One and the NS-2 Network Simulator 25

4.3.1 Round One Experiments 26

4.3.2 Round Two Experiments 27

4.4 Phase Two and the Emulab Testbed 32

4.5 Post Research Results and Discussion 35

4.5.1 TCP Smart Framing 36

4.5.2 TCP and Thin Streams 37

4.5.3 Active Queuing and Implications 37

Chapter 5 Conclusions and Future Work **39**

Bibliography **40**

List of Figures

1.1 TCP/IP Header Overhead 3

1.2 Average CWND vs RTT 8

2.1 TCP Header Structure 11

2.2 Congestion Avoidance Sawtooth Pattern 15

4.1 Network Topology for Experiments 21

4.2 CDF of Fair Network 24

4.3 CDF of Unfair Network 25

4.4 Round 1 - 50 Flows : 1 Second Bin 28

4.5 Round 1 - 100 Flows : 1 Second Bin 28

4.6 Round 1 - 200 Flows : 1 Second Bin 29

4.7 Round 1 - 400 Flows : 1 Second Bin 29

4.8 Round 1 - 50 Flows : 10 Second Bin 30

4.9 Round 1 - 100 Flows : 10 Second Bin 30

4.10 Round 1 - 200 Flows : 10 Second Bin 31

4.11 Round 1 - 400 Flows : 10 Second Bin 31

4.12 Round 2 - 50 Flows : 1 Second Bin 33

4.13 Round 2 - 50 Flows : 10 Second Bin 33

4.14 Round 2 - 200 Flows : 1 Second Bin 34

4.15 Round 2 - 200 Flows : 10 Second Bin 34

Acknowledgments

I would like the thank my supervisor Charles "Buck" Krasic for his incredible patience, which without, this work never would have completed. Also, without the love and support of my parents, I would never have reached this point in my academic career. I appreciate the discussions had with members of DSG which helped to inspire some avenues of investigation when experiments were not going well. Finally, thank you to all that have come in and out of my life over the time of this program, you all have shaped my life in many different ways to help lead me to where I am today.

Chapter 1

Introduction

As the number of latency sensitive applications on the Internet grows, we need to consider the ability of these applications to obtain the necessary bandwidth needed to function within desired specifications. If a situation arises where bandwidth is not available, these applications, such as video streaming, may suffer dramatically and possibly incur total failure. The Internet is currently, and likely to remain, a best effort service. This means we cannot make any absolute guarantees about the bandwidth availability at any given time. We therefore rely on the governing transport protocol to do a reasonable job of allowing competing flows to share bandwidth in a nature deemed to be fair to all applications. TCP is the dominant transport protocol used in the Internet, so to understand how flows will share bandwidth, we must investigate how TCP operates under certain conditions.

When designing a transport protocol such as TCP, the selection of appropriate packet size to use is a fundamental decision. In this thesis we will examine how the chosen packet size can influence the underlying congestion control mechanism of TCP, and hence have an impact on the ability of flows to gain a fair share of available bandwidth. A sender-side modification to TCP based on using an adaptive packet size will be presented with the objective of improving TCP's ability to fairly allocate bandwidth. Also, the concept of fairness among competing flows will

be looked at to determine what fairness should mean, and how to measure if it is being achieved.

1.1 TCP Service Model

The service model employed by TCP is intentionally designed to keep applications oblivious to how the data gets framed into packets for transmission on the network. From the applications point of view, a TCP connection is nothing more that a full duplex, point to point, byte stream. It knows nothing of how this byte stream will be segmented by the protocol. In contrast, the UDP protocol uses a datagram model instead of a byte stream. By using the byte stream model, the protocol has explicit freedom in how it decides to frame the data into packets. Intuitively, TCP tries to send the largest amount of data possible per packet, to minimize overhead from headers, and maximize throughput. This maximum amount is referred to as the maximum segment size, or mss. The mss is determined during the setup of the TCP connection. Typically, in most modern TCP stacks, Path MTU is used to determine what the maximum packet size is that will not cause IP fragmentation [7]. With the dominance of ethernet, this typically leads to a packet size of 1500 bytes.

1.2 Packet Size and Throughput

The main consideration of TCP's packet sizing strategy is to maximize the throughput achieved by the application. Packet size has an impact on throughput because of the overhead incurred from headers, as well as the effect it has on the aggressiveness of the TCP congestion control algorithm. Compared to some other protocols, the TCP header is relatively large. The combined TCP/IP header being at least 40 bytes in size. Figure 1.1 presents overhead from headers as a function of the total packet size. It is clear that once packet size dips to 250 bytes or less, the

overhead begins to be quite significant. Another important issue aside from consuming greater amounts of bandwidth due to headers, is the stress smaller packets put on the routers in the network. The algorithms used in routers for forwarding increase in complexity based on per-packet costs not per-byte. For similar reasons, and aside from overhead concerns, a TCP flow will be more competitive at gaining bandwidth when using larger packets. The queuing mechanism used in most routers is first-in-first-out and based on packets. No consideration is given to the size of the packet when dropping. So if we consider the proportion of bytes sent by a TCP flow, those using smaller packets will suffer a higher drop rate. This in turn causes the congestion control algorithm to converge on a lower average rate. Thus, traditionally implementations of TCP try to avoid transmitting smaller packets to prevent this phenomenon. Small packets are coalesced into larger ones whenever possible using Nagle's algorithm.

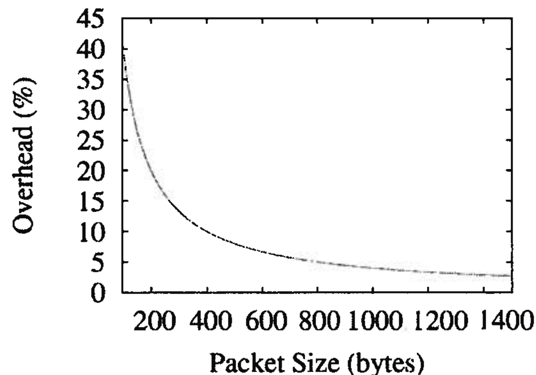


Figure 1.1: TCP/IP Header Overhead

Nagle's algorithm intentionally delays the transmission of packets to allow for smaller ones to be coalesced into one larger packet before they are transmitted [8]. This process, which is bound by a timer, will take place as long as TCP still has unacknowledged data in flight, or until a full *mss* worth of application data is waiting to be transmitted. Although the points presented here are valid concerns

when designing the strategy for packet sizing, there are conditions where using maximum sized packets can de detrimental to the fair sharing of bandwidth among competing flows.

1.3 TCP and Fair Sharing

The congestion control algorithm used in TCP is designed in a fashion to allow individual flows to gain a fair share of the available bandwidth without having to communicate explicitly with the other flows sharing the link. In an ideal scenario, each flow will gracefully adapt its rate of transmission either up or down, according to the traffic currently being experienced in the network. The next chapter will explore the congestion control mechanism in more detail. As computing technologies advance, resources such as storage and CPU power improve faster than bandwidth causing it to be the most contended resource on the Internet. Traffic measurement studies have shown the TCP is consistently the most used transport protocol for traffic on the Internet, estimating it to be used in upwards of 90% of all flows [6, 3]. With this being the case, TCP's congestion control algorithm has implicitly taken on the responsibility for allocating the scarcest resource on the Internet in fair manner.

Traffic on the Internet using TCP as the transport protocol can be classified into two categories of flows. The first category is short lived TCP flows. These flows are commonly referred to as mice, and account for a significant portion of the total number of flows on the Internet. The most obvious example for the source of mice on the Internet is world wide web sessions, whereby the TCP connection will only exist until the data for the desired web object is obtained. Although the mice make up a large percentage of actual flows on the network, they make up only a small percentage of the total number of bytes transferred. The second category of TCP flows are those which persist over a much longer period of time than the mice. These are referred to as elephants. Elephants constitute a large fraction of the total number of bytes transferred on the Internet, but make up a small percentage of the

total flows active [6]. As the main consumers of bandwidth, the sharing behavior they exhibit becomes quite important to the overall fairness. An example is peer to peer file sharing applications. These applications transfer large amounts of media based files, which are typically quite large in size relative to that of a web page. These flows use TCP to transfer the data, and make up a huge percentage of the number of elephants on the Internet [10]. It is anticipated that more elephant style traffic will arise from other applications such as video streaming, which we expect to use TCP, or some TCP-friendly congestion control. TCP's dominance in the area of transport protocols on the Internet is empirical evidence that it does an acceptable job of preventing congestion collapse on the network. However, it has been studied and documented that TCP does a poor job of sharing bandwidth fairly among competing flows [5, 1, 12, 4].

Qualitatively, fairness is associated with the rate achieved by individual flows sharing a common bottleneck link. The rate achieved by a flow using TCP is dependent on the algorithms used in the congestion control algorithm. These include slow start, congestion avoidance, fast retransmit and recovery, and timeout based retransmissions. These algorithms will be covered in more detail in the next chapter. For an elephant style flow, the ideal scenario would take place in the following manner. The flow begins in slow start allowing it to discover the available bandwidth quickly. It then would enter into the congestion avoidance mode where it will exhibit the classic sawtooth transmission pattern. This allows the flow to fine tune its transmission rate over time. In the congestion avoidance mode, the most important variable that controls the rate at which data is sent is *cwnd* (congestion window), which limits the amount of data that can be sent by TCP in one round trip. This can be expressed in terms of bytes or packets, but for now assume it is expressed in packets. The instantaneous sending rate of a TCP flow is then governed by the following equation:

$$bw = cwnd \times \frac{mss}{rtt} \tag{1.1}$$

This equation has been extended into models that characterize TCP friendliness. The equation given by Mathis et al. is an example:

$$bw = \frac{mss}{rtt} \times \frac{C}{\sqrt{p}} \tag{1.2}$$

In this equation p is the drop probability, and C is the constant of proportionality, which is derived from the rate at which the congestion window increases and decreases, as well as certain other details of the congestion control algorithm (the value of C is usually slightly higher than 1). Equation 1.2 reveals the basis of TCP sharing. Importantly, is shows that n flows sharing bandwidth on a link should receive approximately $\frac{1}{n}$ of the available bandwidth. This is due to the fact that they are expected to have equal values for mss, rtt, and p.

1.4 Large Packets Considered Harmful

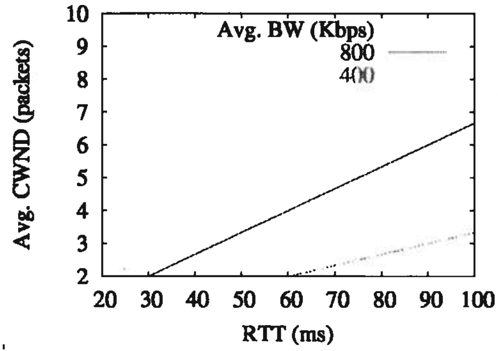
It is important to note that conditions may arise where using mss sized packets inhibits TCP from achieving fair sharing of bandwidth. TCP starts to degrade with regard to fairness when the per-flow fair share of bandwidth corresponds to a low bandwidth-delay product. In the TCP congestion control algorithm, there exists threshold values of $cwnd$ where TCP is unable to maintain congestion avoidance mode, and possibly descends into the undesirable situation of recurring transmission timeouts on packets. These thresholds occur when the congestion window is in the low single digits. If the flow starts experiencing recurring timeouts, its bandwidth will suffer severely, and thus overall fairness is sacrificed.

From equation 1.1, a fixed value of mss leaves the following relationship: $cwnd \sim bw \times rtt$. Figure 1.2 considers the average $cwnd$ values corresponding to different per-flow fair shares of bandwidth and packet sizes. Each sub-figure

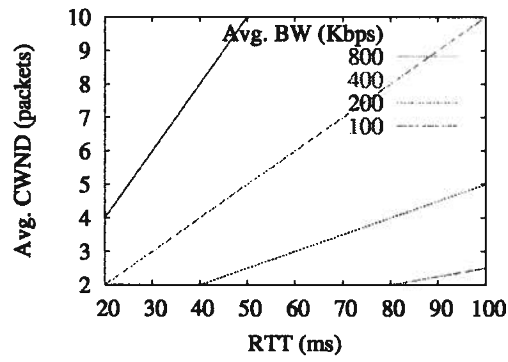
considers several average bandwidth levels and a single packet size. It is shown across a range of round-trip time values that should be a reasonable representation of what broadband users will experience. Starting with Figure 1.2(a), we look at 1500 byte packets. As mentioned above this is typically the largest *mss* that is experienced due to the use of PathMTU and the universal presence of ethernet [6]. Figure 1.2(a) shows that even if the fair share per flow is as high as hundreds of Kbps then *cwnd* values are still quite low, so we speculate that TCP will be unable to share the bandwidth fairly. Figure 1.2(b) looks at the same range of RTTs using smaller 500 byte packets. From the *cwnd* values we see that fairness may breakdown at a point where the fair share per flow is still in the hundreds of Kbps. In Figure 1.2(c), we see that for the same fair share per flow bandwidth, the *cwnd* values are much higher than in the 1500 byte case of Figure 1.2(a). Therefore, using a smaller packet size could extend the zone in which TCP does a reasonable job of attaining a fair share for each flow. It is expected that there will exist scenarios where sustaining fairness will outweigh the negative impact of having higher header overhead from using smaller packet sizes. In this case packet sizes could be naively fixed at lower values. However, the approach would needlessly impose overhead when it is unnecessary. Therefore we propose the more useful approach of adapting the packet size based on the level of congestion being observed.

1.5 Thesis Outline

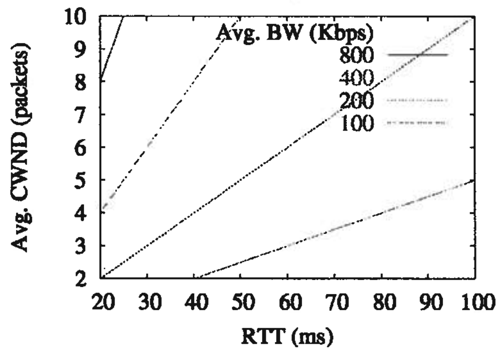
The rest of the thesis will unfold as follows. Chapter 2 will give an overview of TCP and the services it provides as a transport protocol. Chapter 3 will present related work that has been published in the literature. Chapter 4 will give a description of the experimental design used to validate and test our hypothesis. Chapter 5 will present and analyze the results obtained, and finally Chapter 6 will present conclusions and future work to be done.



(a) 1500 byte packets



(b) 500 byte packets



(c) 250 byte packets

Figure 1.2: Average CWND vs RTT

Chapter 2

Overview of TCP

TCP offers a number of useful services to applications that choose it as a transport protocol. In this chapter will will provide an overview of these services focusing mostly on the operations of the TCP congestion control algorithm, as it is this service which is most influenced by the packet sizing strategy used. Readers whom are familiar with the basic operations of TCP may wish to skip this chapter.

2.1 Basic TCP

TCP is designed to provide a completely reliable, full duplex communication between a pair of processes across an unreliable data network. A TCP connection is a logical end-to-end concept defined by a 4-tuple of information. The tuple consists of the sender's IP address and port number, along with the receiver's IP address and port number. This tuple uniquely distinguishes separate TCP flows from one another. A TCP connection is said to be connection oriented due to the fact that before two connecting proceses can send data to one another they must first conduct what is known as the "3-Way-Handshake", during which segments are sent to negotiate and initialize the parameters for the session about to take place. It is during this handshake that the *mss* is first established.

2.1.1 TCP Header

Each TCP segment carries with it a header of information, containing a minimum of 20 bytes of data. The header contains the following information. The first 4 bytes specify the source and destination port numbers, the next 4 bytes contain the sequence number of the packet. The sequence number is used to maintain the order of data and aid in the scheme used to ensure reliable delivery of data. The next 2 bytes specify the header length (it may be longer the 20 bytes), as well as 6 bits used to set varying flags that may be that may be needed. Following this are 2 bytes containing the receiver's current window of data (used for flow control). Finally, the last 4 bytes contain a checksum of the data, and pointer to urgent data in the packet (seldom used). There is also an allowance of more header data for additional options, however, typically the header remains at 20 bytes. Figure 2.1 gives a graphical display of the header information.

2.1.2 Reliable Delivery

The following is a brief description of the mechanism employed by TCP to ensure reliable delivery of data. TCP uses an sequence number/ack based system for keeping track of the order of packets as well as detecting lost packets in the network. TCP views the data as an ordered sequecne of bytes, and this is reflected in the way sequence numbers are chosen for segments. The segments are labled with sequence numbers based on the byte numbering in the stream (first byte would be 0 etc). When data is received the ack packet will contain the sequence number of the next byte expected, and by doing so gives a cumulative acknowledgement of data up to that point in the stream. There is no specification on how receivers should handle out of order data, but many implementations will use some sort of buffering scheme to prevent unnecessary retransmission of data.

Along with sequence numbers, a timer is used to set a limit on the amount of time spent waiting for an ack of some data. The length of time used is calculated us-

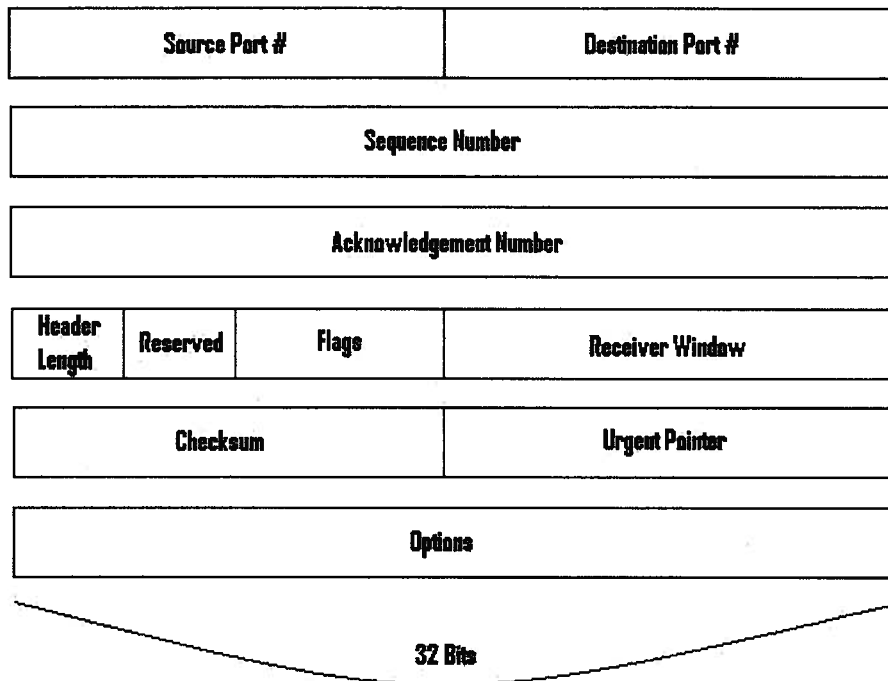


Figure 2.1: TCP Header Structure

ing an estimated value of the rtt combined with an estimated value for the deviation experienced. When the timer expires, data is retransmitted starting at the sequence number of the timed-out packet. An important feature of the timer is its behavior after a timeout takes place. When the timer expires, data is retransmitted and a new timer is started. However, instead of using the estimated RTT values in setting the new timer, the old time used is simply doubled. This process continues as more timeouts occur in the system, causing the timeout period to increase exponentially. We believe once a flow gets into this mode of transmission (recurring timeouts), its ability to fairly compete for bandwidth suffers dramatically. More discussion on this follows in future sections.

TCP does provide a mechanism to avoid waiting for a timeout event to occur before retransmitting data. This is referred to as fast-retransmit and functions as

follows. Packet loss in the network can be inferred by the reception of duplicate acknowledgement (dup acks). A dup ack refers to an acknowledgement for data that has already been acknowledged by a previous packet. This tells us the following. In order to send out a dup ack, the receiver must have received data that is out of order from what was expected. This would cause an ack to be sent for the next expected packet, resulting in a duplicate ack being received for this packet. Fast retransmit occurs if 3 duplicate acks are detected. In this case, data from the last acknowledged packet forward in the stream will be immediately resent, instead of having to wait for the timer to expire.

Together, the timer and the sequence number/ack system provide a completely reliable delivery of data on top of the unreliable service provided by IP.

2.1.3 Flow Control

Another important service provided by TCP is that of flow control. When a TCP connection get initialized a buffer is allocated on both hosts for receiving data. Once data arrives in order to the TCP connection, it is placed in this buffer where it can be read from by the application using the connection. However, this may not happen instantaneously, and therefore the buffer may begin to fill up with data. If the connection has no where to copy data to, then clearly we have a problem and some data my be lost due to the buffer overflowing. To prevent this TCP provides a variable which allows the receiver to advertise how much space is remaining in the receive buffer, which therefore allows the sender to throttle its sending rate if the buffer is full. Figure 2.1 shows the space in the header reserved for advertising this window. This feature should not be confused with congestion control as it is a totally different phenomenon which occurs at the end host, whereas congestion is an event occurring with the network internals.

2.2 TCP Congestion Control

One of the more important features of TCP is the congestion control algorithm. TCP uses an end-to-end approach to congestion control, meaning the end hosts receive no explicit feedback about congestion occurring in the network. The basics of the idea are simple. The host determines the rate at which it will transmit data onto the network as a function of the amount of congestion that is being experienced in the network. So when little congestion is being experienced then the rate goes up, and as congestion increases, the rate goes down.

To control the rate at which data is transmitted onto the network, TCP maintains a block of data which is currently allowed to be transmitted, called the congestion window. This variable is commonly denoted *cwnd*. The congestion window imposes a constraint on the rate at which TCP can send data. It only allows x number of unacknowledged bytes to be in the window at any time, where x is changing based on the level of inferred congestion. This works in tandem with the receiver window variable mentioned above. If the receive window is smaller than *cwnd* than it becomes the constraining variable for the rate of data sent. For the discussion of congestion control, assume the receiver's buffer is significantly larger than the congestion window.

The rest of the congestion control algorithm can be described in terms of how and why the congestion window gets manipulated. First of all we must describe how TCP can infer there is congestion occurring on the network. When a packet is determined to be lost then TCP can take this as a sign of congestion in the network (not many loses will occur due to other phenomenon). TCP knows that a loss has taken place based on two events. When a timeout event is triggered or three duplicate acks arrive, then a packet loss has been assumed to have occurred. Imagine, the packet's journey through the network until the point it gets lost. For the most part it will only gets lost when it arrives at a router and the router has no room left in its buffer for another packet. The packet thus gets dropped. Since the

router has a full buffer, this implies that there is some degree of congestion occurring at that point in the network. TCP thus should act accordingly to a packet loss to try and remedy the congestion situation. We can now look at the major components of the TCP congestion control algorithm.

2.2.1 Additive Increase/Multiplicative Decrease (AIMD)

Additive increase, multiplicative decrease refers to the level of aggression used when increasing and decreasing the congestion window. The intuition is to conservatively increase your window when no congestion is being perceived as there may be available unused bandwidth in the network. On the other side of the coin, we want to aggresively decrease our window in the presence of congestion to remedy the situation as soon as possible. Since most flows in the network will be expected to be using TCP, and experiencing losses at the same time, we are then assured all flows will back off and relieve the congestion (there could be fear of a non compliant flow gaining an advantage). In practice the multiplicative decrease is implemented by cutting the congestion window in half when congestion is detected. However, the window will never drop below 1 mss. On the increasing side of things, the congestion window is increased by 1 mss per RTT. This causes the desired additive increase in the window while no congestion is present. This pattern of linearly increasing the window by 1 and then cutting in half when congestion occurs gives rise to the well known sawtooth pattern demonstrated by the sending rate of long lived TCP flows. Figure 2.2 shows the sawtooth pattern. It should be noted that this phase of TCP congestion control is known as congestion avoidance mode. It has been shown that TCP is capable of fairly distributing bandwidth as long as the flow can stay in this mode.

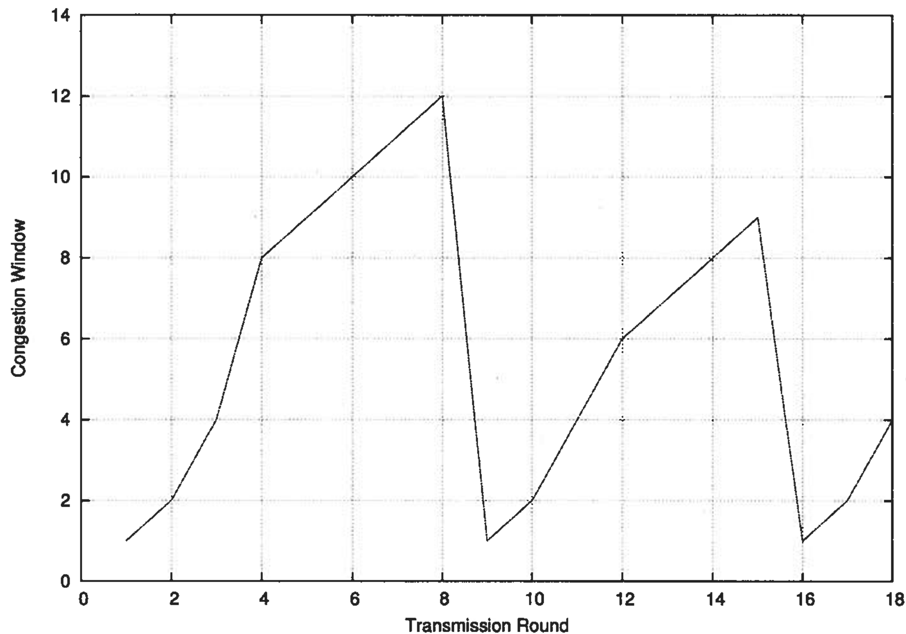


Figure 2.2: Congestion Avoidance Sawtooth Pattern

2.2.2 Slow Start

The term slow start is a bit ironic, as it is the point in the algorithm where the congestion window is increasing at its most rapid rate. When a TCP connection is initialized, and also after a timeout event occurs, the congestion window is set to 1 mss. This begins the flow at a very conservative rate, and thus may be vastly underutilizing the available bandwidth. If the flow went into congestion avoidance mode at this point, it may take an unreasonable amount of time to reach the capacity of the network, as we would be only increasing the window my 1 mss per round trip. To alleviate this problem, the flow begins in slow start mode. During this mode the congestion window is increased exponentially. The window doubles every RTT until a loss event occurs. It is important to distinguish between the behavior that takes place when a timeout occurs versus when three duplicate acks arrive. For the case of three dup acks, AIMD takes over. The window is cut in half and linear increase

takes place from that point on until another loss occurs. However, as mentioned above, when a timeout takes place, the window is reset to a value of 1 *mss* and the slow start process takes over. TCP adds an additional variable to the algorithm called the slow start threshold. This variable, in essence, puts a cap on the slow start phase. Once the congestion window grows to a size equal to or greater than the threshold, slow start ceases and congestion avoidance takes over. When the flow begins, the threshold is set to an arbitrarily high value, and when the first loss takes place, threshold is set to one half of the value of *cwnd* when the loss occurred. So an overall picture of a TCP flow's sending rate would be an exponential increase (slow start), followed by a linear increase once the threshold is broken.

2.3 Chapter Summary

In this chapter we gave a review of the goals and motivation of the TCP algorithm. We gave a brief summary of how TCP provides reliable, in order delivery of data to the application, as well as a brief review of the flow control mechanism used to avoid overloading the end host with data. We focused a bit deeper on the congestion control algorithm employed by TCP, with a description of the additive increase, multiplicative decrease paradigm. Finally, we presented the slow start phase of TCP, used to start a flow and to rapidly regain an appropriate sending rate after data loss has occured.

Chapter 3

Literature Review

In this chapter we will present a review of the work that has been published in the literature with regard to using adaptive packet sizes. The area of TCP congestion control has been saturated with research since TCP's original conception 20 years ago. However, it is surprising to find that very little work has been done investigating the packet sizing strategy used to map the byte stream onto packets. Here is a review of two main papers that present a similar approach to our idea of using a variable packet sizing strategy.

3.1 LTTCP and Variable Packet Size

Tickoo et al. [9] attempt to address the problem of degrading TCP performance over wireless links when there is the presence of a packet error rate with a value in the range of 1% - 5%. They propose a scheme which involves reactive and proactive use of forward error correcting packets, and an adaptive packet size to ensure a minimum number of packets exist in the congestion window at any given time. They identify that the most significant problem encountered when using TCP over wireless is the timeouts occurring due to packet erasures. It is therefore desirable to minimize the number of timeouts that occur. It is observed that if all packets in the congestion window are lost, then a timeout will occur. By refining the granularity

of the packet size, the congestion window will contain more packets, and thus there is less chance of all packets in the window being lost. Secondly, if some packets are lost there is also a chance of timeout. If fast-retransmit is not triggered by 3 duplicate acks arriving then the timeout will happen. This is once again mitigated by a finer granularity in the congestion window. By making the packet size smaller, there is more chance some packets will make it through and generate dup acks, to prevent the timeout from occurring. The authors note that by reducing packet size, overhead is increased. This overhead is justified by the gain made in the range which TCP maintains a high level of performance. Also, since the packet size adapts based on the degree of packet erasures, the overhead is only ever significant when packet erasures are at their worst.

3.2 TFRC and Variable Packet Size

The work by Widmer et al. [13] approach the idea of a variable packet size in terms of the TFRC protocol. TFRC stands for TCP-Friendly congestion control and is a version of equation based congestion control. Equation based schemes explicitly calculate the sending rate desired based on a given equation defined in the protocol. The work investigates the effects of using a smaller than standard (mss) sized packet on the overall throughput achieved by a flow using TFRC, and the results this has on fairness in a bandwidth limited network. To appreciate the dynamics of how the packet size influences throughput and fairness in TFRC, we must first look at the equation governing the transmission rate. This is given in equation 3.1.

$$R_{TCP} = \frac{S}{r(\sqrt{\frac{2l}{3}} + (12\sqrt{\frac{3l}{8}}) \times l(1 + 32l^2))} \tag{3.1}$$

In this equation R_{TCP} represents the expected throughput for a TCP flow experiencing like conditions on the network. It is calculated as a function of the the round trip time r, the loss event rate l, and the packet size S. For a more detailed

analysis of equation based congestion control refer to [2]. From this equation the rate used by a TFRC flow is linearly dependent on the packet size. Clearly, if we have a flow using a smaller packet size, s, than its competing TCP flows using S, the throughput will suffer significantly as a result. This can be remedied by using S in the calculation instead of s. Intuitively, this should achieve fairness for all flows independent of the size of packet being used. In practice this fails. A problem arises due to the way TCP modifies its congestion window. TCP will only halve its window in response to one loss event per round trip time. This has the effect of creating a window of time where all other packet losses are essentially ignored. This creates a bias in favor flows that use TFRC with small packet size, where packet size is represented by S instead of s. To transmit at a rate proportional to S with s sized packets, the packet rate must increase. This creates more packets in the window per round trip time, and thus more packets that could potentially be dropped and then ignored due to the above phenomenon. [2] shows that the resulting consequence is the ability of the small packet flow to completely overwhelm the others in a bandwidth bottlenecked network. The work presents potential solutions and evaluations which are orthogonal to our discussion on variable packet size.

Chapter 4

Experimental Design and Analysis of Variable Packet Size in TCP

We now focus our attention onto the experimental investigation of the use of variable packet sizes in TCP. In this chapter we will describe in detail the design of our experiments, and the experiences we encountered along the journey of investigating.

4.1 Experimental Design

The experiments were designed in a three phase investigation. Phase one consisted of running experiments in a simulated network environment. Simulation software has been used widely to design and test new network protocols. For our experiments we chose the NS-2 software. NS-2 is described further below. For phase two, we ran experiments in an emulated environment, using the tools available from Emulab. Emulation allows a controlled real world test. Emulab is described below. Phase three involved the implementation of our algorithm into the Linux kernel.

For the experiments the following topology was set up. We use the classic

dumbbell topology, where two sets of nodes are connected to each other by a bottleneck link. On each end of the bottleneck are n nodes which will each launch a TCP flow over the bottleneck. Each node on one side of the link is connected to the bottleneck router by a 100MB connection, and configured as a LAN so latency over the link from the router to the end node is negligible. The bottleneck link is a 25MB link with 20ms of latency, making for a 40ms RTT. Each router implements a standard drop tail queue, with a packet limit of 125 packets. The TCP flows are configured with a maximum recv window of 500 packets. This topology is shown in figure 4.1.

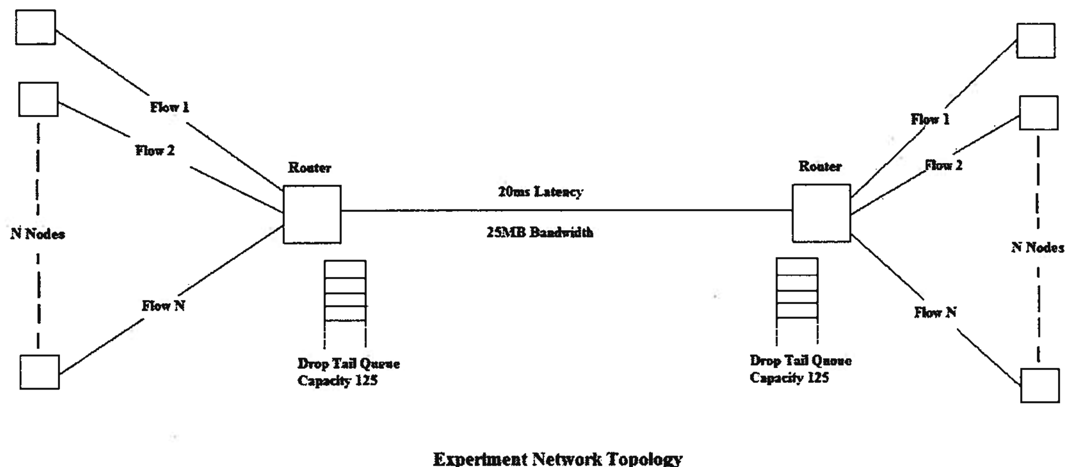


Figure 4.1: Network Topology for Experiments

4.2 Fairness

Before discussing in further detail the design and results of the experiments, we must first divert our attention to an important concept. We have been using the term fairness quite loosely thus far, and before continuing it is important to put a more quantitative definition to the word. Described above, we mention that for a flow to

be considered to be receiving a fair share of available bandwidth, it should receive $\frac{1}{n} \times bw$, where n is the number of flows sharing the link and bw is the available bandwidth. This analysis is correct, however the situation gets more complicated when we want to determine if the overall fairness being achieved on a bottleneck link is satisfactory. In practice most flows will not get exactly $\frac{1}{n}$ of the bandwidth, so how do we assign a value to represent how close or far away each flow is to its desired fair share. We addressed this problem by looking at three different metrics to evaluating the overall level of fairness on a link.

4.2.1 The Max/Min Metric

The Max/Min metric attempted to assign a fairness value to a set of flows sharing a link. If we have a set of flows, N, of size n, that are all sharing a link then we define the Max/Min equivalence ratio as follows:

$$er = \frac{min_{i \in n}(avg_bw_i)}{max_{i \in n}(avg_bw_i)} \tag{4.1}$$

Note that this is an extended version of the two flow Equivalence Ratio used by Floyd et al. in [2]. This measure appeared to have promise but we quickly discovered a flaw in its design. By using the minimum and maximum we are leaving ourselves extremely vulnerable to outliers in the data. For instance, if only one flow out of 1000 or even 10,000 ends up receiving 0 bandwidth, then our Equivalence Ratio would be 0. This hardly seems appropriate as a measure of the overall fairness achieved on the link. It tells us very little information about what the other flows may be attaining. Basically, it was compensating for the poor performance of only one flow. It is this feature that led us to abandon the Equivalence Ratio as our metric of choice.

4.2.2 The Jain Index

The second metric we looked at was Jain's fairness index, which had been designed and used in practice to measure fairness. The Jain index is calculated as follows.

$$f(x_1, ...x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} \tag{4.2}$$

The index assigns a value between 0 and 1 to evaluate the fairness achieved over a set flows. If all flows get their fair share then the value will be 1, and if all flows get no bandwidth then the value will be 0. The metric seemed to have promise as it has been used by other researchers to measure fairness in the past. We, however, discovered a discrepancy with regard to our work. After calculating a Jain index over different sets of data and then doing a more manual inspection of the data, we found the Jain index to do a poor job of illustrating when some flows are getting starved for bandwidth. We observed that a certain percentage of flows could receive 0 bandwidth and the index would still reflect a reasonably high number. We found this to be incompatible with our research goals. It was the exact opposite of the Max/Min problem, as the Jain seemed to greatly under compensate for flows that will get completely starved. This resulted in our abandoning the Jain as our metric, and also rethinking the nature of our research. We decided that perhaps it was not fairness in general that we were concerned with but, more specifically, it was that of bandwidth starvation of flows that mattered most. If an application such as video streaming gets no bandwidth for a period of time then it will completely die. This left us with the task of designing a new metric that could garner information about the level of fairness being achieved, but also illustrate the degree to which some flows may be completely starved for bandwidth.

4.2.3 Cumulative Distribution Function

After considering what features our metric needed to be sensitive to, we came up with a cumulative distribution function which would illustrate the level of fairness

being achieved across a group of flows without overcompensating or under compensating for starvation. Also introduced into the metric is the scale of time that the flows are competing against each for bandwidth. If we observer that flows share fairly over a long time scale, this doesn't imply that they will on short time scales. We introduced the notion of time windows into the analysis. Real time applications needing minimal latency between received packets will still suffer if they are not getting a fair share on short time scales, even if they achieve fairness on a larger window of time.

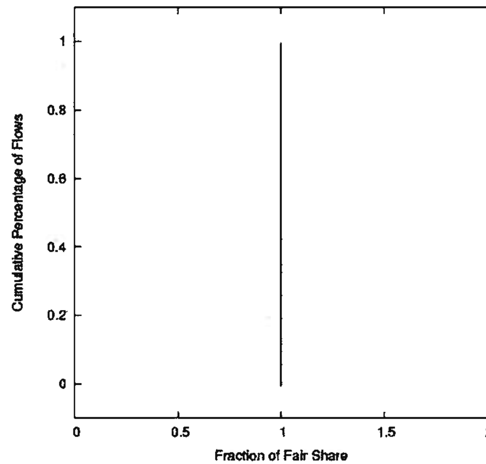


Figure 4.2: CDF of Fair Network

The CDF is computed on traced packet data as follows. For each window we compute how much data each flow got across the network. We then divide this by the fair share of bandwidth in the time window to get the percentage of fair share for each flow. This flow is then slotted into a bucket based on what percentage of fair share it achieved. The buckets are broken up on a granularity of 5%. After all windows are computed, the average is taken and graph data is generated to show the cumulative percentage of flows in each bucket. Our function the percentage of fair share to the percentage of flows that got at least that much bandwidth. For

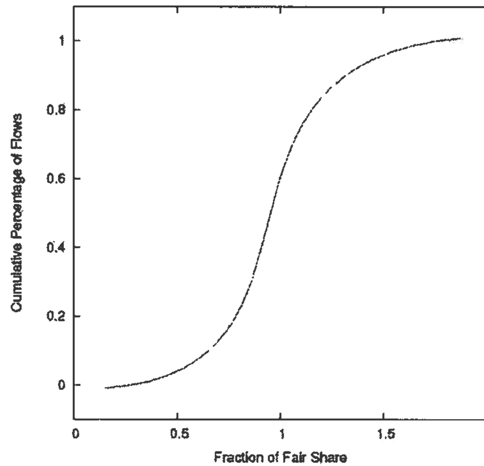


Figure 4.3: CDF of Unfair Network

example if the point (0.3, 0.5) is on the graph, this means that 50% of the flows got 30% fair share or less. As the percentage of fair share grows, the percentage of flows will approach 100%. A network with perfect sharing would thus have CDF graph of a horizontal line at x = 1 as in figure 4.2. This would be interpreted as all percentage of flows achieved 100% fair share. As sharing behavior degrades a CDF graph will start to have tails curling toward smaller percentages of fair share, as well as curling toward greater than fair share, as some flows suffering implies other using their bandwidth. This is shown in figure 4.3.

4.3 Phase One and the NS-2 Network Simulator

For phase one of our experiments worked with simulation and the NS-2 simulator was our software tool of choice. NS-2 is an event driven network simulator developed in a combination of C++ and OTCL, an object oriented version of the TCL scripting language. It provides support for simulation of TCP, wireless networks, and a variety of other network protocols. Simulations are defined in the OTCL script, where the user can create network topologies, as well as agents in the network such

as TCP/UPD transmitters, which can be attached to nodes in the topology. The software provides a mechanism to run TCP flows for a fixed amount of time, and tools are provided to gather packet trace data generated by the simulation which can then be analyzed accordingly.

4.3.1 Round One Experiments

The simulation portion of the experiments was broken up into three rounds of testing. Our goal in phase one is to validate our hypothesis that using a variable size packet in TCP congestion control will improve the performance of TCP in regard to sharing bandwidth among competing flows fairly.

For the first round of testing, we ran experiments using a homogeneous packet size among all flows in the simulation. Simulations were run using packets sizes of 1500, 1000, 500, and 250 bytes. For each packet size we also ran separate experiments, varying the number of flows competing for bandwidth. The number of flows used consisted of 50, 100, 150, 200, and 400 flows. The idea being to drive the simulation from a scenario of reasonable congestion to one of severe congestion. Also, the time window for which fairness was measured was varyied in the analysis of the data. We used windows of 0.1s, 1s, 10s, and 100s. Different applications may be more concerned with fairness in the smaller windows (eg. media streaming), and others may only care about their fairness being amortized across longer windows (eg. a large bittorrent download).

The results were encouraging, and demonstrated the ability for a smaller packet size to sustain fairness among flows where the standard packet size of 1500 bytes showed severe deterioration. Figures 4.4, 4.5, 4.6, and 4.7 show the CDF's of the collected data for 1 second windows. We see that as the number of flows grows, pushing the network into a more congested state, the 1500 byte flows begin to do poorly. With only 50 flows active in the network, we see that about 30% of flows are getting at most 50% of the fair share of bandwidth, where the smaller packets

achieve near perfect sharing of bandwidth. As we move into the severely congested range with 400 flows active on the network, we see that sharing has completely been destroyed when using 1500 bytes packets, with about 50% of flows getting completely starved for bandwidth. However, the 500 and 250 bytes packets are still achieving near perfect sharing.

Figures 4.8, 4.9, 4.10, and 4.11 show the CDF's for the 10s windows. With the longer windows, the 1500 byte flows improve their performance in the presence of mild congestion. We see with 50 and 100 flows present the curve is close to fair. However, once again the level of congestion rises we get poor fairness performance from the larger packet sizes. With 200 flows, 10% of the 1500 byte flows get starved, and with 400 flows, both the 1500 bytes and 1000 bytes packets show an unacceptable level of starvation with over 20% of flows getting zero bandwidth. The smaller size packets allow for a near ideal fair sharing of bandwidth, with the 500 byte packets only suffering slight degradation in the 400 flow case.

Overall, the round one experiments were quite promising. Our hypothesis of smaller packet sizes enabling a better sharing of bandwidth under levels of high congestion was demonstrated by the data collected from the NS-2 experiments.

4.3.2 Round Two Experiments

Having tested the performance of TCP with varying packet sizes in a homogeneous packet size environment, it was important to allow for competing flows to have different sizes. This was the goal of round two experiments. It was unknown how the smaller packets would compete with larger packets, and quite relevant to our research. If small packets suffered unreasonable performance loss when competing with larger packet flows, we cannot expect it to be a popular choice among applications. Small packets are guaranteed to suffer a minimum penalty of the ratio of large to small packet size being used. This is because routers are packet based and not byte oriented when forwarding and dropping packets. So for two flows to get fair

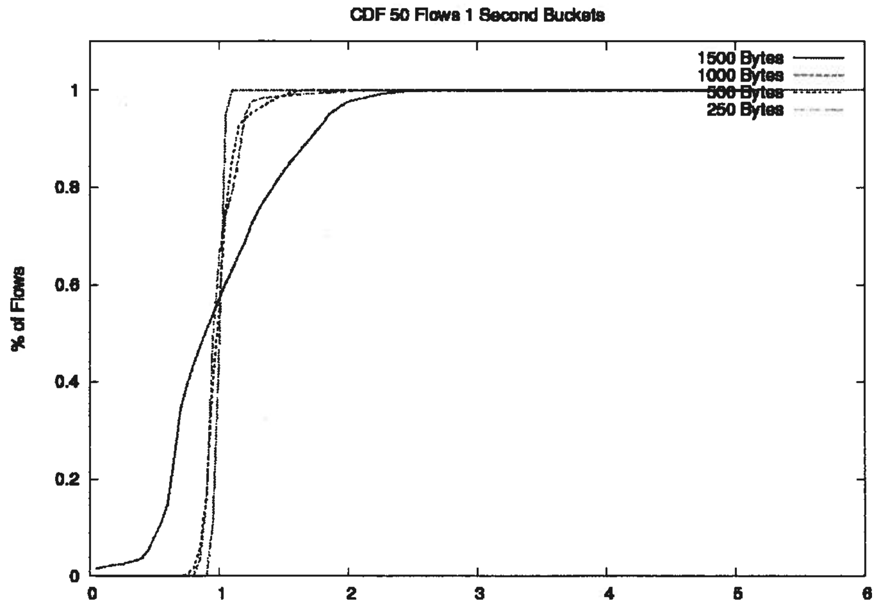


Figure 4.4: Round 1 - 50 Flows : 1 Second Bin

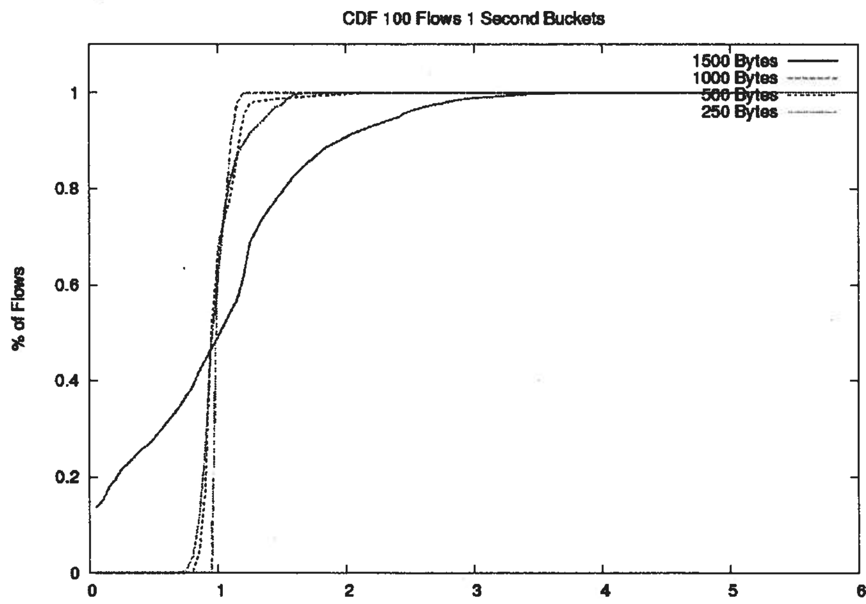


Figure 4.5: Round 1 - 100 Flows : 1 Second Bin

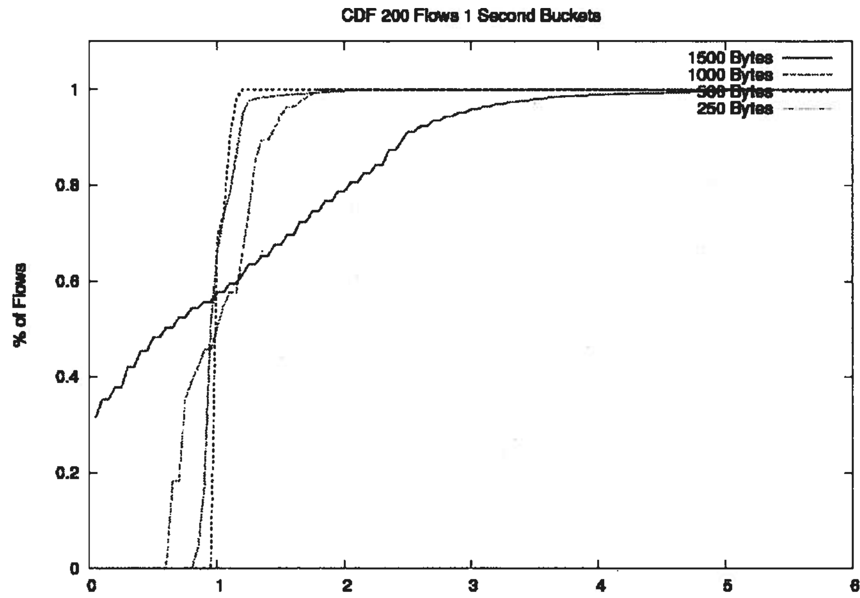


Figure 4.6: Round 1 - 200 Flows : 1 Second Bin

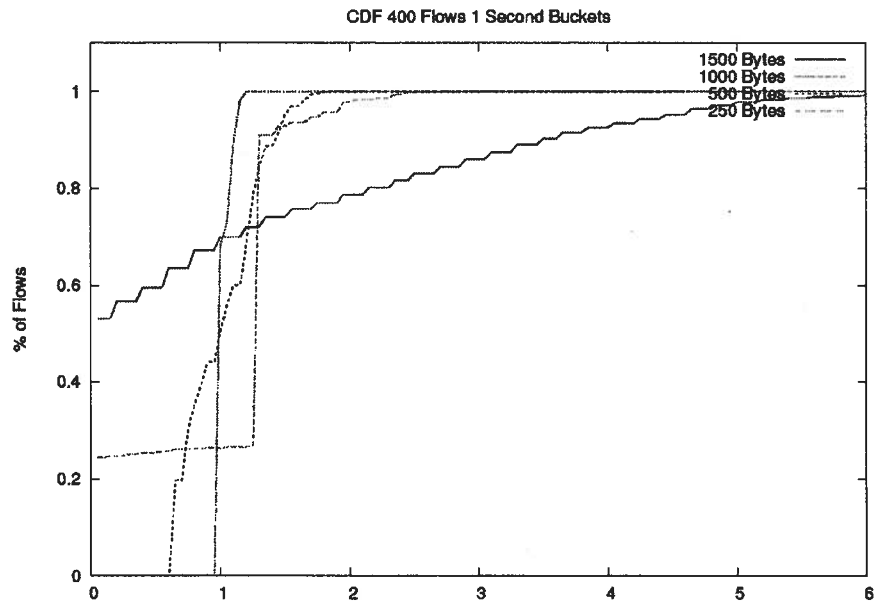


Figure 4.7: Round 1 - 400 Flows : 1 Second Bin

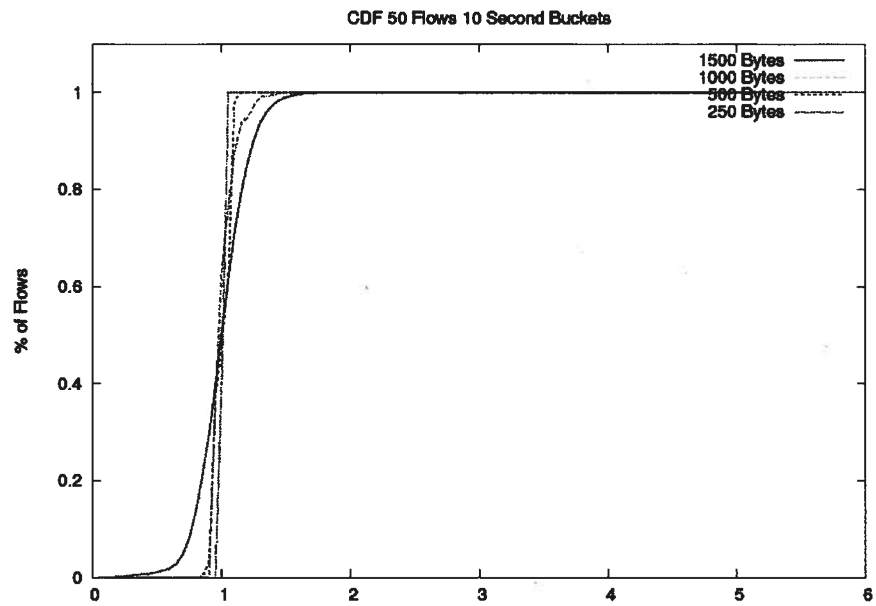


Figure 4.8: Round 1 - 50 Flows : 10 Second Bin

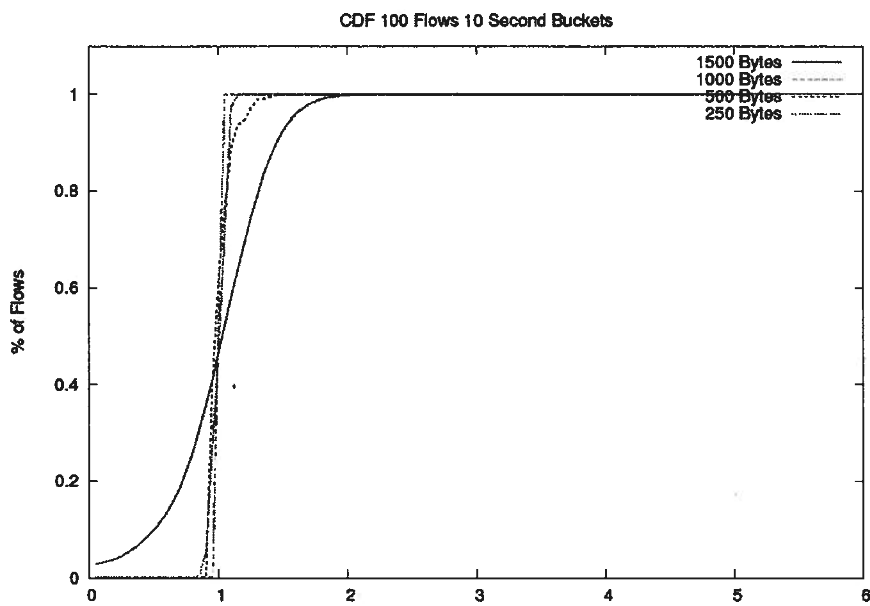


Figure 4.9: Round 1 - 100 Flows : 10 Second Bin

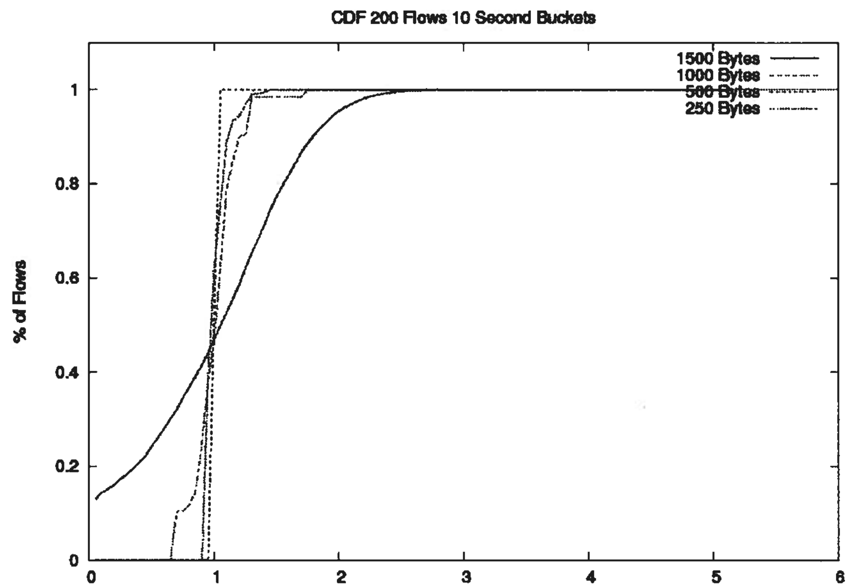


Figure 4.10: Round 1 - 200 Flows : 10 Second Bin

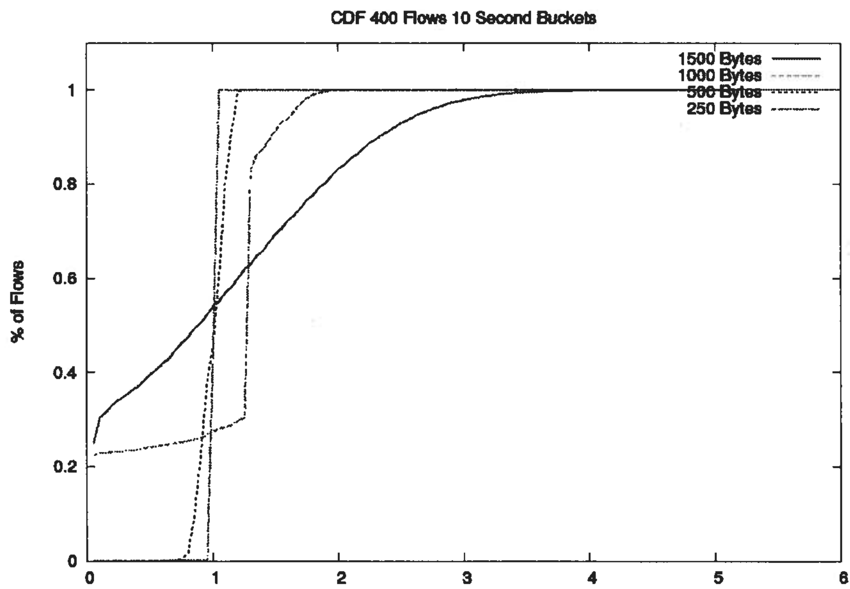


Figure 4.11: Round 1 - 400 Flows : 10 Second Bin

treatment from a router means that they got the same number of packets through. If one flow uses packets of size x and another uses size y, with $x > y$, then the flow using the larger packet will get x/y times more bandwidth through. This natural penalty was accounted for in our analysis, and we were interested to see if any further penalty was incurred. Experiments were run similar to the first round, with one new dimension added to the data. We would now divide the flows being run into two groups. One being large packets, for which we used at 1500 bytes packet size, and the other being small packet flows. For small packets, we ran experiments using 250 bytes packets and also 500 byte packets. There were three scenarios of flow groupings. One had 25% large, another 50% large, and the final 75% large packets. The remaining flows used the small packet size. The results from the simulations were quite discouraging. Figures 4.12, 4.13, 4.14 and 4.15, show a subset of the data gathered. Other experiments showed similar results. We can see that all flows seem to be suffering a large penalty. Even when using only 50 active flows, with 75% of them large, there is still a great deterioration in bandwidth sharing. We could not reconcile this data with our understanding of the TCP algorithm, and decided to abandon the NS-2 simulator as our means of a testing tool, feeling that perhaps the nature of our experiments had exceeded the abilities of the software to accurately model real world network behavior.

4.4 Phase Two and the Emulab Testbed

For phase two of our experiments we moved to an emulated an environment. After generating inconclusive results in the simulations, we were hopeful that emulation would prove fruitful. We used the Emulab network testbed to conduct our experiments [14]. Emulab is a cluster of computers set up by the University of Utah. The machines are loaded with software that allows for user to specify a network topology, and run controlled, repeatable experiments on the cluster. Users are given root access to the machines and define the topology using NS-2 scripts, which was con-

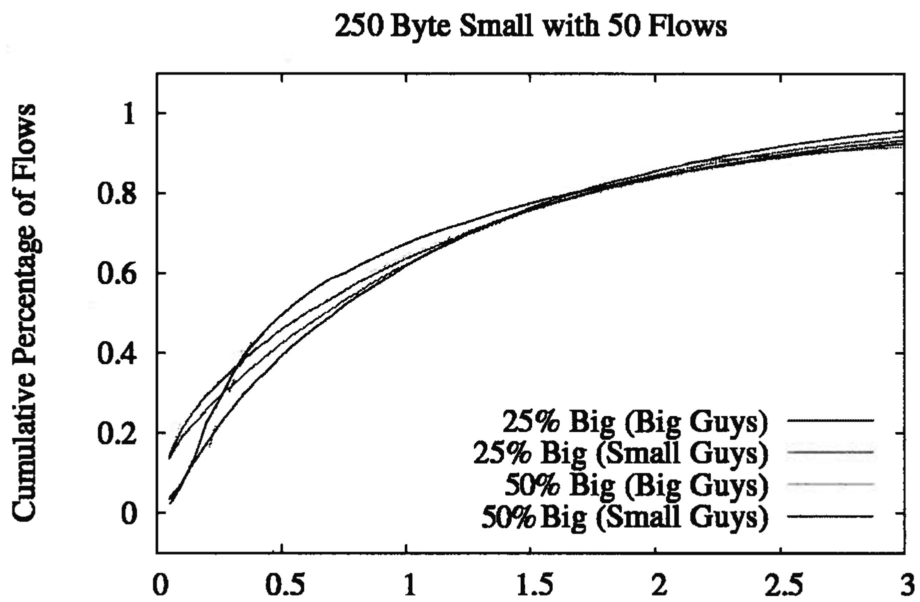


Figure 4.12: Round 2 - 50 Flows : 1 Second Bin

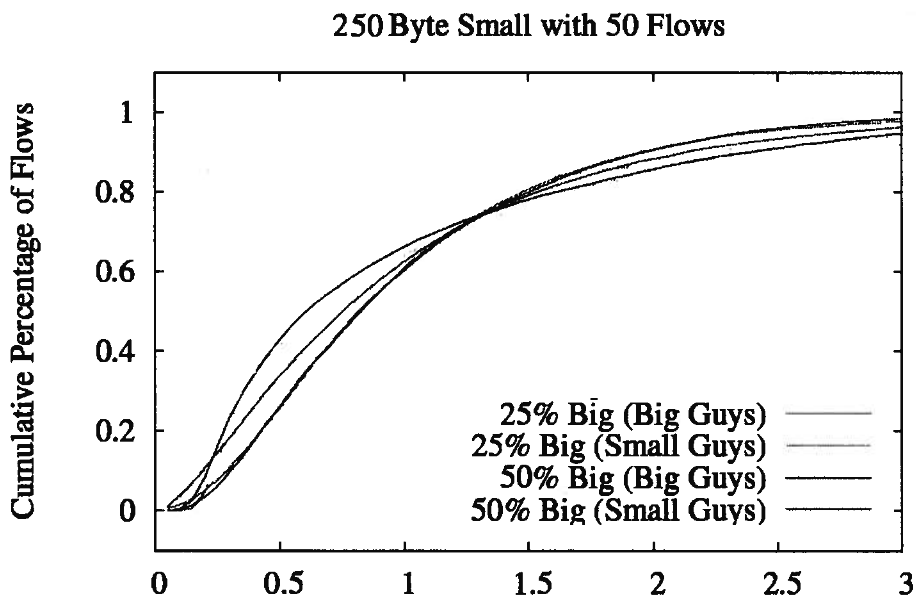


Figure 4.13: Round 2 - 50 Flows : 10 Second Bin

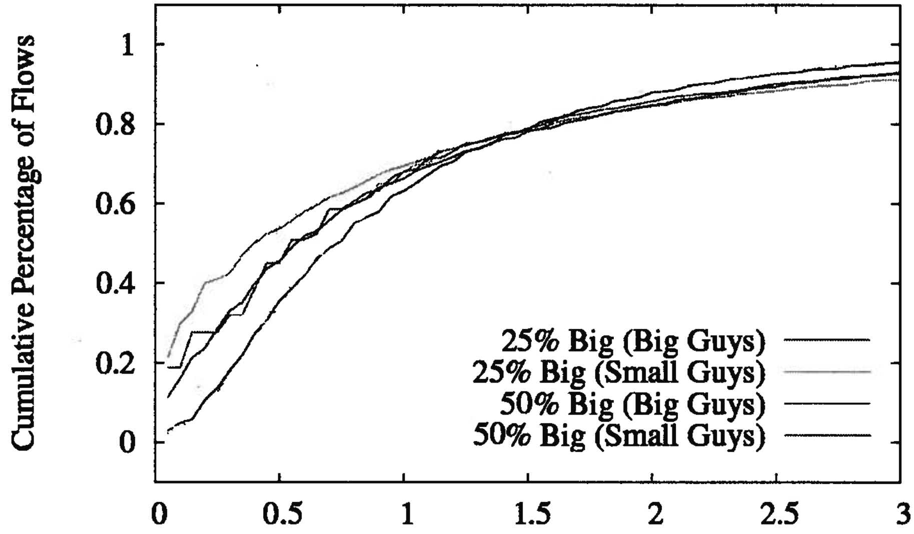


Figure 4.14: Round 2 - 200 Flows : 1 Second Bin

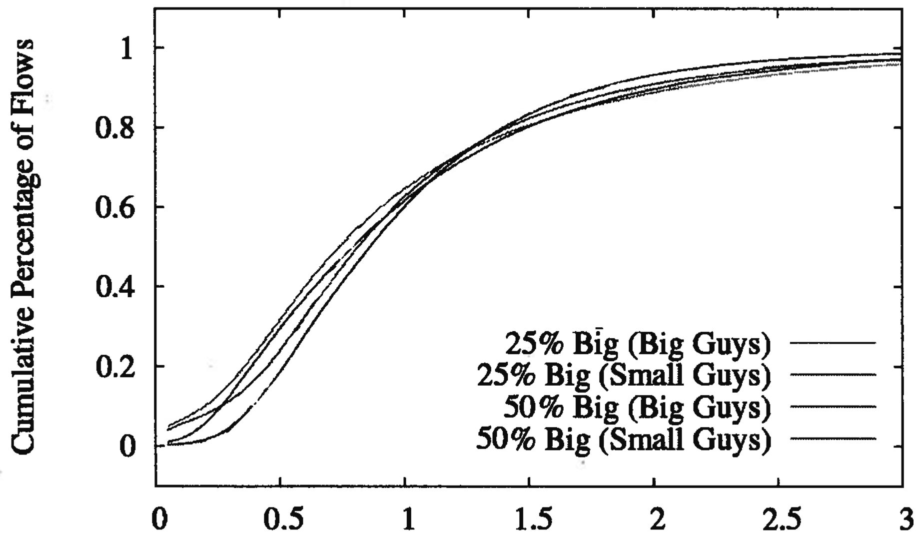


Figure 4.15: Round 2 - 200 Flows : 10 Second Bin

venient for our use. For our experiments we created the same topology as described in the simulations above. Linux Fedora Core 3 operating system was loading on the machines. To generate TCP flows on each machine the Mxtraf software was used. Mxtraf is capable of saturating a network with TCP flows, and provides the user with control over the number of flows running, and the size of the packets being used. To capture the data, a packet sniffer application was written using the libpcap C library. This allowed us to capture the data and conduct offline processing. We also ran a real time flow monitoring tool capable of inspecting TCP parameters of the flows running in real time. Experiments were run in the same fashion as round 2 of the simulations. Once again we encountered unexplainable problems with the data we collected. Using the real time flow monitoring tool, we observed that the TCP flows were capping the congestion window at a level much lower than the available bandwidth was dictating. No dropped packets were detected, so we would expect the flows grow the congestion window in accord with the TCP algorithm. However, this was not the case. With large numbers of flows running, we could no where near saturate the network due to this phenomenon. We examined all parameters in the experiment, including the queue sizes, the socket send and receive buffers, and could find no limiting variable. Quite frustrated, and unable to explain why the flows were behaving the way they were, the research was halted barring some new insight into our anomalous circumstances.

4.5 Post Research Results and Discussion

After the project had shut down and the results had been compiled some relevant research was published in the literature which bore some relevance to our discussion. In this section we will discuss the results obtained by Mellia et al. in [16] who used a similar approach to ours for designing a more resilient TCP. Also, we will present some work done by Petlund et al. in [15] which shows some of the vulnerabilities of TCP when using small packets which our experiments may have suffered from to

induce the anomalous behavior. Finally, we will discuss the role that active queuing approaches may play in the future of effective congestion control and their impact on the relevance of our discussion on TCP fairness in the current Internet architecture.

4.5.1 TCP Smart Framing

Marco Mellia et al. propose a modification to the TCP implementation which applies to the manner in which data is segmented in the early stages of the slow start phase of the TCP algorithm. The work is motivated by observation of the large number of short lived flows that live on the network that suffer unnecessarily when incurring packet loss while initiating a connection. It was observed that as a flow starts off, it begins in slow start phase with a congestion window of 1 packet of size mss. If this packet gets dropped the only way to detect is to timeout which is the most undesirable manner to detect congestion. This is quite a penalty for a short lived flow whose entire payload may only be a few segments on the network (eg. a simple http request). To help alleviate this issue, a scheme is proposed to break up the initial segment in the conegestion window into multiple segments (namely, at least 4). The idea being to allow the possibility for fast-retransmit/fast-recovery to take place. Since this is dependent on receiving three duplicate acks, there must be at least three segments received for each dropped segment in the network. This poses no greater bandwidth requirement on the network, other than the light overhead of the TCP header. Once the congestion window grows large enough, traditional sized segments may be restored. The work showed an improved performance with regard to the amount of time needed for web servers to complete data transfers, as well as a higher probability of triggering fast retransmit to detect data loss. Results also showed a more fair distribution of bandwidth among flows in highly congested networks. The work is parallel to our variable packet size, and does provide some positive indication that our premise was sound.

4.5.2 TCP and Thin Streams

Petlund et al. researched the performance of TCP when using a thin stream. A thin stream is defined as a TCP flow that uses small packets well below the *mss*, as well as having high interarrival times that prevent fast retransmission from being effective. The result is that the main cause of retransmission is timeouts which result in exponential backoff. Thus, the flows suffer greatly when loss occurs, even though they are not contributing a heavy load on the network.

A large premise of this work is that packets have a high interarrival time. The software we were using in our emulation was sending small packet flows and attempting to saturate the network, which is not in line with the conditions of a thin stream. However, we felt this work worth mentioning, as an explanation for the behavior observed is not available.

4.5.3 Active Queuing and Implications

Much work has been done on developing a more intelligent queuing mechanism than drop tail queuing. Currently, drop tail queuing dominates the Internet. However, as the demand for bandwidth rises, new schemes become more likely to come into application since drop tail performance is starting to show deterioration. Different algorithms have been proposed with the most popular being RED (random early detection) [17], and BLUE [18]. The approach used by active queues is to probabilistically drop packets based on the algorithm used. Drop tail queues penalize bursty flows (a large number of packets at once fill the queue and cause drops) and also cause TCP synchronization. This is a phenomenon where all flows on the network experience simultaneous congestion, run the TCP backoff algorithm, and then burst data all in the same period causing a continued period of congestion. By probabilistically dropping packets these hits can be avoided. RED monitors the queue size and drops packets with higher probability as the queue approaches full, where a full queue has a probability of 1. This implies the queue never fills up

which is slightly counter intuitive since we are trying to maximize network performance. RED correlates the quantity of data a single flow sends with the probability of dropping the packet, so fairness among flows is a strong consideration. One of the biggest problems with RED is that it requires a careful manual tuning of probabilities to provide good performance. BLUE was introduced to alleviate the pain of manual tuning of parameters. It functions in a manner so that the parameters will converge on optimal values given the assumption that the mix of traffic stays relatively constant.

Active queuing bares relevance to our discussing in that we have not explored the performance of TCP in highly congested conditions while functioning under a network with active queuing deployed. It is unknown how fair these algorithms will allocate bandwidth among competing flows, and is worth investigating. If TCP still shows poor behavior in these scenarios, work should be done to evaluate how variable packet sizes behave with these queuing schemes. Instinctively one would expect the smaller packets to get a greater share of bandwidth, as the active queuing schemes take the number of bytes a flow has transmitted into the equation. However, this is just speculation, and empirical evidence is needed.

Chapter 5

Conclusions and Future Work

The experiments conducted validated our hypothesis that larger packet sizes on the Internet suffer from poor sharing behavior when conditions of congestion progressively worsen. Also shown, smaller packets in a homogeneous packet size environment can allow for fair sharing of bandwidth to occur into severely congested periods on the network. However, when extending the experiments into a mixed packet size environment, we unfortunately could not draw any decisive conclusion on the performance of small vs large packets. The idea of adapting the packet size as congestion increases remains promising and largely unexplored. Future work could investigate deeper into why the TCP flows in the Emulab experiments were not growing the congestion window as defined by definition of the algorithm. Given this resolution, more decisive conclusions could be gathered from experimental data shedding light onto the feasibility of mixing large and small packet flows on a network. If experiments showed promise, and implementation in the Linux kernel could be undertaken. After which real world experiments could be conducted with small packets.

Bibliography

[1] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, August 1999.

[2] Sally Floyd, Mark Handley, Jitendra Padhye, and Jorg Widmer. Equation-based congestion control for unicast applications. In *SIGCOMM*, pages 43–56, 2000.

[3] C. Fraleigh, S. Moon, C. Diot, B. Lyles, and F. Tobagi. Packet-Level Traffic Measurements from a Tier-1 IP Backbone. Technical Report TR01-ATL-110101, Sprint ATL, November 2001.

[4] Luigi A. Grieco and Saverio Mascolo. Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control. *Computer Communication Review*, 34(2):25–38, 2004.

[5] Matthew Mathis, Jeffrey Semke, and Jamshid Mahdavi. The Macrosopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM Computer Communication Review*, 27(3):994–1010, July 1997.

[6] S. McCreary and K. Claffy. Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange.

[7] J. C. Mogul and S. E. Deering. Path MTU discovery. RFC 1191, Internet Engineering Task Force, November 1990.

[8] J. Nagle. Congestion control in IP/TCP internetworks. RFC 896, Internet Engineering Task Force, January 1984.

[9] Shivkumar Kalyanraman Omesh Tickoo, Vijaynarayanan Subramanian and K. K. Ramakrishnan. Lt-tcp: End-to-end framework to improve tcp performance over networks with lossy channels. In *In Proceedings of Thirteenth International Workshop on Quality of Service.*

[10] Stefan Saroiu, Krishna P. Gummadi, Richard Dunn, Steven D. Gribble, and Henry M. Levy. An analysis of Internet content delivery systems. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, pages 315–328, Boston, MA, December 2002.

[11] UCB/ISI et al. The Network Simulator ns2. http://www.isi.edu/nsnam/ns/.

[12] J. Widmer, R. Denda, and M. Mauve. A Survey on TCP-Friendly Congestion Control, May/June 2001.

[13] Jorg Widmer, Catherine Boutremans, and Jean-Yves Le Boudec. End-to-end congestion control for tcp-friendly flows with variable packet size. *SIGCOMM Comput. Commun. Rev.*, 34(2):137–151, 2004.

[14] The Emulab Testbed http://emulab.org

[15] Andreas Petlund, Kristian Evensen, Carsten Griwodz, Pal Halvorsen TCP Enhancements for Interactive Thin-Stream Applications In *Proceedings of the Network and Operating System Support for Digital Audio and Video (NOSSDAV'08), Braunschweig, Germany, May 2008, pp. 127 - 128*

[16] Marco Mellia, Michela Meo, Claudio Casetti TCP Smart Framing: A Segmentation Algorithm to Reduce TCP Latency *IEEE/ACM Transactions on Networking*, 13(2):316–329, April 2005

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

[18] W. Feng, D. Kandlur, D. Saha, K. Shin The Blue Queue Management Algorithms *IEEE/ACM Transactions on Networking*, 10(4), August 2002