2006

On the Performance Evaluation of High-Speed Transport Protocols

Bridget Hillyer
University of North Florida

ON THE PERFORMANCE EVALUATION OF HIGH-SPEED TRANSPORT PROTOCOLS

by

Bridget Hillyer

A thesis submitted to the
Department of Computer and Information Sciences
in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

May, 2006

The thesis "On the Performance Evaluation of High-Speed Transport Protocols" submitted by Bridget Hillyer in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by the thesis committee: Date

Signature Deleted

5/30/06

Dr. Sanjay Ahuja
Thesis Advisor and Committee Chairperson

Signature Deleted

6/5/06

Dr. Zornitza Prodanoff

Signature Deleted

5/30/06

Dr. Robert Roggio

Accepted for the Department of Computer and Information Sciences:

Signature Deleted

6/5/06

Dr. Judith Solano
Chairperson of the Department

Accepted for the College of Computing Sciences and Engineering:

Signature Deleted

9/5/06

Dr. Neal Coulter
Dean of the College

Accepted for the University:

Signature Deleted

11 SEPT 2006

Dr. David Fenner
Dean of the Graduate School

ACKNOWLEDGEMENT

I wish to thank Yongguang Zhang and Thomas Henderson for the use of XCP for Linux for evaluation purposes. XCP for Linux Release 1.0.1 (Aug/2004), is a product of HRL Laboratories and Boeing Phantom Work, Copyright 2004, The Boeing Company.

This research would not have been possible without the system administration help provided by Jeff Bowen and Mike Holmes. Expert Linux advice was given by Mike Russos.

My husband, Keith Hayes, has been exceptionally patient, supportive, and kind throughout this entire process. Thank you.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

As high-speed networks with large bandwidth delay products (BDP) become more common, high-speed transport protocols must be developed that perform well in these contexts. TCP has limitations in high BDP networks. A number of high-speed TCP proposals have emerged, including BIC TCP, High Speed TCP, and H-TCP. XCP is an intraprotocol communication mechanism that promises even greater performance by providing explicit feedback from routers about congestion. It requires changes to routers and end hosts, though, whereas the other experimental protocols only require changes to an end host.

We evaluated the performance of XCP against BIC TCP, High Speed TCP, H-TCP, and NewReno TCP. We found that in a controlled environment, XCP gave much better performance than the other TCPs. XCP was sensitive to misconfiguration and environmental factors, though, and was more difficult to deploy. More work is required to make XCP more stable. The other TCPs did not perform better than NewReno TCP but show promise, as most performed almost as well as NewReno TCP.

Chapter 1

INTRODUCTION

The Transmission Control Protocol (TCP) is the end-to-end protocol of the Internet and has contributed to the success of the Internet. It has been successful because it is conservative and stable. As the Internet has grown, links have developed much greater capacity. In addition, newer types of links such as satellite and wireless links have longer delay. These types of links can be described as high bandwidth delay product links, or high BDP links. The capacity, or BDP, of a link is measured as a product of bandwidth and round-trip time (RTT). As either or both of these factors increases, the BDP increases. It is well known that TCP does not operate well on high bandwidth delay product networks. Its conservative policies lead to inefficiency, amongst other problems.

TCP has a number of problems in high BDP networks. TCP increases its congestion window - the amount of unacknowledged data a TCP flow can have in the network at one time - by one packet per RTT. It decreases the congestion window by half upon a loss event. This leads to inefficiency. TCP is unable to keep high BDP links full, therefore underutilizing bandwidth. Another problem is that TCP is prone to instability and becomes oscillatory. Also, since TCP can only increase by one packet per RTT, it is unfair to long RTT flows. Finally, since all packet loss is interpreted as a congestion indication, loss due to errors is incorrectly interpreted as congestion.

These problems with TCP adversely affect satellite links because of their long RTT. These adversely affect wireless links because of their high error rates. More and more applications exist that could take advantage of high bandwidth delay product networks. These include sharing of large scientific datasets and multimedia. These applications and types of links require improvements to TCP.

In response to these problems, a number of proposals for high-speed TCPs have been made including High Speed TCP [Floyd02], Scalable TCP [Kelly03], BIC TCP [Rhee04], H-TCP [Leith04], and FAST TCP [Low04]. Each makes changes to the congestion control algorithms of TCP. Congestion occurs when a bottleneck in a network path is overwhelmed. Congestion control is the way TCP handles and avoids congestion events. A more radical proposal is XCP, the eXplicit Control Protocol, which unlike the other TCPs, makes changes to the routers as well as the end-systems. It gives explicit feedback about congestion in the network; feedback which is not present in the current TCP. XCP is more than just a new transport protocol; it is an interlayer communication protocol, allowing communication between the network and transport layers. The explicit feedback allows end-systems to react quickly and accurately to congestion in the network.

XCP has been shown to have a number of benefits. It has been shown in simulation and experiments to have high utilization, maintains small queues, and exhibits almost no packet loss [Katabi02, Zhang05]. Overall, its performance appears to be impressive. Additionally, it decouples fairness and utilization control. Nonetheless, problems with

the protocol have been noticed, such as its sensitivity to a number of environmental factors and incorrect control information [Zhang05]. The most troubling fact about XCP is the amount of changes required in the network for its implementation. All routers in a path as well as the end-systems must be changed. Therefore, there are concerns about its deployability.

With so many competing proposals to replace TCP, careful performance evaluation must be undertaken to determine which ones are deserving of further investigation. A number of studies have been undertaken comparing the performance of the various high-speed TCPs. A problem with these performance studies is that there is no standard, agreed-upon set of metrics or methods. There is ongoing work to develop these standard metrics and methods by different groups, including the Transport Models Research Group, chartered by the IRTF [Floyd05B]. One concern is ensuring a realistic environment to test the TCPs that actually matches the real Internet environment. Some factors include types of background traffic and bottleneck capacity. Another issue is controlling for variation in the network stacks used for testing the various proposals. Everything, including operating system kernel and hardware, should be the same between the different tests.

Some research efforts have evaluated the performance of XCP [Zhang05, Falk05B]. So far, no one has compared the performance of XCP against any of the other high-speed TCPs. It is a challenge to compare XCP to the other high-speed TCPs because they are fairly different types of protocols. XCP is inter-protocol communication, requiring

changes at the routers and end-systems. The other high-speed TCPs only require changes at the end-systems and are essentially just updates to TCP. Also, most of the high-speed TCPs have been well-integrated into the Linux kernel and deployed; XCP has not been vetted in this way.

We conducted a performance evaluation to compare XCP to the other high-speed TCPs. It was an initial attempt to understand how these could be compared. XCP, BIC TCP, High Speed TCP, and H-TCP were all compared with the current NewReno TCP. Tests were run with traffic generators, and environmental factors, including delay and loss were emulated. Measures such as average bandwidth and packet loss rates were taken and compared.

Chapter 2

TRANSMISSION CONTROL PROTOCOL

TCP is the main end-to-end protocol operating in the Internet today [Postel81]. The Internet Protocol (IP) provides an unreliable, connectionless packet delivery service. TCP sits on top of IP to provide the reliable, connection-oriented service that many applications require. TCP has two main functions: providing this reliable data delivery service and performing congestion control to protect the network from incapacitating congestion.

2.1 Reliable Data Delivery

As part of TCP's reliable data delivery service, it does the following things: reliable transmission of data, in-order delivery of bytes, flow control, and multiplexing. Reliable transmission of data involves using acknowledgements, checksums, and a retransmission timer to ensure that error-free data is delivered to a destination. TCP ensures that data is delivered in order at the level of bytes by using per-byte sequence numbers and a sliding window. The sliding window is also used for flow control where the receiver advertises a window size of the amount of data it can accept without getting overwhelmed. Finally, incoming data is multiplexed between multiple processes through the use of port numbers.

2.1.1 Reliable Transmission

The first thing that must happen with a TCP connection is that the connection must be established. This is accomplished with the three-way handshake. When a sender wishes to open a connection with a receiver, it sends a TCP segment with the SYN flag set. The SYN flag can be seen in the TCP header in Figure 1. The receiver replies with the SYN and ACK flags set, and the sender replies with the ACK flag set, finishing the handshake.

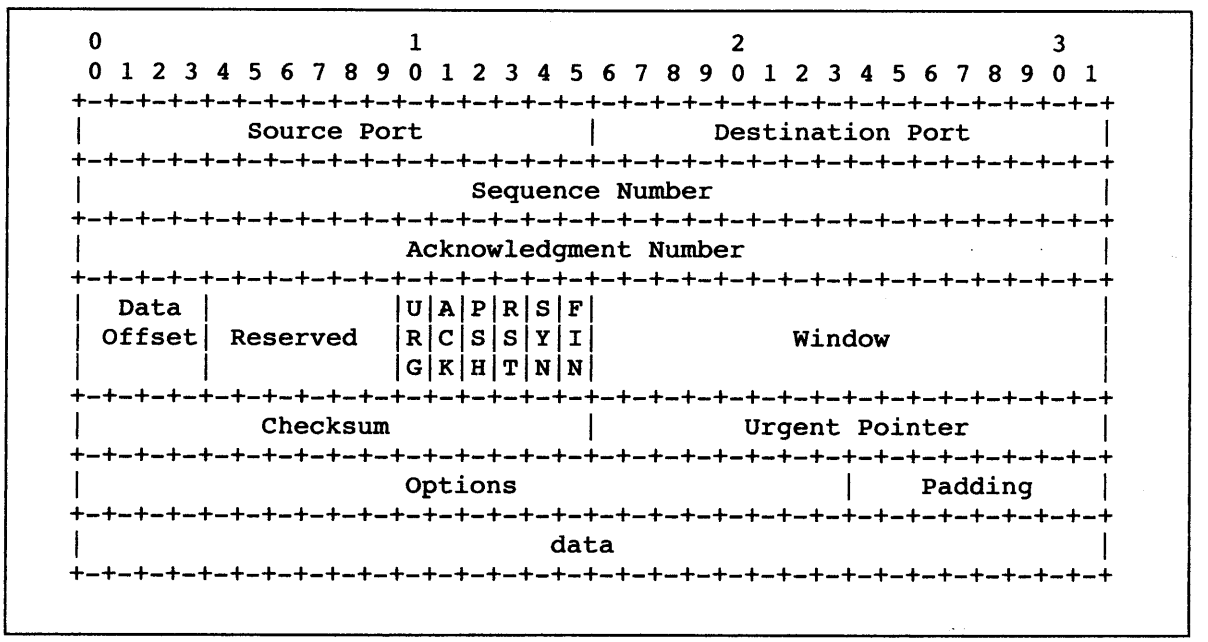

Figure 1: TCP header.

Other things that happen during connection establishment are that TCP parameters and options can be set. There are parameters such as window scaling and other extensions to TCP that can be negotiated between the sender and receiver. The options field can be seen in the TCP header. Options are additions to TCP that may or may not be present in the sender or receiver.

After the connection is established, data can be transmitted, and the receiver tells the sender it received the data through the use of positive acknowledgements. A positive acknowledgement returns the sequence number of the byte after the last byte that was received. The acknowledgements are also cumulative; they acknowledge the last segment that was received where each segment before it was also received. The checksum field can also be seen in the TCP header. It is used to guarantee that data arrives error-free.

The sender has a retransmission timer, which keeps track of how long it has been since a segment was sent out. If an acknowledgement has not been received by the RTO (retransmit timeout) the data is retransmitted. The RTO is dynamically updated to the smoothed RTT plus variance. Samples of the RTT are taken continuously.

2.1.2 In-Order Delivery

The sequence number field can be seen in the TCP header in Figure 1. Each byte has a sequence number in TCP. There are 32 bits for the sequence number field, which allows

a sequence number space of 2^{32}. Today larger amounts of data move through greater capacity and longer delay links, so this sequence number space is inadequate to guard against sequence number wrap-around, where the same sequence number could be seen within the same connection. Therefore, a timestamp option is used that appends a timestamp to the sequence numbers, increasing the sequence number space.

Together with the sliding window, this allows for in-order delivery of bytes. The sliding window, which is also used for flow control, is a means to keep track of what data is outstanding, or "in-flight," i.e. sent, but not yet acknowledged. Receivers can check the sequence numbers of incoming data to make sure they fall within the window, and can reorder those that arrive out of order. If out of order segments arrive, only the last ordered byte will be acknowledged, so lost segments will be retransmitted. As the new out of order segments arrive, duplicate acknowledgements (ACKs) will be generated. Therefore, the sender can begin retransmitting when it sees a number of duplicate ACKs. This is called fast retransmit.

The cumulative acknowledgement mechanism of TCP is somewhat limiting when it comes to retransmissions. Another extension to TCP is selective acknowledgements, or SACK. When SACK is implemented, non-contiguous blocks of data can be acknowledged, allowing for retransmission of only those segments that have not arrived at the receiver. It has been shown to improve performance and is being used more and more in the Internet.

2.1.3 Flow Control

The sliding window is also used for flow control. The receiver advertises the amount of data it can handle at one time through acknowledgements. This is called the receiver window. The window field can be seen in the TCP header. It uses 16 bits, so the window can be at most 2^{16} bytes. This is too small given the needs of applications, the larger capacity, and longer delay links available. Therefore, the Window Scaling option was added with RFC 1323, which allows a much larger window size with a scaling factor applied.

The receiver is able to throttle the sender with the use of the receiver window to overcome rate mismatches between the two. If the receiver does not have enough buffer space to handle incoming data, it can reduce the size of the receiver window and send that back in an acknowledgement to the sender. The sender must then reduce the amount of data it is sending. The sender uses the smaller of the receiver window and the congestion window, which will be seen later in the discussion of congestion control.

2.1.4 Multiplexing

Each end-system can have multiple processes sending and receiving data over the network. TCP distinguishes between the different processes with the use of port numbers. A port number is a 16-bit integer. Together with the IP address, this uniquely identifies the connection on that host.

The standard application programming interface to TCP is the Berkeley Socket interface. A socket is an abstraction for the connection. It can be read to and written from, much like files can.

2.2 Congestion Control

TCP's reliable data delivery mechanisms control how a host interacts with other hosts. TCP's congestion control mechanisms control how a host interacts with the network.

Van Jacobson is primarily responsible for the initial TCP congestion control algorithms from the 1980s [Jacobson88]. These algorithms allowed TCP to scale with the growth of the Internet, avoiding congestion collapse. Congestion occurs when too many packets create a bottleneck on a network path and buffers at the bottleneck fill up. Unnecessary retransmissions can then further bog down the network, leading to collapse.

Congestion control mechanisms in TCP both attempt to avoid congestion and control congestion once it has been detected. The basis of the mechanisms is the congestion window (cwnd), the number of bytes that can be sent based on current delay in the network. The only indicator of congestion is packet loss. There are various congestion control mechanisms at work in current TCP implementations. One is Additive Increase Multiplicative Decrease, or AIMD, in which the congestion window is halved every time packet loss occurs and increased by one segment each Round Trip Time (RTT) if there is no packet loss. Another is the retransmit timer, which uses exponential backoff when a

retransmitted packet is dropped. Slow Start is another mechanism. It checks the network to see if it can handle traffic by increasing the window by one segment per acknowledgement. Another mechanism is ACK-clocking in which the arrival of acknowledgements at the sender sets the pace for the transmission of new data.

2.2.1 Congestion Control Mechanisms

First, TCP probes the network to see how much load it can handle. This process is called Slow Start. After a connection is established, cwnd is set to 1 MSS (Maximum Segment Size) then increased segment per segment for each acknowledgement received. This produces exponential increase. When the acknowledgement for the 1 MSS arrives, cwnd is increased by 1 MSS. After the acknowledgement for those 2 MSS arrives, cwnd is increased by 2 MSS, then 4 MSS, then 8 MSS, and so on, thus increasing exponentially.

The TCP sender keeps track of another variable – ssthresh, or Slow Start Threshold. This is initially set to the size of the receiver window. Slow Start continues until it reaches ssthresh, then congestion avoidance begins. When a timeout occurs or three duplicate ACKs are received, signaling packet loss, and, therefore, congestion, one half of the current window size is saved in ssthresh. Upon a timeout, cwnd is set to 1 MSS and Slow Start begins again. If three duplicate ACKs are received, this also signals packet loss, and the apparently missing segment is retransmitted (Fast Retransmit) and congestion avoidance is entered (Fast Recovery) [Allman99A].

2.2.2 Congestion Avoidance

When ssthresh is reached, congestion avoidance begins. Then cwnd grows by one segment each RTT. This is linear growth compared to the exponential increase of Slow Start. This growth continues until congestion occurs. Window growth in congestion avoidance can be defined as:

cwnd = cwnd + 1/cwnd

This is the Additive Increase Multiplicative Decrease (AIMD) policy of TCP. Growth is linear, or additive, and, after packet loss occurs, the congestion window is halved, or decreased by a multiplicative factor. This is an efficient and somewhat fair control algorithm. Problems with fairness with the AIMD policy will be discussed below.

2.2.3 Tahoe, Reno, and NewReno

There have been three major versions of TCP. The first was Tahoe, which included most of the behavior described above. The significant difference was that upon receipt of duplicate ACKs, Fast Retransmit was entered, but, then, instead of entering congestion avoidance, cwnd was reduced to 1 MSS and Slow Start began again. The next version of TCP, Reno, added Fast Recovery. Fast Recovery is an attempt to stay in congestion avoidance instead of falling back to Slow Start. In Fast Recovery, cwnd becomes:

cwnd = ssthresh + 3 MSS

This is to account for the three packets for which duplicate ACKs were sent. Each time a duplicate ACK is received after this, another MSS can be sent until new data is acknowledged. Then cwnd goes back to ssthresh [Allman99A].

NewReno is the newest version of TCP, and is now the most deployed version on the Internet [Floyd05A]. What distinguishes it from TCP Reno is how it behaves if multiple segments are lost during Fast Recovery. Reno cannot recover from multiple packet loss during Fast Recovery. In NewReno, if a partial ACK is received, signaling that another segment was lost, it is retransmitted [Floyd99].

2.2.4 Congestion Control in High BDP Networks

Although these current congestion control schemes have allowed for the explosive growth of the Internet, there are problems with them. The problems are particularly apparent in large bandwidth delay networks. There are three main scenarios that present challenges. First, there is poor link utilization in high bandwidth delay networks. TCP's additive increase means that flows cannot fill up spare bandwidth quickly. A single TCP flow can fill up a 10 Gbps link, but this creates large, oscillatory queues. Also, in the presence of any realistic level of packet loss, the throughput is cut dramatically [Floyd02]. Second, TCP flows with long RTT cannot obtain a fair share of bandwidth at bottleneck links. Shorter RTT flows can open up their congestion windows faster, more quickly grabbing available bandwidth. Third, since dropped packets are the only indication of congestion, lossy links (such as wireless links) can be unnecessarily forced

into congestion response. Network capacity is increasing and new technologies such as satellite and wireless links are becoming more common, making these problems more significant than they have been in the past [Falk03].

Chapter 3

EXPLICIT CONTROL PROTOCOL

The designers of XCP considered the state of congestion control in the Internet and decided to start over from scratch. Working with no constraints – no requirements for backwards compatibility or concerns about deployment – what would a brand new congestion control system look like? Instead of the binary indicator of congestion of the current TCP, packet loss, they designed a system that provided explicit feedback about congestion. The feedback is not only correct about congestion (as opposed to using packet loss, which is sometimes due to error), but the feedback can also communicate the degree of congestion in a link.

The result was the eXplicit Control Protocol (XCP), a congestion control system for Internet transport protocols that performs well on high bandwidth delay product networks [Katabi02]. Each packet has a congestion header that carries per-flow congestion state. Routers along the way can modify the congestion header, providing feedback to the endpoints to control bandwidth allocation. Since the per-flow state is in the packet instead of the router, XCP can scale. One of the most important aspects of XCP is that it decouples congestion control from fairness control. Deployment of XCP would require changes both to the end hosts and the routers, so deployment is a concern [Falk03].

3.1 Description of Protocol

XCP is a congestion control system, which involves end-systems and routers. It belongs to neither the transport nor network layers, but, rather, provides feedback between the transport and network layers, violating the separation of layers principle. The end-systems provide feedback to the routers and the routers provide feedback to the end-systems. Each packet has a congestion header, which carries the congestion window, the estimated round trip time, and the requested increase in throughput. The router monitors its input traffic rate and provides feedback to flows on whether they should increase or decrease their congestion windows by updating the congestion header. Other routers on the path can overwrite the feedback so the ultimate feedback is from the bottleneck. The receiver copies the congestion header to an acknowledgement, which the sender uses to update its congestion window [Katabi02].

3.1.1 Congestion Header

The format of the XCP congestion header can be seen in Figure 2. The header sits between the IP and transport headers since it is communication between the end-systems and the network. The Protocol field is the protocol of the data in the packet. Length is the length of the congestion header in bytes. Version is the version of XCP. Format is a code that differentiates between a standard and minimal congestion header format. The RTT field is the beginning of the actual control information in the congestion header. It is the round trip time measured by the sender. X used to be throughput. It has been changed to inter-packet time of the flow calculated by the sender to avoid divisions in the

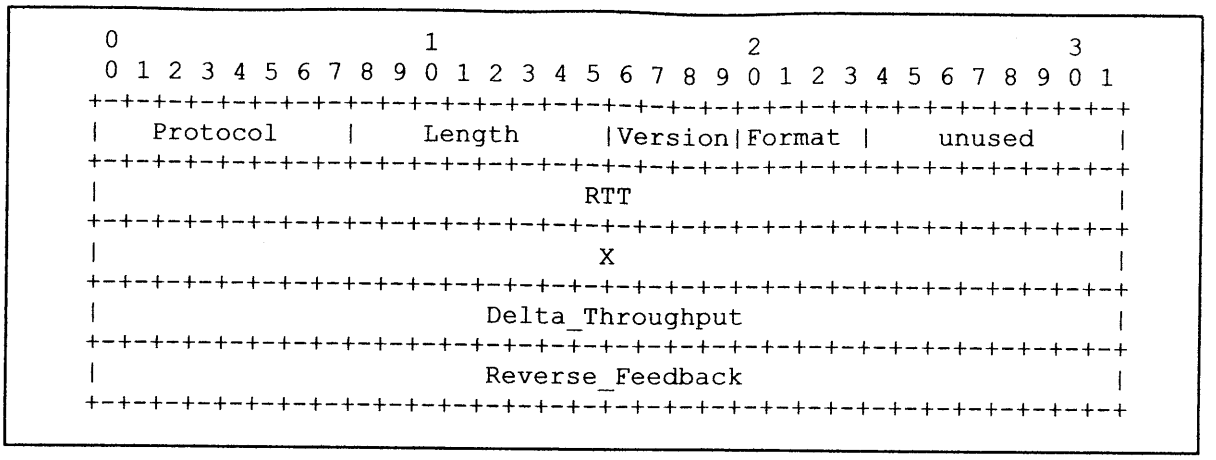

Figure 2: XCP congestion header.

router. The Delta_Throughput field is the desired or allocated change in throughput set by the sender. Routers along the path can change this field. Reverse_Feedback is the Delta_Throughput value received by the receiver. The receiver copies Delta_Throughput into the Reverse_Feedback field of an acknowledgement to return to the sender [Falk05A].

3.1.2 XCP End-Systems

The XCP sender maintains the congestion window (cwnd), desired throughput, estimate of current actual throughput, maximum throughput allowed by XCP, estimate of current inter-packet time, and an estimate of the current round trip time. It fills in the corresponding fields in the congestion header of outgoing packets. Delta_Throughput

(bytes/second) is calculated by dividing the desired throughput by the number of packets in a RTT so that the value is per-packet instead of per-flow, keeping to the principle of keeping per-flow state out of the routers. If this is the first packet, estimates are set to zero.

Delta_Througput = desired_throughput – throughput / throughput * (RTT/MSS)

where MSS is the Maximum Segment Size.

XCP receivers copy Delta_Throughput to the Reverse_Feedback field in the congestion header of returning acknowledgements. When the sender gets the feedback, it adjusts its cwnd by the following formula:

cwnd = max(cwnd + feedback * RTT, MSS)

Although packet loss should be rare with XCP, when it occurs, the system should fall back to standard TCP congestion control behavior [Falk05A, Katabi02].

3.1.3 XCP Routers

The XCP routers compute the feedback to send back to the XCP end-systems. They do this with an efficiency (or congestion) controller and a fairness controller. Having two separate controllers provides the decoupling between congestion and fairness control

central to XCP. The control decisions are made once per control interval. The control interval is computed as the average RTT of the flows on this link.

3.1.3.1 Efficiency Controller

The efficiency controller computes the desired increase or decrease in traffic based on the aggregate traffic. It is not at all concerned about the separate flows. The fairness controller is concerned only with splitting up the aggregate bandwidth between flows. The policy of the efficiency controller is Multiplicative Increase Multiplicative Decrease (MIMD). The efficiency controller computes the aggregate feedback each control interval as follows:

F = a * (capacity - input_bw) - b * queue / avg_rtt

Capacity is the link capacity in bytes/second. The input_bw is the average bandwidth of arriving traffic. The queue is the persistent queue, the minimum standing queue over a special queue estimation interval. The average RTT, avg_rtt, is the value used to govern the control interval. Two constant parameters, a and b, are set to 0.4 and 0.226, respectively, based on stability analysis [Falk05A, Katabi02].

3.1.3.2 Fairness Controller

The fairness controller takes the aggregate feedback computed by the efficiency controller, and splits it up among the flows on the link. It uses an Additive Increase

Multiplicative Decrease (AIMD) policy. The MIMD policy of the efficiency controller ensures that the spare bandwidth is allocated. The AIMD policy of the fairness controller converges to fairness much like the AIMD policy of TCP. These two policies working in concert accomplish both performance requirements of the protocol. Positive feedback is split equally per-flow (additive increase), and negative feedback is proportional to the flow's capacity (multiplicative decrease). The total feedback is computed as the positive feedback minus the negative feedback. Positive and negative feedback are calculated as follows:

pos_fbk = Cp * X

neg_fbk = Cn * Pkt_Size

Cp and Cn are, respectively, the scaled positive and negative feedback to be distributed. X is inter-packet time. Pkt_size in the negative feedback calculation makes the negative feedback proportional to the flow's capacity.

If the aggregate feedback is zero, bandwidth shuffling is applied. Bandwidth shuffling redistributes a small portion of the capacity so that new flows can obtain a share of the capacity in a fully loaded system and converge to fairness.

3.2 Deployment Issues

Since XCP requires changes to routers as well as to end-systems, there are significant deployment issues. First, the protocol must be very mature and well-tested before router manufacturers would agree to deploy it in routers. Second, XCP, with its communication across transport and network layers, is not an incremental change to Internet congestion control. How such a large change could take place is difficult to imagine. Would it happen on a cloud-by-cloud basis? Would it be deployed wholly on research networks such as Internet2? Also, if it is to coexist with regular TCP, can it fairly share bandwidth with regular TCP flows?

In the original XCP paper, Katabi describes a way to deploy XCP on a per-cloud basis. TCP flows would be mapped to an XCP flow as they enter the cloud. Katabi also suggests that XCP could be TCP-friendly in a network shared with TCP flows. A router could have an XCP and a TCP queue and could use weighted-fair queuing so both would have equal average bandwidth [Katabi02].

The XCP specification discusses some of the issues of XCP deployment and offers some ideas of how it could be handled. One idea is to use XCP as a TCP performance-enhancing proxy (PEP). TCP PEPs are used in networks today to improve performance, sandwiching a more aggressive protocol between two TCP connections. XCP could be run in between two TCP connections as a PEP, and this could be used for incremental deployment [Falk05A].

3.3 Simulations and Tests of XCP

Early simulations and tests had promising results for XCP. Katabi's simulations showed that XCP could give fair bandwidth allocation, almost no packet drops, high utilization, small standing queue size, and scalability [Katabi02]. Zhang confirmed these findings with his implementation of XCP, but also went on to test a number of factors such as link sharing, non-congestion losses, and non-XCP queues [Zhang05]. He found that in some cases XCP could develop incorrect feedback that negatively affects its performance.

In the paper that originally described XCP, Katabi reported the results of simulations with the network simulator ns-2 that compared the performance of XCP against TCP Reno with four different queuing schemes: Random Early Discard (RED), Random Early Marking (REM), Adaptive Virtual Queue (AVQ), and Core Stateless Fair Queuing (CSFQ), all with Explicit Congestion Notification (ECN) enabled. They simulated different capacities, delays, number of sources, and topologies. The results were as mentioned above: XCP outperformed TCP in all cases [Katabi02].

Zhang implemented XCP in Linux as a TCP option. He studied the effects of environmental factors on the performance of XCP in an experimental test bed using his Linux implementation. XCP requires accurate control information such as link capacity in order to work correctly. Problems arise when it is unable to get a good estimate of the load. Zhang manipulated different factors such as the configured sending rate, link contention, lossy wireless links, and non-XCP queue bottlenecks. He found that XCP's

behavior is affected by all of these factors [Zhang05]. The experimental test bed looked like Figure 3. The dummynet bridge is used to emulate link delay. The link in and out of the bridge is 10 Mbps to create a bottleneck.

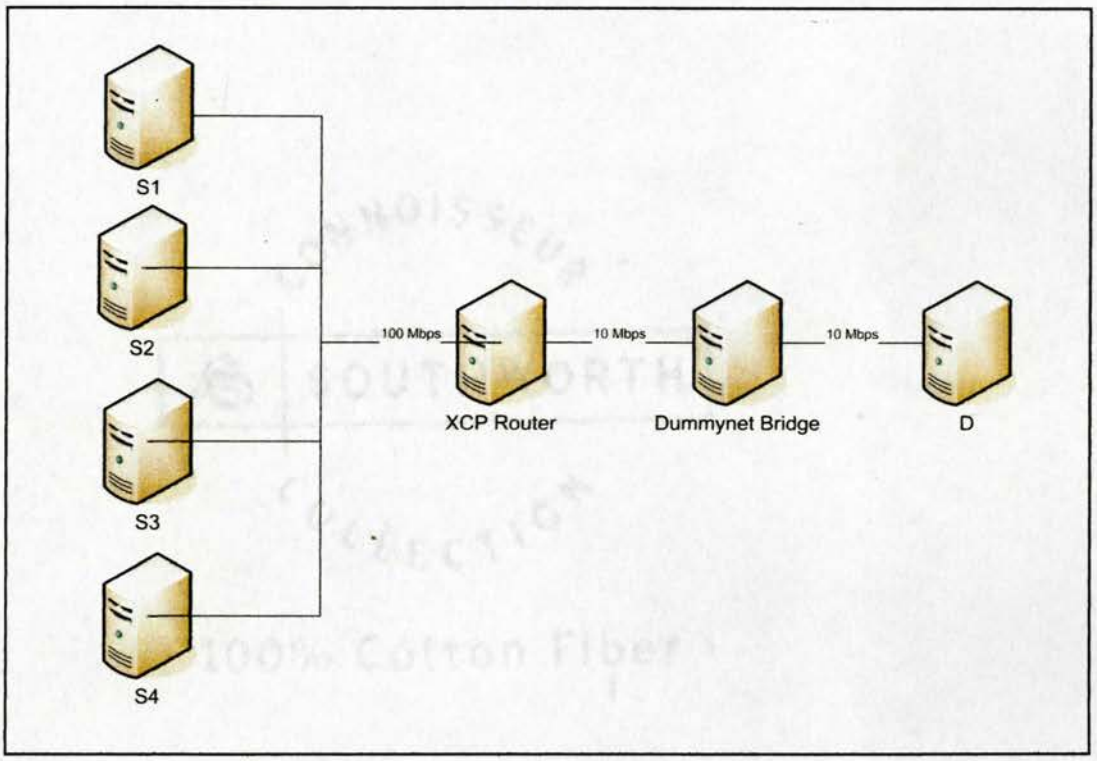

Figure 3: Test bed.

The ISI research group at USC has developed a FreeBSD implementation of XCP and is studying its performance. This FreeBSD implementation, including end-system and router kernel code, is available. Additionally, they have made available an ns-2 distribution with XCP code. An Internet draft of the XCP specification has also been prepared [Falk05A]. They have begun to publish preliminary results of their performance studies. One such study considers the performance of XCP over satellite networks. The authors compared the performance of XCP with other experimental high-speed TCPs – BIC-TCP and FAST TCP using an ns-2 simulation. They considered the situation when long RTT flows compete with short RTT flows and found that XCP performed the best in terms of link utilization and fairness [Falk05B].

3.4 XCP for Linux

Zhang developed his XCP for Linux as a TCP option based on Katabi's initial description of XCP. This was before the XCP specification was written, so there are some differences between Zhang's implementation and the reference implementation being developed at ISI. XCP for Linux was chosen for this study because the other high-speed TCPs such as BIC TCP and High Speed TCP were developed for Linux. This way, the underlying operating system and most of the network stack would be the same between the different experimental protocols.

One main difference between Zhang's implementation and the reference implementation is XCP for Linux is a TCP option, and the reference implementation sits between the

transport and network layers. It is closely integrated with the Linux TCP implementation this way. Also, XCP for Linux is based on the original description of the protocol, and the protocol has evolved as the specification has been developed. The congestion header in XCP for Linux is closer to what Katabi described. It has three fields: H_feedback, H_rtt, and H_cwnd. This has changed to what was described above: RTT, X, Delta_Throughput, and Reverse_Feedback. These different fields have made some changes to the calculations in the router. Finally, XCP for Linux does not use a control interval in order to avoid using a high-resolution timer in the Linux kernel but instead uses a delayed computation approach [Zhang05].

Chapter 4

HIGH-SPEED TRANSPORT PROTOCOLS

Several proposals for high-speed TCPs exist. These include BIC TCP, High Speed TCP, H-TCP, FAST TCP, and Scalable TCP. The high-speed TCPs included in our experiments are considered in detail below – BIC TCP, High Speed TCP, and H-TCP. Their performance in various experiments to compare the different high-speed TCPs is also noted. The methodology of these experiments will be discussed in Chapter 5.

4.1 BIC TCP

Injong Rhee at North Carolina State University devised a new congestion control algorithm called BIC TCP that uses a form of binary search to find the most appropriate sending rate for current network conditions [Rhee04]. BIC TCP stands for Binary Increase Congestion control TCP. This high-speed TCP was designed to exhibit RTT fairness, TCP friendliness, and stability. Results of performance evaluations compared to other high-speed TCPs have been favorable [Rhee04, Rhee06, Cottrell03, Leith05]. BIC TCP was implemented as a modification of TCP NewReno with SACK and is now included by default in the 2.6 version of the Linux kernel.

The binary search increase portion of BIC TCP works as follows. A current minimum window size, W_{min}, and maximum window size, W_{max}, are maintained. These are, respectively, the window size after fast recovery and the window size when the last packet loss occurred. The midpoint between W_{min} and W_{max} is computed to determine the new window size. Then feedback from the network is checked to determine if there was packet loss or not. If there was packet loss, that window size is set to the new W_{max}, and if there was not packet loss, that window size is set to the new W_{min}. This is repeated until the difference between W_{max} and W_{min} is less than a minimum increment, S_{min}. The increase function of binary search increase is logarithmic, unique amongst the high-speed TCPs. When the window size approaches the point of previous packet loss, the increase rate gets smaller, reducing packet loss.

There is also an additive increase portion of the algorithm. When the midpoint between W_{min} and W_{max} constitutes an amount of increase greater than a maximum increment, S_{max}, the window size is increased by S_{max} until the increase is less than S_{max}.

The decrease function is multiplicative decrease with a factor, β, set to 0.8. There is also a slow start period after the window size grows past W_{max}. It adds an exponential factor of S_{min} to W_{max}, and then switches to additive increase. A fast convergence stage allows smaller windows to get a fair share of bandwidth. Finally, regular TCP behavior is followed at low window sizes [Rhee04].

Cottrell found BIC TCP performed very well in their experiments. It gave good utilization, stability, and fairness. BIC TCP showed some stability problems on long paths with large windows [Cottrell03]. Hamilton Institute experiments showed some problems with fairness and convergence times [Leith05]. Rhee also found RTT unfairness issues with BIC TCP, but the addition of background traffic to experiments improved BIC's fairness. Background traffic also improved convergence times. Rhee believes that the control dynamics of BIC interact more properly with background traffic than the other protocols [Rhee06]. BIC TCP seems to be a favorite of the research community. It provides good utilization and an improvement over Reno TCP with only a change at the sender.

4.2 High Speed TCP

High Speed TCP was proposed by Sally Floyd as a change to regular TCP for large congestion windows [Floyd02]. It has also been evaluated in performance studies, and has been shown to have some increase in performance over regular TCP [Cottrell03, Leith05, Rhee06]. It is relatively simple to implement, only requiring a change in the sender. It is also in the 2.6 Linux kernel as an alternative congestion control algorithm.

Like BIC TCP, High Speed TCP does not change the regular TCP congestion control algorithms for small windows. Once larger windows are reached, the response function changes to a function of the size of the congestion window. That is, the congestion window size is looked up in a table to find the appropriate increase factor. This provides

a more aggressive increase function. The decrease factor for High Speed TCP is correspondingly less aggressive than regular TCP, also based on a table of values [Floyd02].

Cottrell found HS-TCP to be "gentle" – it does not aggressively grab bandwidth from competing protocols. In fact, they found it to exhibit odd intra-protocol behavior. When an HS-TCP flow shares a link with another HS-TCP flow, the first flow gives up bandwidth to the second flow [Cottrell03]. The experiments carried out at the Hamilton Institute also showed fairness problems with HS-TCP. They additionally found that convergence times could be long. There were some issues with RTT unfairness, too [Leith05]. Rhee found HS-TCP exhibited better fairness as background traffic was added to their experiments [Rhee06]. Despite issues with performance noticed in the various experiments, HS-TCP was found by all to give an increase in utilization over standard Reno TCP with a small change in the sender host.

4.3 H-TCP

H-TCP was developed at Hamilton Institute [Leith04]. Its performance has been evaluated alongside BIC TCP, High Speed TCP, and other high-speed TCPs. Outside of studies undertaken by the Hamilton Institute, the performance of H-TCP has not been as favorable as the other protocols [Leith05, Cottrell03, Rhee06].

H-TCP is like High Speed TCP in that it is a modification to the regular TCP AIMD congestion control algorithms. It changes the increase factor to be based on the time elapsed since the time of the last congestion event. Like H-TCP, High Speed TCP only changes the response function for large congestion windows. The decrease factor is reduced to make it less aggressive, based on the maximum throughput seen since the last congestion event [Leith04].

Cottrell found H-TCP to have good stability and fairness [Cottrell03]. Hamilton Institute (the creators of H-TCP) found H-TCP to have good fairness and convergence times. They found H-TCP to have much better RTT fairness than the other protocols [Leith05]. With the addition of background traffic, Rhee found H-TCP gave lower utilization, less stability, higher packet losses, and less RTT fairness. On the other hand, with background traffic added, H-TCP still had good fairness and convergence times [Rhee06].

4.4 Other High-speed Transport Protocols

Other high-speed TCP proposals exist. One of the most promising, besides the ones noted above is FAST TCP. FAST TCP is delay-based and grew out of TCP Vegas [Low04]. FAST TCP won the Bandwidth Challenge at SC2005. It uses queuing delay in addition to packet loss as congestion measures. It also uses pacing at the sender to avoid losses. Unfortunately, although an implementation is available for the 2.6 version of Linux, the version for Linux 2.4 is no longer available. The developers decided to close

the source; since the 2.4 version was a patch against the kernel, they can no longer distribute it. Since our experiments were against the 2.4 version of the Linux kernel, we could not include FAST TCP in our experiments.

Another proposal is Scalable TCP from Tom Kelly [Kelly03]. Scalable TCP uses an exponential increase factor and a smaller multiplicative decrease factor. Other proposals include TCP-Westwood, LTCP, and TCP-Africa. Modifications to TCP for high-speed networks are an active area of research and will likely continue to produce more variants.

Chapter 5

PERFORMANCE STUDIES

5.1 Experimental Evaluation of High-speed Transport Protocols

A number of studies have compared the performance of the various high-speed TCPs. It is an active research effort to develop methods and metrics for measuring the performance of high-speed TCPs. The results of these studies have been mentioned in Chapter 4 in regards to the different high-speed TCPs. The methods and measurements of the studies are themselves a subject of interest.

Bullot, Cottrell, and Hughes-Jones compared the performance of HS-TCP, Fast TCP, Scalable-TCP, HSTCP-LP, H-TCP, and BIC-TCP in real high-speed production networks [Cottrell03]. They considered throughput, RTT, intra- and inter-protocol fairness, stability, and impact of reverse traffic. They found that all of the experimental TCPs had better performance than TCP NewReno with SACK. Evaluating these protocols over a real network is very enlightening and an important step in testing a protocol's readiness for real network use. The downside of testing in a real network is environmental factors cannot be controlled.

Leith and Shorten of Hamilton Institute (developers of H-TCP) also compared the performance of these TCPs, but in a test bed, with a focus on fairness between competing flows [Leith05]. They also attempted to develop a set of benchmark tests, which control for differences in network stack implementation. All machines used a modified version of the Linux 2.6.6 kernel. The results of both of these performance studies are complex and indicate a wide variation of fairness, aggressiveness, and stability issues with the various TCPs. They indicate a good deal of further study is needed to understand the performance aspects of these high-speed TCPs.

Rhee (the developers of BIC TCP) created an experimental test bed and evaluated an exhaustive list of recent high-speed TCPs: BIC, CUBIC, FAST, HSTCP, H-TCP, and Scalable TCP [Rhee06]. Like the other studies, their focus was on developing methodologies for evaluating TCPs. The focus of their experiments was to test the effects of background traffic. Background traffic can change the behavior of network protocols, and they aimed to see what behavioral changes they could observe. All of the protocols they examined showed different behavior with the presence of background traffic compared to experiments run without background traffic. For example, all of the protocols, except FAST, became more TCP friendly (share bandwidth with regular TCP) when background traffic was added. With background traffic, the high-speed TCPs have less bandwidth so become less aggressive with smaller window sizes. Also, random packet drops increase with increased traffic, changing the dynamics of the protocol behaviors.

None of these performance evaluations include XCP, highlighting the difficulty in comparing XCP to the other high-speed TCPs. XCP makes changes to the router while none of the other TCPs do anything to the router. The TCP header is changed with XCP, so different tools must be used to inject traffic and monitor its behavior. XCP for Linux includes the XCP congestion header as a TCP option. As discussed above, Zhang measured the performance of XCP for Linux in an experimental test bed much like the other high-speed TCP performance studies but only against TCP Reno.

5.2 Issues in the Evaluation of Protocols

There is a great need in the networking research community for better protocol performance studies [Floyd03]. There are three main areas that must be addressed when developing a TCP performance study: model, metrics, and methods. Each area has special challenges.

There are many factors to consider when setting up a TCP experiment. TCP version, queuing scheme, number of flows, capacity, and propagation delay are just a sampling of the different factors that have to be determined when creating an experiment. There are many challenges in tuning these parameters. Research into the current use of the Internet can reveal what current traffic looks like and what networking configurations are most prevalent. This can be used to set up realistic experiments. With experimental protocol implementations though, the future of the Internet has to be taken into consideration. What will Internet traffic look like in the future? Will traffic continue to be dominated

by short web-like flows? Will satellite links become very common, making long delays more common?

One of the main choices to make when setting up a TCP study is whether to do a simulation or an implementation study. Both have advantages and disadvantages. Networking protocols can be fully simulated in software. Some of the available simulators include OpNet, x-sim, and the Network Simulator, ns-2. The benefit of using a test bed is an actual TCP implementation can be tested on a real network, although this can be limited by the amount of equipment on hand. Also, a test bed is limited in that it does not accurately reflect protocol performance on the Internet. [Allman99B]. The chief advantage of doing a simulation is it is inexpensive since it requires less equipment. Both simulation and implementation studies are necessary, so the decisive factor in deciding between the two is what is to be studied.

Ultimately, the most important consideration when setting up an experiment is what TCP model will be used. Some factors in a TCP model are topology, type of traffic, queuing policies, and congestion. There are many competing considerations when setting up a model. One consideration when picking a model is isolating behaviors of the protocol. Another consideration is what is being studied. For example, if congestion control behaviors are being studied, a congested link must be included in the test bed or simulation setup. These two issues can lead to tradeoffs. In the case of using background traffic, the behavior of the protocol is no longer isolated. Nonetheless, the protocol operating in isolation of background traffic is an unrealistic situation.

Understanding the current (or future) situation of the Internet can determine what is realistic. Most flows on the Internet are short, so using short flows in an experiment would be realistic. This may not be what is being studied about the protocol though, so bulk transfers may be used to elicit a certain behavior of the protocol [Floyd03].

What is to be studied about a protocol also drives what metrics will be collected in the experiment. Some standard metrics are throughput, link utilization, packet drops, and latency. If the effects of a protocol on an application such as VoIP are being studied, jitter is important, and so queuing delay should be measured. With bulk transfers, throughput and delay are most important. There is some disagreement in the networking community about which metrics are important, so this is an area where standards may be useful.

Finally, when setting up and executing a TCP experiment, all details of the experiment should be recorded and published so it can be repeated. This includes TCPs and versions, operating system kernel version, TCP parameters such as buffer sizes, router queuing method, topology, traffic, length of experiment, monitoring tools, etc.

There have been a number of proposals for standard methods and metrics for TCP performance evaluation. One such effort is the Transport Modeling Research Group (TMRG) chartered by the IRTF. They are producing a number of documents that provide models for the evaluation of transport protocols [Floyd05B]. Wei also made a

proposal for a TCP benchmark suite [Wei05]. A compendium of the various proposals for methods, models, and metrics are presented below.

5.2.1 Methods

There is general agreement that new protocols must be subjected to both NS simulation and test bed experiments before they are fully understood and accepted. Live Internet tests are also useful for seeing how the protocols operate in a real-world setting.

Test bed experiments must be setup in a way that accounts for the vagaries of the hardware. Different network interface cards and drivers can have a large effect on performance. CPU utilization, NIC performance, and other system parameters should be monitored. Network stacks should be the same between experiments in every way except for the piece being studied. Network stack parameters should be tuned to elicit the behavior of the protocol and in a consistent manner. Network setup should be tested before experiments are run to ensure all path parameters such as link capacity are what are expected. There are many factors at work even in a simple test bed, and they must all be controlled [Wei05].

5.2.2 Models

Wei's proposal for a benchmark suite puts forth suggestions for topologies, traffic patterns, loss rates, and so forth, which would be reasonable in a TCP study [Wei05].

This is a proposal though, and consensus does not exist in the networking community about these models. Two considerations can be followed with some certainty. First, these models should reflect current reality of the Internet. Second, experiments should vary one factor at a time. Following is a list of some of the proposed model factors and suggested values.

- Network topology and configuration – number of nodes and connections between the nodes. Dumbbell topology (see Figure 4) is the most common topology used in networking research. Others, such as ring and parking lot, can also be used.
- Workloads – where and when flows start and end. Small number of flows and many flows should be tested. Start and end times should be varied to show link sharing and synchronization behaviors. Poisson arrival with heavy tail file size distribution is known to be a typical workload in the Internet. Tests must also run long enough for the protocols to converge.
- Link capacities: 56 Kbps, 10Mbps, 100Mbps, 155Mbps, 622Mbps, 1Gbps, 2.5Gbps, and 10Gbps are all values that are seen in the Internet today.
- Propagation delay: 1ms, 10ms, 50ms, 100ms, 200ms, and 600ms are all values seen in the Internet today.
- Packet loss rates: 0%, 0.001%, and 0.1% are all guesses that Wei made as to realistic packet loss rates; but these are not understood well [Wei05].
- Queuing discipline: method of queuing used in routers. Examples are Random Early Discard (RED), Random Early Marking (REM), Adaptive Virtual Queue (AVQ), and Core Stateless Fair Queuing (CSFQ). A queuing discipline governs

how packets are buffered while they are waiting to be transmitted. The most common queuing discipline seen in the Internet is drop-tail, or FIFO. When a queue fills up at the router, drop-tail drops the next incoming packet. The other queuing disciplines have more "intelligent" methods. For example, RED randomly drops packets when the number of packets in the buffer is between certain configurable thresholds. This helps avoid synchronization.

5.2.2 Metrics

In the Internet Draft, "Metrics for the Evaluation of Congestion Control Mechanisms," the following metrics are considered:

- throughput;
- delay;
- packet loss;
- responsiveness;
- throughput or delay oscillations;
- fairness;
- convergence;
- environmental robustness;
- failure and misbehavior robustness;
- deployability;

- specific transport metrics; and
- user-based metrics [Floyd05B].

In the proposal for a TCP benchmark suite, Wei includes most of the above, except robustness, deployability and user-based metrics [Wei05].

Of the above metrics, only throughput, delay, and packet loss need be measured. The others can all be calculated based on throughput, delay, and packet loss. Tools such as iperf can measure throughput. Delay and packet loss can be measured by Dummynet routers or can be inferred at the host with instruments such as Web100 [Wei05].

Throughput can be either aggregate throughput of all links, measured at the router, or per-connection, flow-based transfer time. Throughput is usually distinguished from goodput. Goodput is the successful part of the throughput – the packets that were actually useful and not dropped. Iperf measures goodput. Delay can be queuing delay at the router or flow-based transfer time. Packet loss can also be measured at the router or per-flow. It may be useful to measure whether packet loss is due to congestion or corruption [Floyd05B].

Responsiveness and oscillations are different sides of the same coin. Responsiveness is how quickly the protocol responds to changes such as congestion. A protocol should respond quickly, but should not respond too much to transient changes. Oscillations can occur in queuing delay and throughput. They should be minimized so the protocol is

stable [Floyd05B]. Stability can be measured as standard deviation in queuing delay and throughput [Wei05].

Fairness and convergence are also related. Fairness is how well different flows of either the same or different protocols share a link. There are different measures of fairness: max-min fairness, proportional fairness, Jain's fairness index, and the product measure. The most commonly used measure of fairness in the protocol studies and the one suggested by Wei is Jain's fairness index:

$$F = (\sum_{i=1}^{n} \bar{x}_i)^2 / n \sum_{i=1}^{n} \bar{x}_i^2$$

x_i is the average throughput of each flow [Floyd05B, Wei05]. In the case of high bandwidth delay product links, fairness between links of different round-trip times is also a concern [Katabi02]. Convergence is the time it takes to arrive at fairness when a new flow starts after an existing flow.

Robustness is the ability of the protocol to handle problems such as a difficult environment, failures, and misbehaving users. An example of a challenging environment is a wireless network with a high error rate. An example of a failure is a router going down. Misbehaving users can lie about the amount of congestion on a link to confuse congestion control mechanisms. Goodput is a measure that can be used for robustness [Floyd05B].

Floyd describes deployability in terms of the following change requirements of a protocol: at the sender, at the receiver, at sender and receiver, at a single router, at all routers on a path, at end-systems and all routers along a path. Complexity of protocol implementation is another concern [Floyd05B].

Certain transport protocols may require special metrics. Also, metrics can be user-based, such as application-specific functions [Floyd05B]. Metrics may become standardized, but these types of metrics show, to some extent, they must be fluid and reflect the current reality of what is being measured.

Chapter 6

EXPERIMENTAL EVALUATION

6.1 Motivation for Experiments

Three studies of the various high-speed TCPs were discussed in Chapter 5 [Cottrell03, Leith05, Rhee06]. The results of the studies were discussed in Chapter 4, and the methodologies were discussed in Chapter 5. We performed a similar study to evaluate the performance of the high-speed TCPs. We included XCP in addition to the other high-speed TCPs to see how the performance of XCP compares to the other high-speed protocols. Zhang and Henderson did a performance evaluation of XCP against NewReno TCP, but did not include any other high-speed TCPs [Zhang05].

Cottrell conducted his study on a real production network [Cottrell03]. The other three studies used test beds with emulation of environmental factors such as delay and loss [Leith05, Rhee06, Zhang05]. Our study uses a test bed that includes a bridge with a network emulator to emulate realistic loss and delay.

The Hamilton Institute study focused on protocol behaviors such as fairness [Leith05]. Rhee's study concentrated on protocol behavior in the face of background traffic [Rhee06]. Cottrell focused on protocol behavior important to scientific researchers who

need to share large datasets and are attempting to implement these high-speed TCPs in the current Internet. Therefore, they concentrated on measurements such as throughput, fairness, and stability, which would be important in current production networks where links would be shared with other protocol traffic [Cottrell03]. The XCP study validated previous simulation findings then looked at how environmental factors would affect the behavior of XCP [Zhang05].

The main focus of our study is comparing the performance of XCP to the other high-speed TCPs as well as to NewReno TCP. Therefore, we will concentrate on user or application-based performance measures such as average throughput and loss. Measures that capture protocol behavior such as fairness and stability are left to future studies. Measures such as fairness and stability are important in terms of the overall effect of a protocol on the network, such as how TCP-friendly a protocol is. TCP-friendliness is how well a protocol shares bandwidth with the older TCP, which is important as protocols are incrementally deployed. It is beyond the scope of this study, though.

Cottrell's study and the Hamilton Institute study emphasized having as uniform a network stack between all experiments as possible [Cottrell03, Leith05]. Our experiments emphasized this, as well. The hardware in the network test bed was the same in all experiments. Care was taken to have as similar a patch level for the Linux kernel as possible between the various high-speed TCP experiments. They were all performed on Linux kernel version 2.4.

Results can be compared to the four different performance studies [Cottrell03, Leith05, Rhee06, Zhang05]. Another consideration is recording enough detail of the experiments so they are reproducible. Care was taken to record as much detail as possible.

6.2 Experimental Setup

An experimental test bed was set up to test the performance of XCP, BIC TCP, High Speed TCP (HS-TCP), H-TCP, and NewReno TCP. Guiding principles for setting up the test bed and experiments came from the preliminary work done on metrics and methods standardization for TCP performance evaluation by the TMRG [Floyd05B] and Wei. [Wei05]. The other high-speed TCP performance studies by Cottrell [Cottrell03], the Hamilton Institute [Leith05], and Rhee [Rhee06] also each served as partial models.

The test bed looked like Figure 4. Each node was a standard commodity PC, a Dell OptiPlex GX400, with the configuration listed in Table 1.

CPU	Intel Pentium 4 1.40 GHz
Memory	512 MB ECC RDRAM
NIC	3Com Etherlink 10/100 PCI
Hard Drive	20 GB IDE

Table 1: Host configuration.

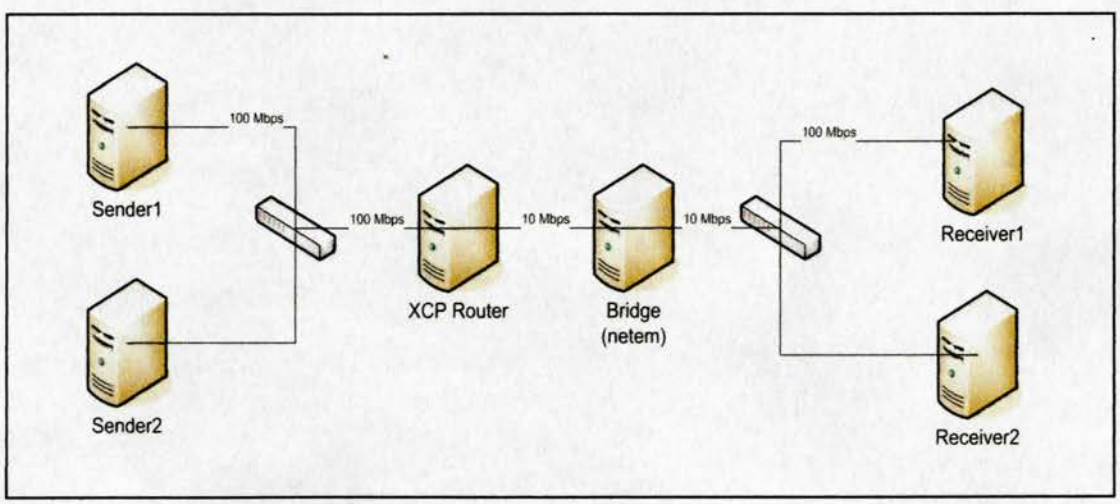

Figure 4: Test bed.

The base Linux installation for each node was Debian 3.1 Sarge. The hosts, sender1, sender2, receiver1, and receiver2, each had a few different patch levels of the Linux kernel version 2.4 installed on them to meet the requirements of the various high-speed TCP patches. The XCP end host patch was for Linux kernel 2.4.20. The BIC TCP patch was for Linux kernel 2.4.25. The H-TCP patch was for Linux kernel 2.4.23. High Speed TCP was included in the web100 package for Linux 2.4, so did not require a separate patch level of the Linux kernel. NewReno TCP and High Speed TCP were both tested with Linux kernel 2.4.26. All of the kernels except Linux 2.4.20 were built with web100 instrumentation. Each of these Linux kernel 2.4 patch levels was built with the appropriate high-speed TCP patch applied and installed as separate kernel images on each end host. Each machine would be rebooted to the appropriate kernel image for each set of tests.

On the router, Linux kernel 2.6.6 was installed, as this was the kernel version that the XCP router kernel module was developed and tested for. The bridge had Linux kernel version 2.6.15 so the network emulator, netem, could work properly.

It is well known that the default settings for TCP in all major operating systems, including Linux, are not optimal for achieving good performance in high-speed networks. Therefore, the guidelines for TCP tuning from the TCP Tuning Guide were followed [Tierney06]. Different parameters for TCP can be set with the sysctl system in Linux. The sysctl tool modifies kernel parameters at runtime. Tcp_timestamps and tcp_window_scaling are required for high-speed networks. They allow for large enough

window sizes in high-speed environments. They are on by default. Sack is well known to provide performance improvements over NewReno TCP, and the tcp_sack parameter is on by default.

High-speed network contexts require larger buffer sizes for TCP. The size of the congestion window is related to the amount of buffer space each socket has allocated from the kernel. If the buffer size is too small, the congestion window cannot open up as much as it should. Larger buffer sizes allow the congestion window to grow to fill the larger pipes in a high-speed network. Therefore, the TCP buffer sizes were reset to values appropriate for a 100 Mbit network. The parameter names and new size in bytes are listed below in Table 2.

net.core.rmem_max (max buffer size)	16777216
net.core.wmem_max	16777216
net.ipv4.tcp_rmem (min, default, max)	4096 65536 16777216
net.ipv4.tcp_wmem	4096 87380 16777216

Table 2: Network kernel parameters.

Another thing that can be configured to increase TCP throughput is the txquelen, or the size of the interface queue, as suggested by the TCP Tuning Guide. The recommendation was to set it to 1000, which was done to all hosts with the ifconfig command.

All of the hosts were instrumented with web100 [Web06]. The web100 software is a set of instruments for the TCP/IP stack in the operating system kernel. It collects statistics on all TCP events. We chose to use tcpdump to measure the experimental flows instead of web100 because of difficulties in getting web100 to work with the XCP hosts. The web100 software was still used, though, because it provided an implementation of HS-TCP.

The two sender hosts and the router were connected to a 10/100 Mbit switch, all communicating at 100 Mbps. A crossover cable connected the router and bridge. The bridge and the two receiver hosts were connected to a 10/100 Mbit switch. The paths into and out of the bridge were set to 10 Mbps to create a bottleneck, allowing congestion to build up at the interface at the router leading to the bridge. It was important to allow congestion to occur to elicit the congestion control behavior of the various protocols. The primary differences between the different protocols being tested are the congestion control algorithms.

The router with Linux kernel version 2.6.6 was set up to act as a router by turning on IP forwarding. In Linux, setting the sysctl variable net.ipv4.ip_forward to 1 does this. The default queue in Linux is a drop-tail, or FIFO, queue. This was kept as the queue as it is the most common type of router queue seen in the Internet. When the XCP tests were run, the XCP router kernel modules were loaded.

The bridge was required to pass on the XCP packets without installing a second XCP router, as all routers along an XCP path must be XCP-enabled. Bridges forward packets at the data link layer, bypassing IP. Also, the bridge served as the bottleneck to build congestion along the path by setting both its interfaces to 10 Mbps with the Linux utility mii-tool. The network emulation tool, netem, is built into the Linux kernel, and it was used to emulate various network conditions for the tests [Netem06]. Netem can inject delay so as to emulate realistic network paths such as the Internet or satellites. It can also inject loss to emulate realistic levels of loss such as are seen in the Internet or on wireless links.

Many tests were run on the network to determine the optimal settings for the high-speed TCP tests. The traffic generator and measurement tool, iperf, was used to run these preliminary tests. The bandwidth delay product of the network can be used to determine the most appropriate window size to use when running tests. With a 10 Mbps network (the bandwidth at the bottleneck) and a 102 msecs RTT (measured with ping after netem injected 100 msecs of delay at the bridge), the following is the bandwidth delay product of the network:

BDP = Bandwidth x RTT = 10 Mbps/8 bits * .102 s = 127,500 bytes

A 128,000 byte window was used as a starting point for experimenting with window sizes to find the size that would give optimal performance. A number of tests were done with various window sizes, and the best performance was achieved with a 128,000 byte window at the sender. The receiver's window size was set to 16 MB so that it would not be a limiting factor.

Experiments were to be run with a 14 msecs RTT (measured with ping after netem injected 10 msecs of delay at the bridge) on the same network, giving the following bandwidth delay product:

10 Mbps/8 bits * .014 s = 17,500 bytes

Window sizes were experimented with at 14 msecs delay, starting at 17,500 bytes. A 32,000 byte window was found to give optimal performance. Therefore, tests were run with both 32,000 byte windows and 128,000 bytes windows.

The maximum bandwidth utilization was reached by using five streams at the same time. The tool to generate traffic and measure throughput for XCP, a modified version of ttcp, is unable to generate multiple streams, though, so the tests were limited to a single stream at a time.

6.3 Description of Experiments

Tests were designed according to the recommendations of the TMRG and Wei. [Floyd05B, Wei05]. The goal was to measure performance of the various high-speed TCPs in a high-speed network context with network conditions as in the current Internet. Performance was measured in terms of average throughput, packet loss, and delay. As many high-speed TCPs were measured as possible against XCP. The set of protocols to be studied created some limitations. For example, only single flows could be measured because the tool for generating traffic for XCP only supported single streams. Also, it was problematic to include background traffic because only XCP flows could go through the XCP router.

Traffic was generated with a modified version of ttcp. Ttcp creates a flow of socket-based TCP traffic and measures throughput. It reports goodput, successfully transmitted data. Also, it measures user data, not including headers. It does memory to memory data transfers. The modified version supplied with XCP for Linux had an XCP mode and regular TCP mode.

Ttcp was started on the receiver host as a sink with a 16 MB window. Then ttcp was started on the sender side as a source to connect to the receiver machine with a time length of 60s and initial window size of 128,000 bytes. Another set of tests was run with initial window size of 32,000 bytes. Tests were run for 60s to allow time for TCP to achieve steady state. A longer time would have been preferable, but memory limits on

the test machines appeared to cause problems with longer tests at low delay. In fact, tests run with 14 msecs delay would not run for longer than 55s. Nonetheless, tests run for five minutes at high delay did not seem to have very different results from the 60s tests.

Realistic network conditions were emulated with netem at the bridge. Tests were run with added 10 msecs, 100 msecs, and 500 msecs delay. This was split between the two interfaces on the bridge; so 10 msecs was added as 5 msecs on interface eth0 and 5 msecs on interface eth1. With the delay added, ping was run on sender1 to receiver 1, and RTTs for 10 msecs added delay averaged an actual total 14 msecs, 100 msecs averaged 102 msecs, and 500 msecs averaged 502 msecs. Loss was also injected with netem at the bridge. Tests were run with 0.001%, 0.1% and 1% loss combined with 100 msecs delay. Each test was run five times, and results were averaged.

All tests were repeated for each high-speed transport protocol. At the end of a set of tests with one protocol, the host computer was rebooted into the kernel image compiled with next protocol patch. For the XCP tests, the XCP kernel router module was also loaded, and an XCP qdisc was added to the network interface connected to the bridge with the following command:

tc qdisc add dev eth1 root xcp capacity 10Mbit limit 500

6.4 Results

While the tests were being run, the packet capture tool tcpdump was run to monitor the flows. The packet capture library used by tcpdump, libpcap, drops packets at high speeds, so a modified version of libpcap was used which supports MMAP mode in the Linux kernel. Average throughput was reported by ttcp. The packet capture dump files were analyzed by the tcptrace tool [Tcptrace06] .

6.4.1 Throughput

Table 3 shows the average throughput reported by ttcp, averaged over five runs of each test. Figures 5 and 6 show the average throughput for 128,000 byte windows with the different values of loss and delay emulated by netem. The average throughput is actually average goodput (successfully delivered packets), because that is what ttcp reports.

	TCP	BIC	HS-TCP	H-TCP	XCP
Delay					
14 msec	3523.132	2405.468	3528.976	3624.134	3377.954
102 msec	1905.032	1593.166	2058.978	1129.838	2652.072
502 msec	923.946	298.452	332.152	146.81	342.59
Loss					
0	1905.032	1593.166	2058.978	1129.838	2652.072
0.001	2081.214	1548.918	2061.308	1091.904	2342.266
0.100	1473.22	1403.12	1508.45	1134.52	2447.02
1.000	966.63	1012.644	1031.442	882.384	2673.926

Table 3: Average throughput reported by ttcp. Units are Kbit/sec with a 128,000 byte window.

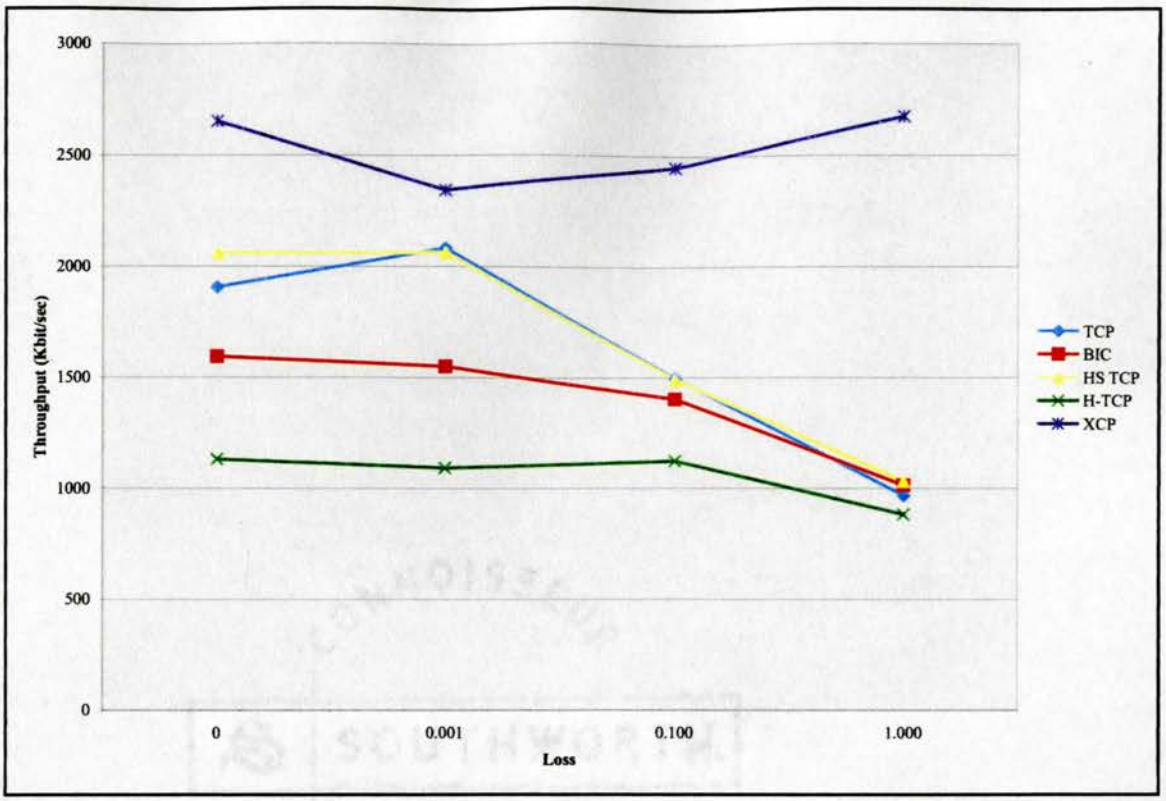

Figure 5: Average throughput with different levels of loss for a 128,000 byte window.

Results show XCP outperforming the other protocols in most instances. NewReno TCP also does well. The performance of NewReno TCP degrades as delay or loss increases, as is expected. That is why it has problems on high BDP networks. XCP performs well with increasing levels of loss, but is affected by increasing delay.

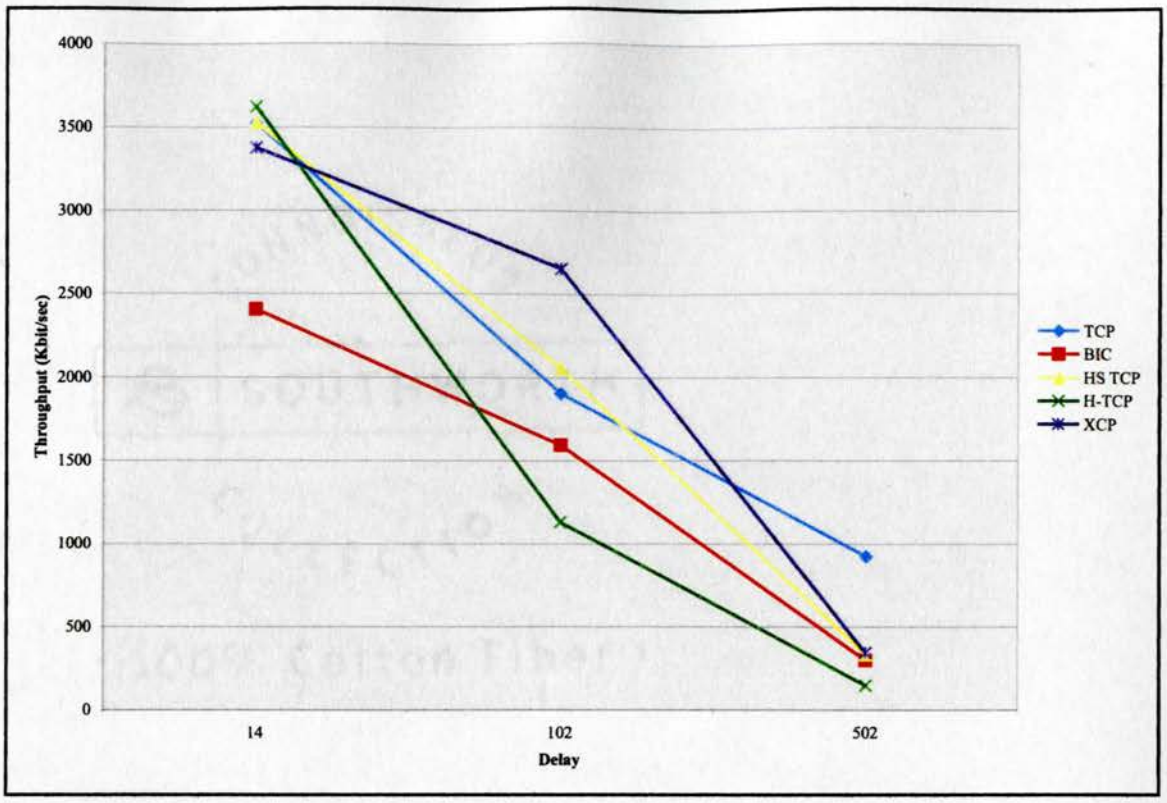

Figure 6: Average throughput with different levels of delay for a 128,000 byte window.

The effect of loss can be seen in Figure 5. XCP is almost unaffected by increasing levels of loss. Loss affects NewReno TCP and High-Speed TCP the greatest of the protocols. The performance of BIC TCP and H-TCP does not degrade as significantly as NewReno TCP and High-Speed TCP.

The effect of delay can be seen in Figure 6. Increased delay affects the performance of all of the protocols. H-TCP is most greatly affected by increasing delay. XCP is

surprisingly affected negatively by increased delay. XCP is supposed to perform well on long delay networks.

An XCP flow with 10 msecs injected delay and no injected loss is shown in the time sequence graph in Figure 7. The time sequence graph shows all major events over the lifetime of the flow. The red Rs mark retransmissions. The green 3s are triple duplicate ACKs, which typically trigger TCP fast retransmit. These are difficult to differentiate in the graph since they overlap. They overlap since retransmissions happen as the duplicate ACKs are received. Each of these events indicates packet loss. There are few packet losses, especially once convergence occurs, as would be expected with an XCP flow.

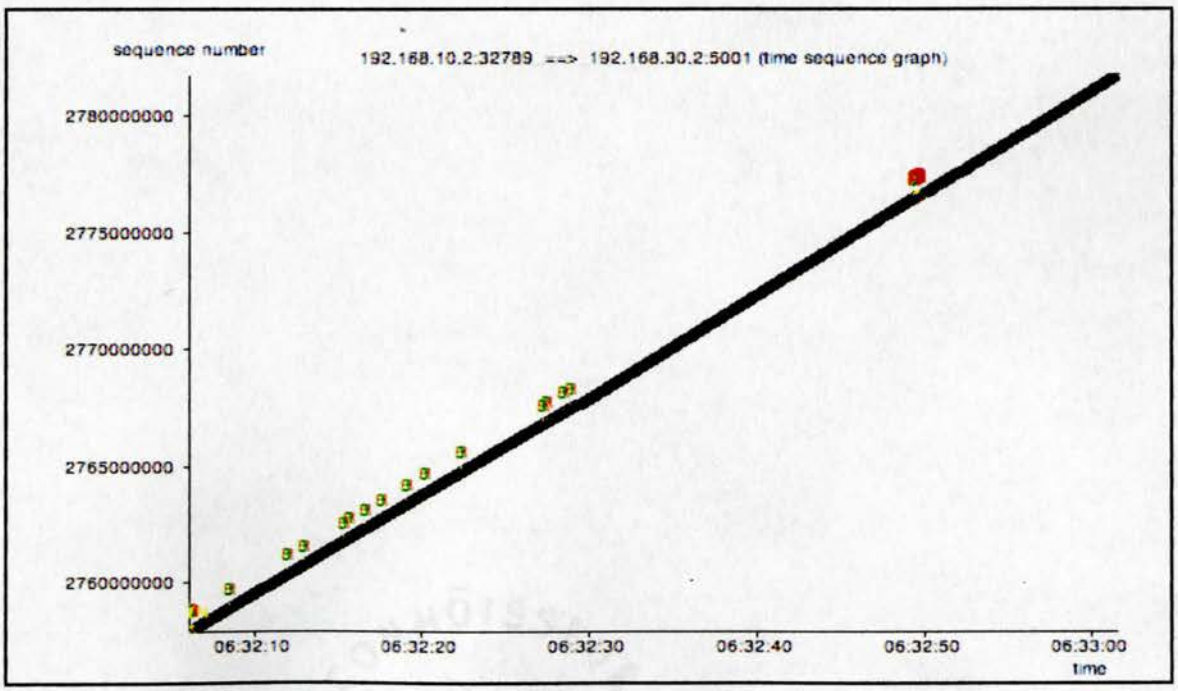

Figure 7: XCP flow with few retransmissions.

A NewReno TCP flow with 10 msecs injected delay and no injected loss is shown in the time sequence graph in Figure 8. Many retransmissions are seen at the beginning of the flow but then taper off once steady state is reached.

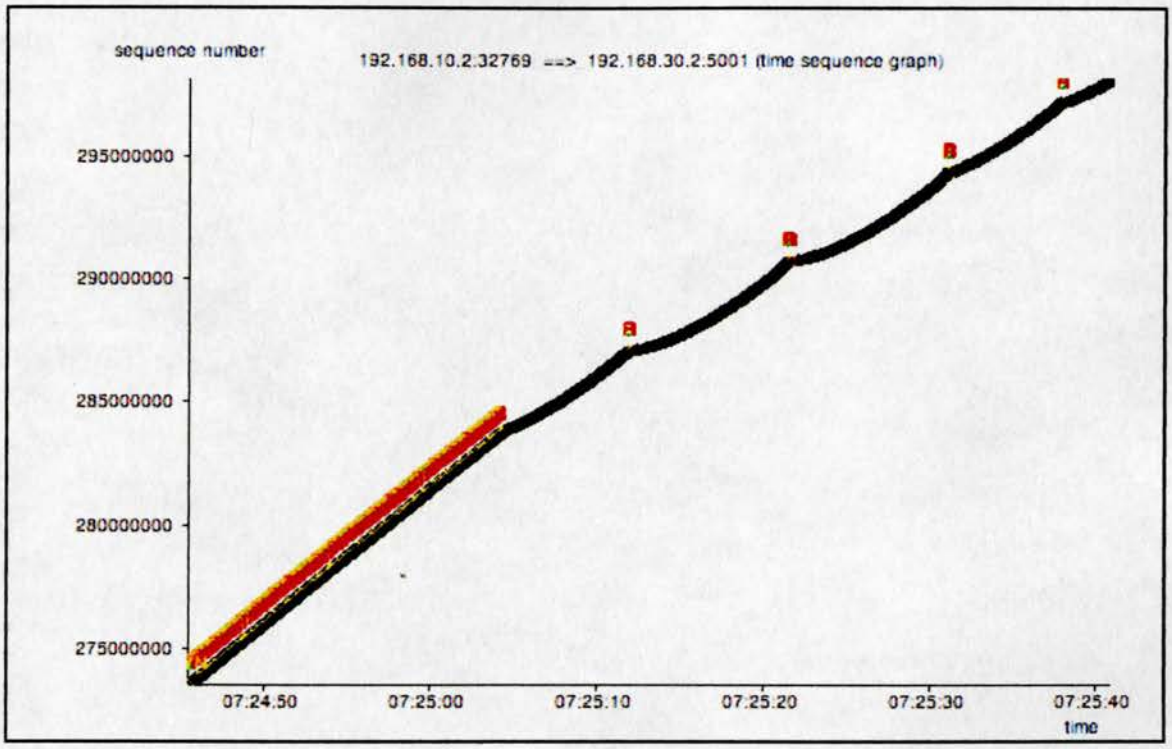

Figure 8: TCP flow with many retransmissions.

The outstanding data graph of the same TCP flow is seen in Figure 9. An outstanding data graph provides an estimate of cwnd over time (the red lines). It shows the typical TCP behavior of entering slow start after packet loss. The window size increases exponentially until packet loss causes the window size to drop back to one MSS. Steady state is reached about halfway through the flow and congestion begins to be avoided.

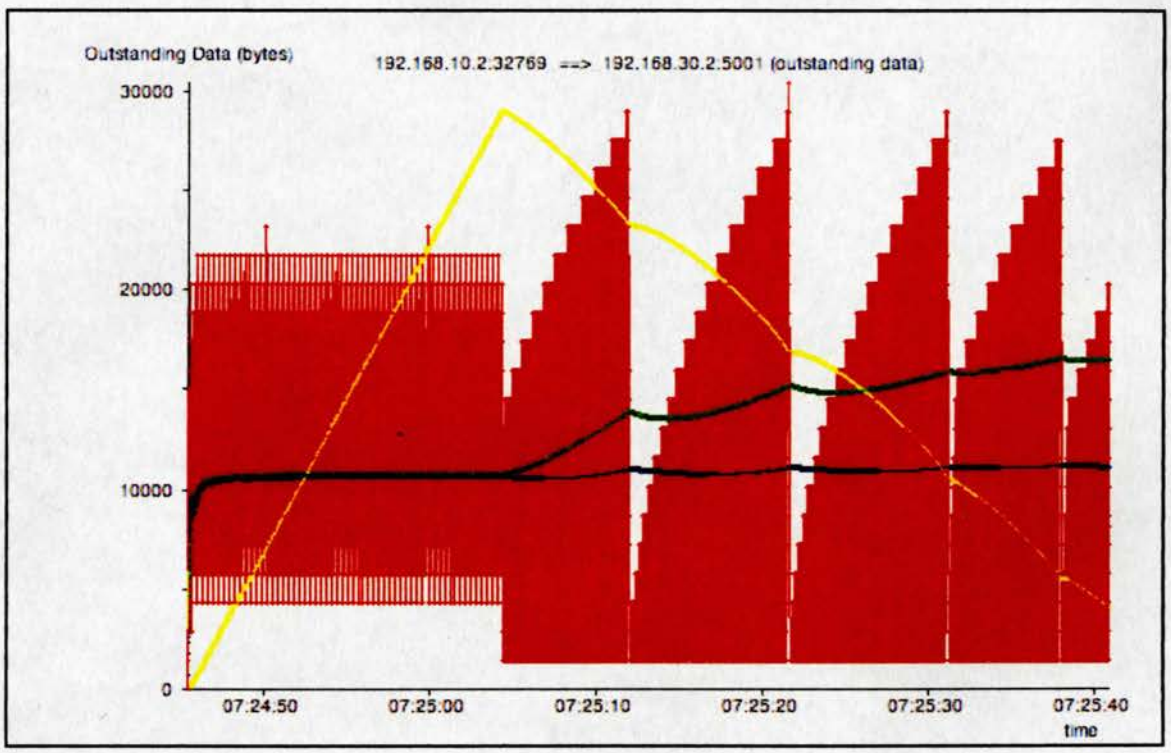

Figure 9: TCP flow avoiding congestion.

The behavior of BIC TCP can be seen in the outstanding data graph in Figure 10. It is from a flow with 100 msecs injected delay, which shows the behavior of the protocol more clearly. The first half of the flow is the additive increase phase, and the second half is the binary search phase. The binary search phase appears to drop the congestion window fairly low. BIC is intended to be TCP friendly and RTT fair, so it is supposed to be less aggressive. It reaches equilibrium quickly then backs off which also helps it avoid self-inflicted losses.

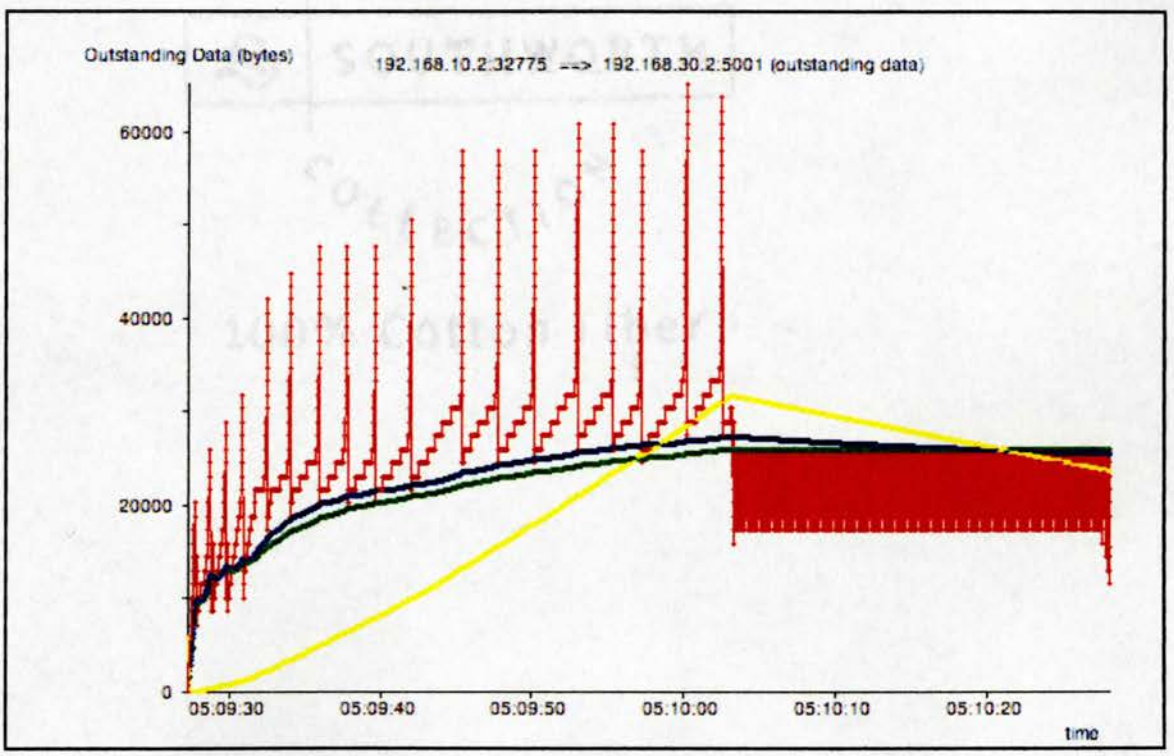

Figure 10: BIC TCP flow.

The behavior of HS-TCP can be seen in the outstanding data graph of Figure 11. It is a flow with 100 msecs injected delay and zero injected losses. The greater increase factor and lower decrease factor relative to NewReno TCP can be seen throughout the flow.

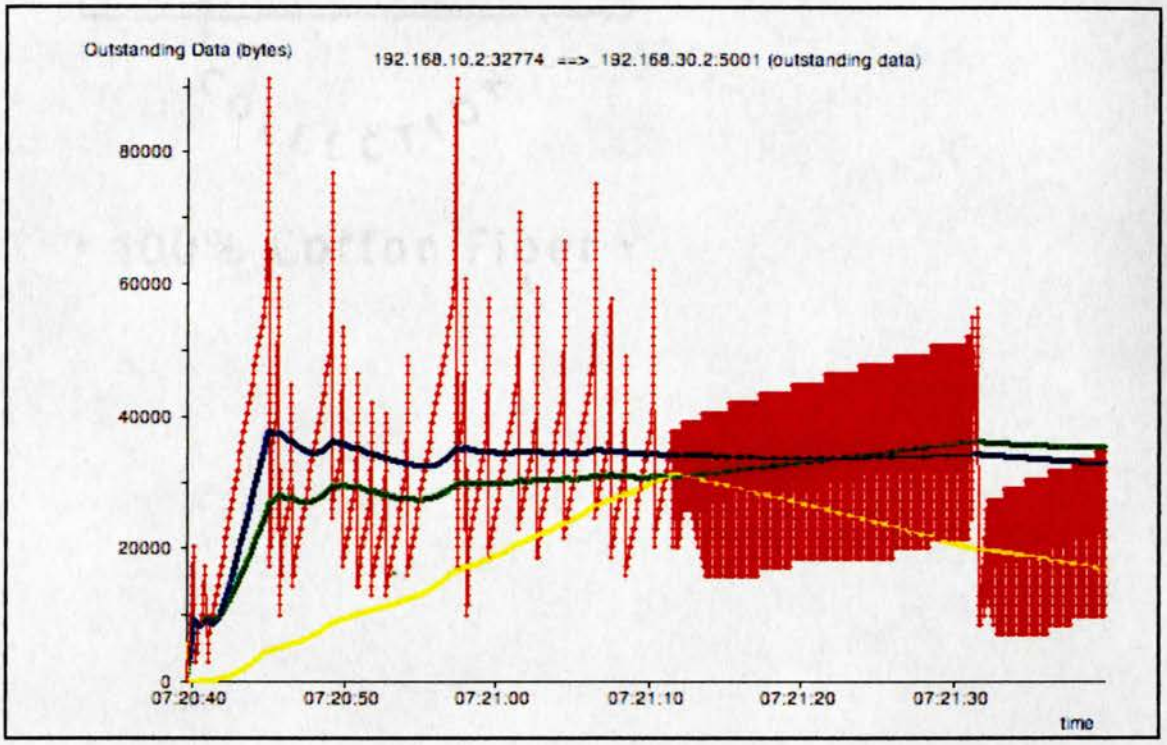

Figure 11: HS-TCP flow.

The behavior of H-TCP can be seen in the outstanding data graph of Figure 12. Compared to the NewReno TCP tests, the increase factor appears to be greater when the window size reaches about 28,000 bytes. This corresponds to the H-TCP specification. The decrease factor does not appear to be less severe than NewReno TCP, though.

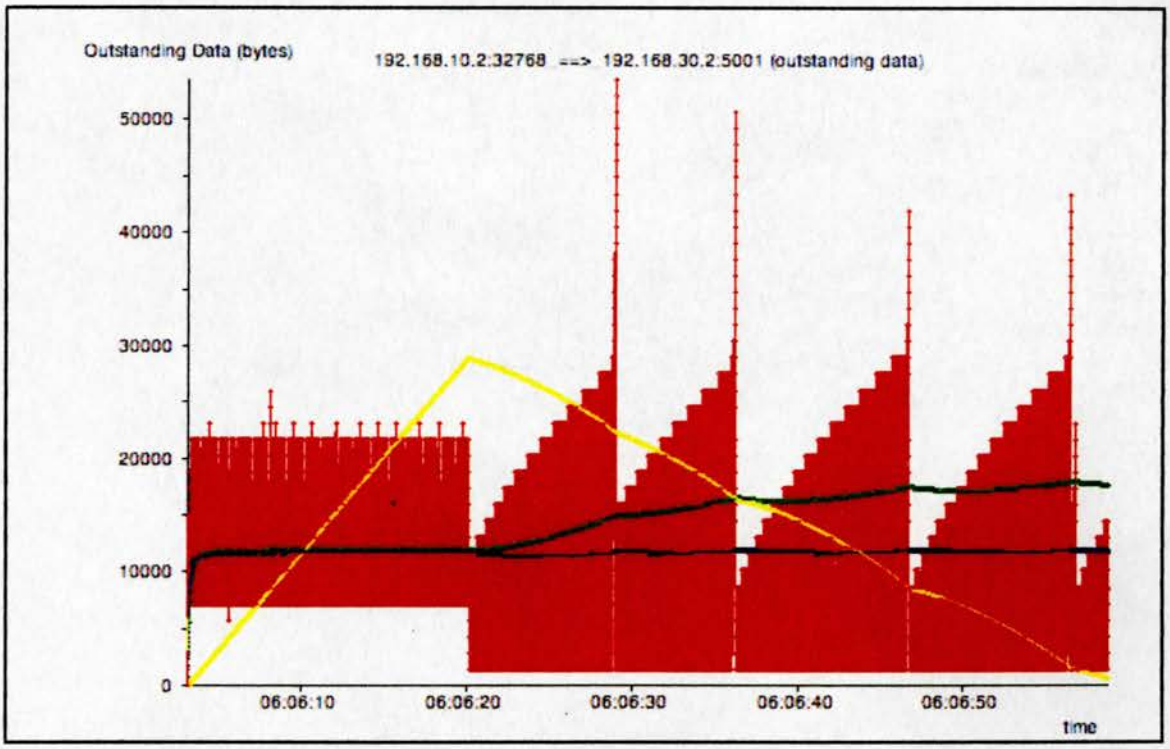

Figure 12: H-TCP flow.

6.4.2 Packet Loss and Delay

Figure 13 shows the measured delay from packet traces with 100 msecs of injected delay and 0%, 0.001%, 0.1%, and 1% levels of injected loss. XCP and H-TCP have the lowest delay. They are also not as affected by increasing loss. NewReno TCP and HS-TCP have the highest delay. Delay drops off for NewReno TCP, BIC, and HS-TCP as loss increases.

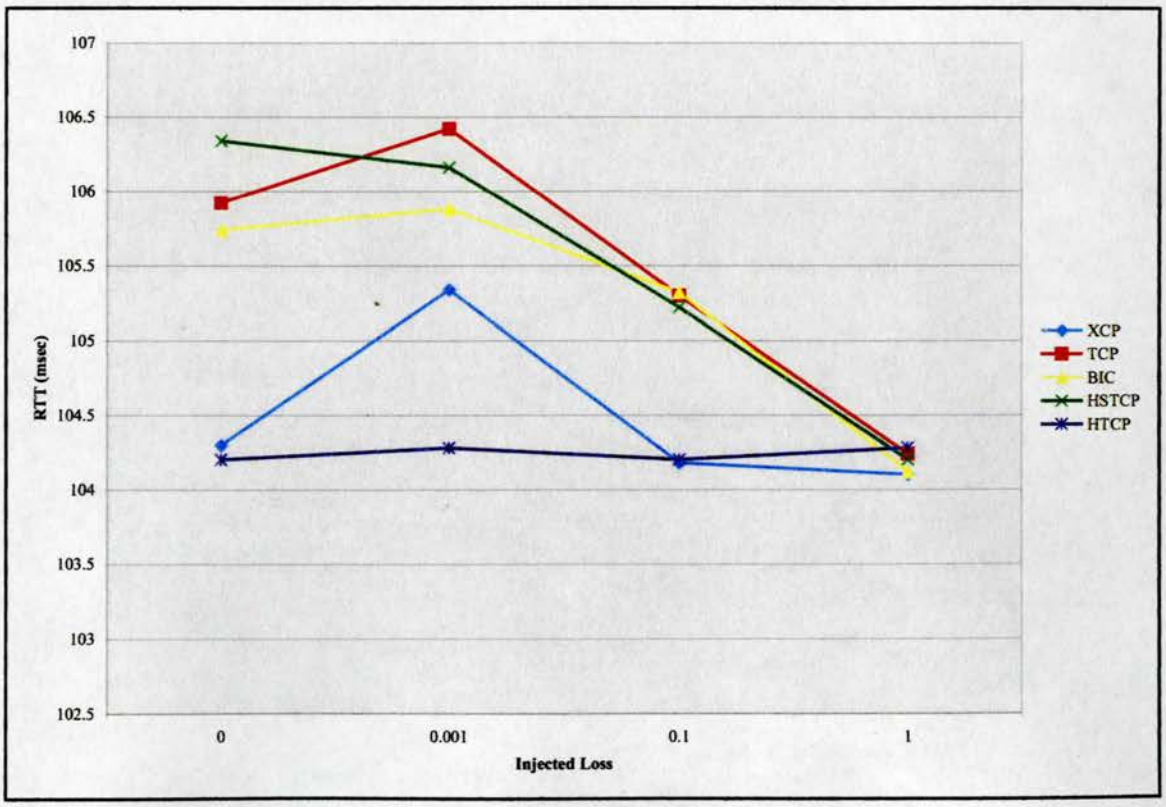

Figure 13: Delay with injected loss.

Figure 14 shows measured levels of packet loss from packet traces with varying levels of injected delay. Packet loss does not increase significantly for NewReno TCP, High-Speed TCP, and BIC TCP with increasing delay. H-TCP and XCP both have increasing packet loss with increased delay. It appears the increased delay injected with netem throws off XCP's control laws. This explains the decreased average throughput for XCP as delay increases seen in Figure 6. One possible explanation for this behavior could be bursty sending of data causing an incorrect estimation of RTT by XCP. Bursts of data would cause a queue buildup at the XCP router, which would increase delay. This would throw off RTT estimations, and the error would increase as the amount of delay increases. The seesaw cwnd seen in Figure 15 shows burstiness, supporting this theory.

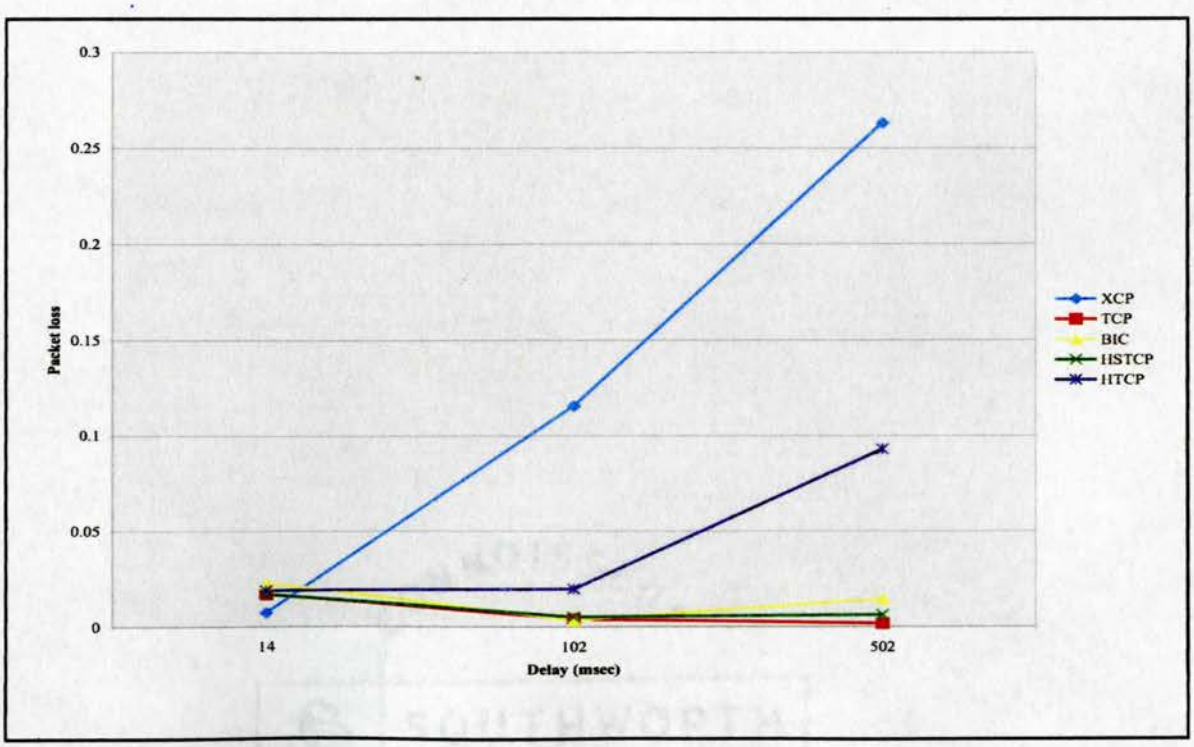

Figure 14: Packet loss with injected delay.

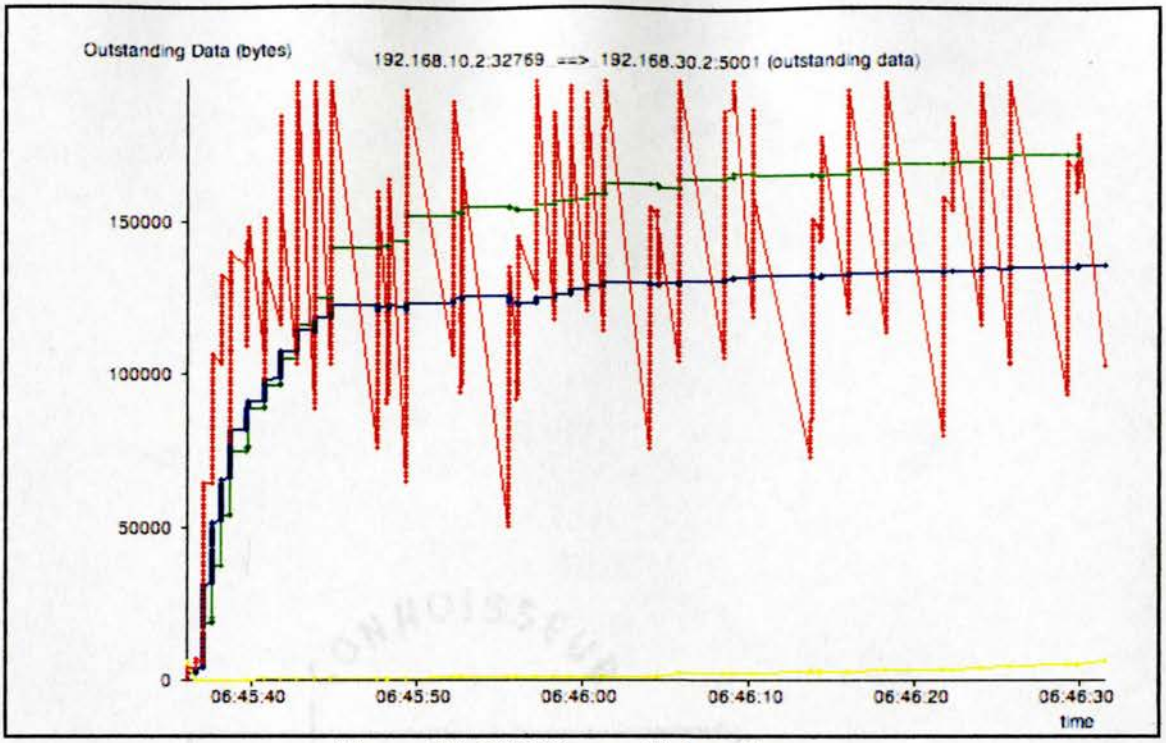

Figure 15: XCP flow with burstiness.

6.4.3 XCP Sensitivity to Misconfiguration

Tests were originally run with the capacity of the XCP router set to 10 Mbit, the line speed of the bottleneck seen in Figure 4. Results were as seen in Figures 16 and 17. XCP is clearly outperformed by all of the other protocols.

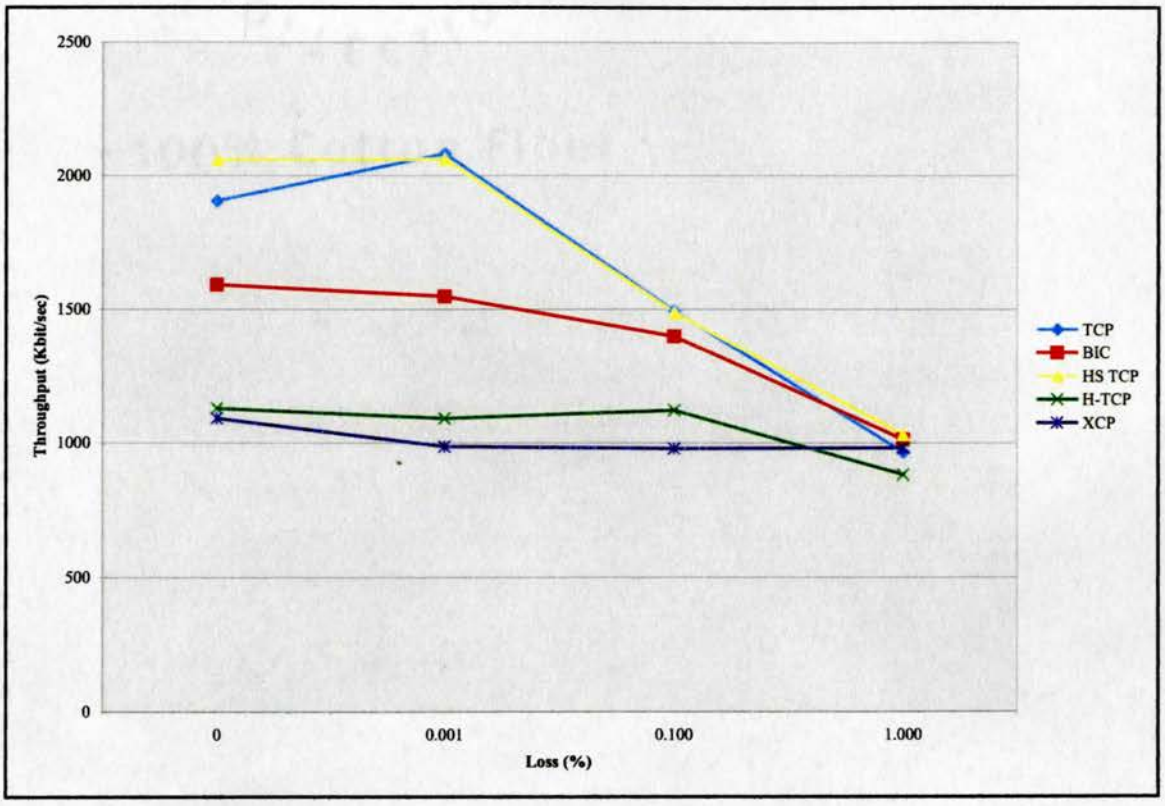

Figure 16: Average throughput with misconfigured XCP router with different levels of loss for a 128,000 byte window.

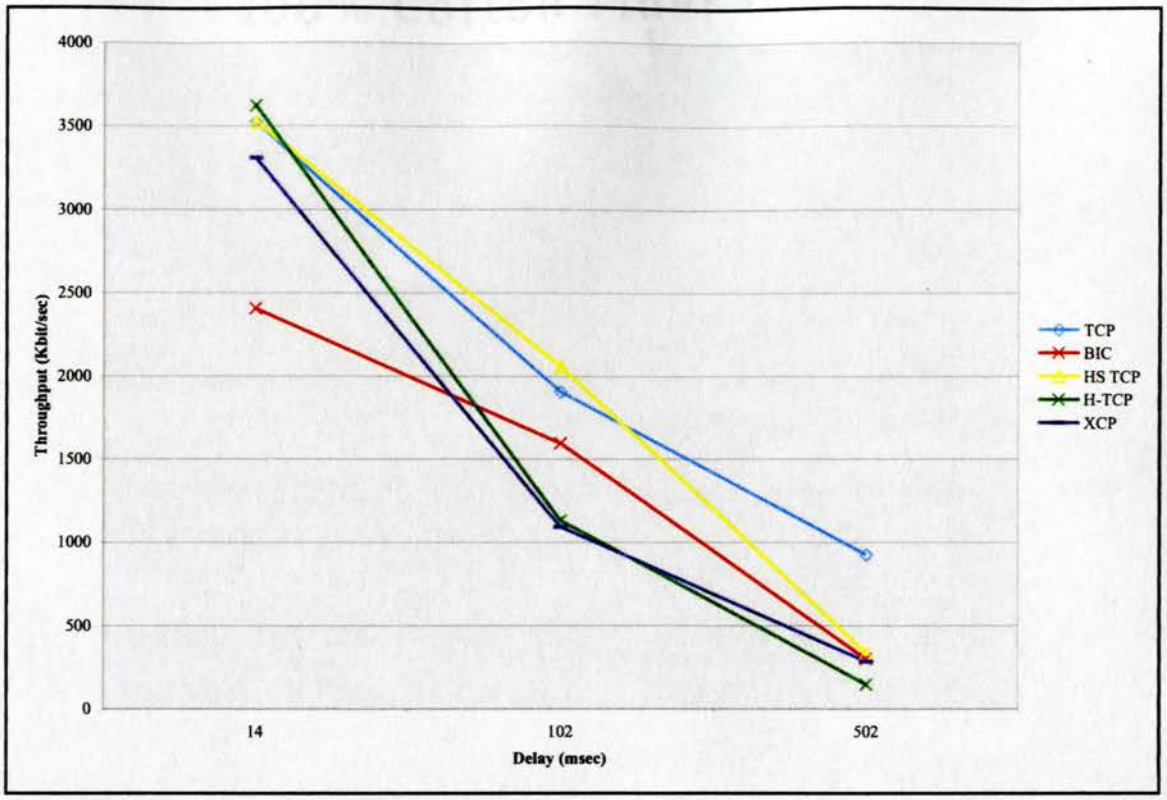

Figure 17: Average throughput with misconfigured XCP router with different delay values for a 128,000 byte window.

The packet traces of the tests were analyzed to determine what was happening during the XCP tests to cause such poor performance. Figure 18 shows a slice of a time sequence graph of a representative XCP flow with 10 msecs injected delay and zero injected losses. It shows sequence numbers on the Y-axis and time on the X-axis. All significant events over the course of the connection are shown in the graph.

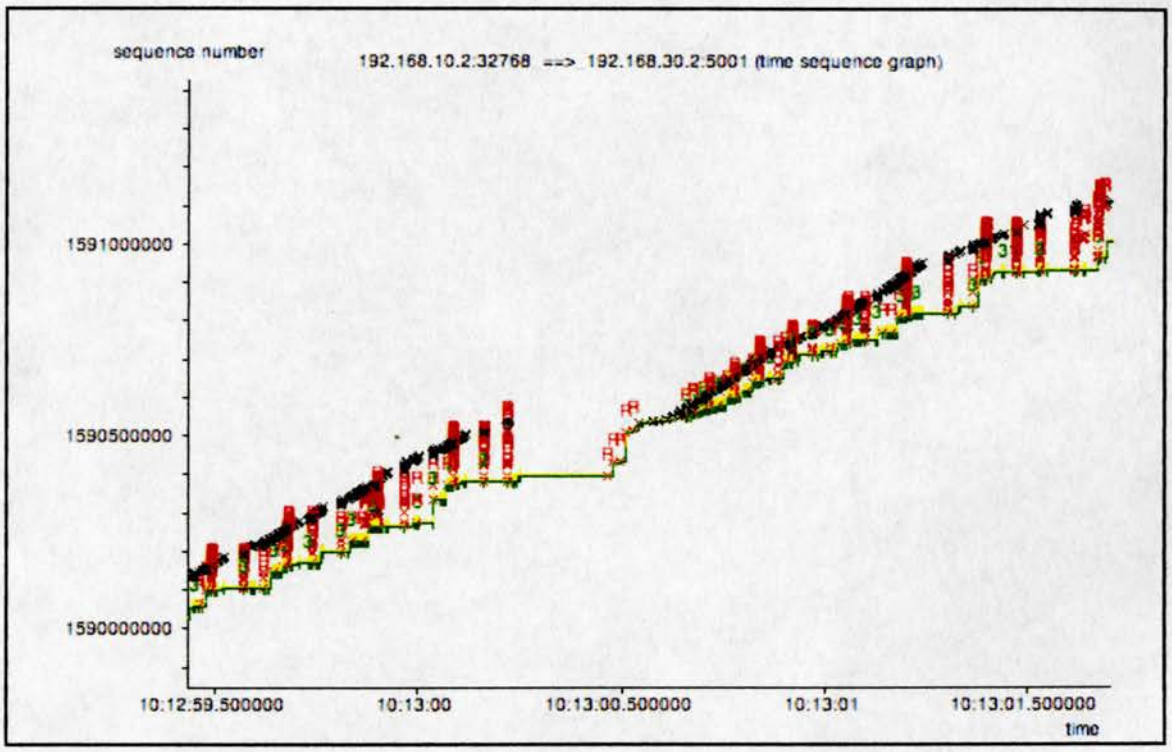

Figure 18: XCP flow with many retransmissions.

The red Rs in the graph show retransmissions. The multiple retransmissions continue over the entire lifetime of the flow. Figure 19 shows the outstanding data graph for the same connection over the entire lifetime of the flow. This graph gives an estimate of the congestion window over the course of the flow. The congestion window grows to almost 200000 bytes and then shrinks back down to zero continually.

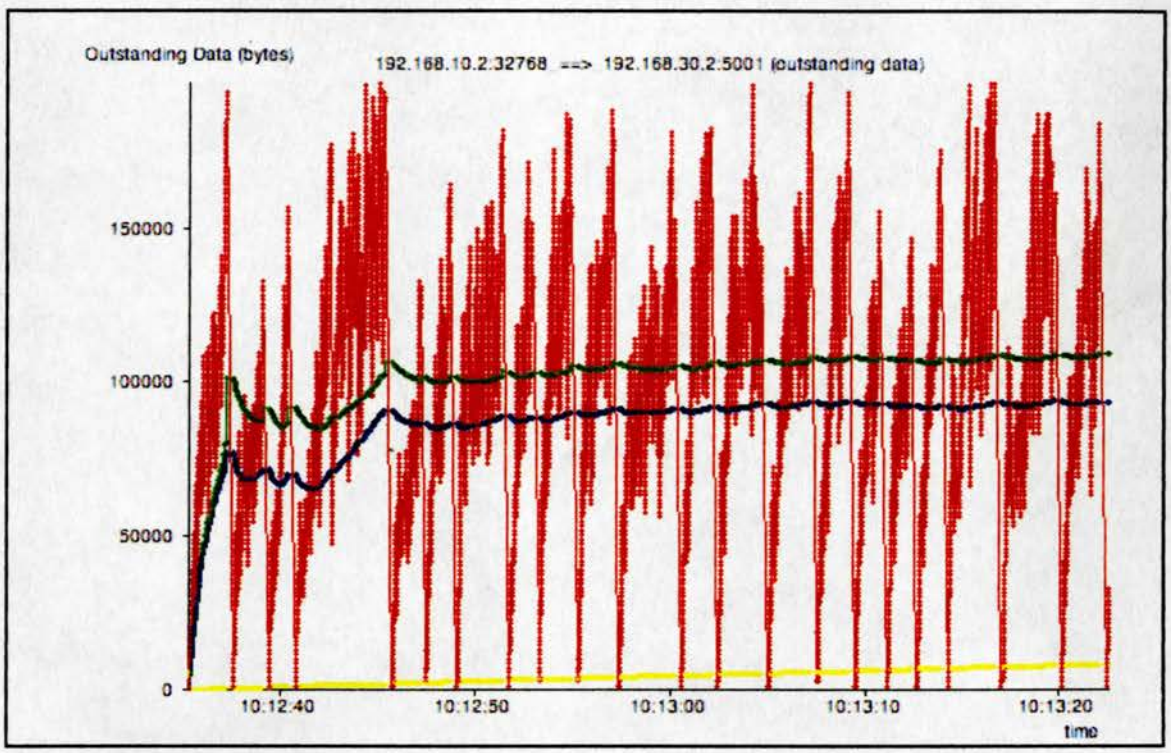

Figure 19: XCP flow with oscillating congestion window.

XCP's control laws are clearly being thrown off. Investigation revealed that the XCP router was misconfigured. Zhang encountered the same issue in his tests with XCP for Linux [Zhang05]. The capacity setting in the XCP router must account for protocol overhead. It should be set to lower than the line speed of the network. The capacity of the XCP router was lowered, and the results seen in Figures 5 and 6 were achieved. Overestimating the capacity of the network causes XCP to assign more bandwidth to flows than is actually available. Then the sender increases cwnd by more than the network can handle, causing packet drops, as was seen in Figure 18. Figure 20 shows the outstanding data graph of an XCP flow with 10 msecs injected delay after the configuration problem was fixed. The burstiness in the flow is markedly reduced.

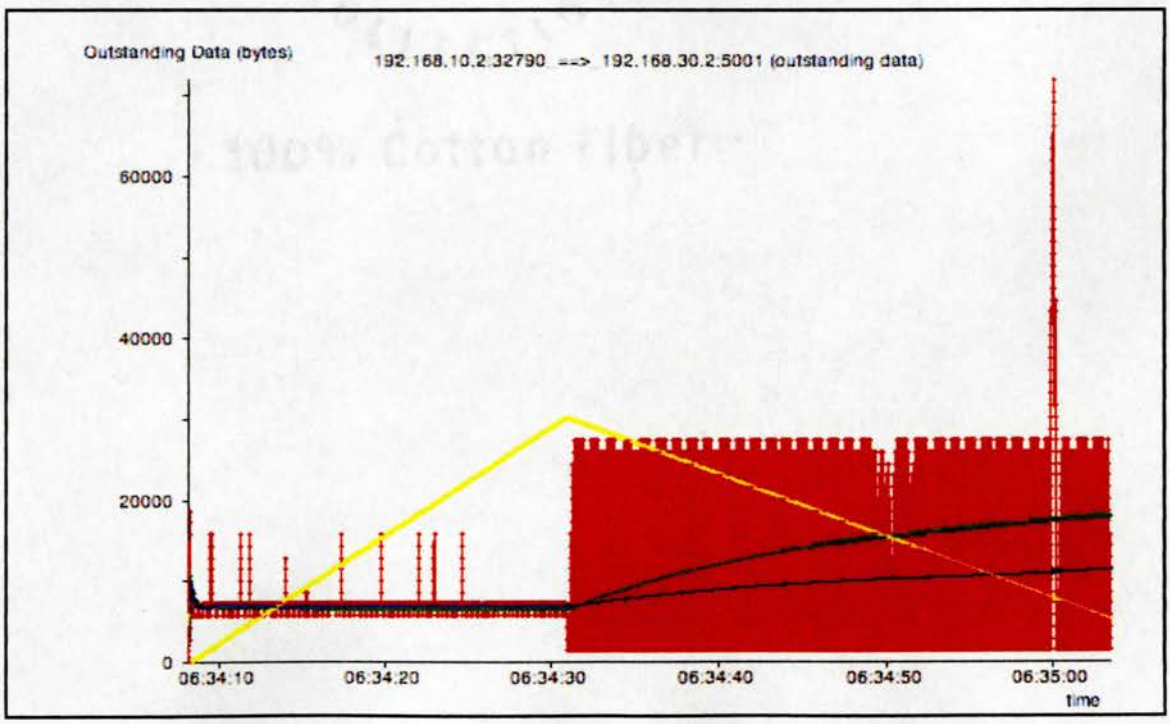

Figure 20: XCP flow with less burstiness.

6.4.4 Effect of Smaller Windows

32,000 byte windows were found to give good performance at 14 msecs delay, so all tests were run with 32,000 byte windows as well as 128,000 windows. 32,000 byte windows were sub-optimal for 102 and 502 msecs delay. None of the protocols achieve as high performance with smaller windows, but XCP seems to be affected the worst. Figure 21 shows a test with varying levels of injected delay with a 32,000 byte window. Figure 22 shows a test with 100 msecs injected delay and varying levels of loss with a 32,000 byte window. XCP cannot reach 700 Kbit/sec throughput. XCP cannot grab spare bandwidth with the smaller windows.

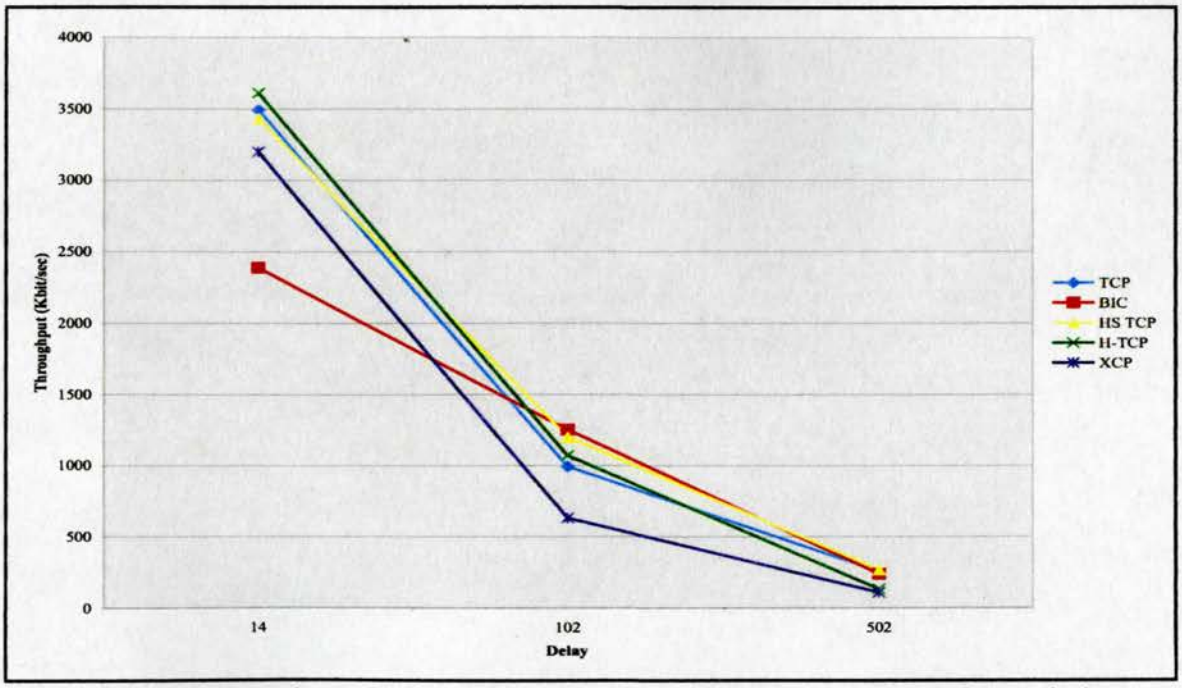

Figure 21: Average throughput with varying levels of delay with 32,000 byte windows.

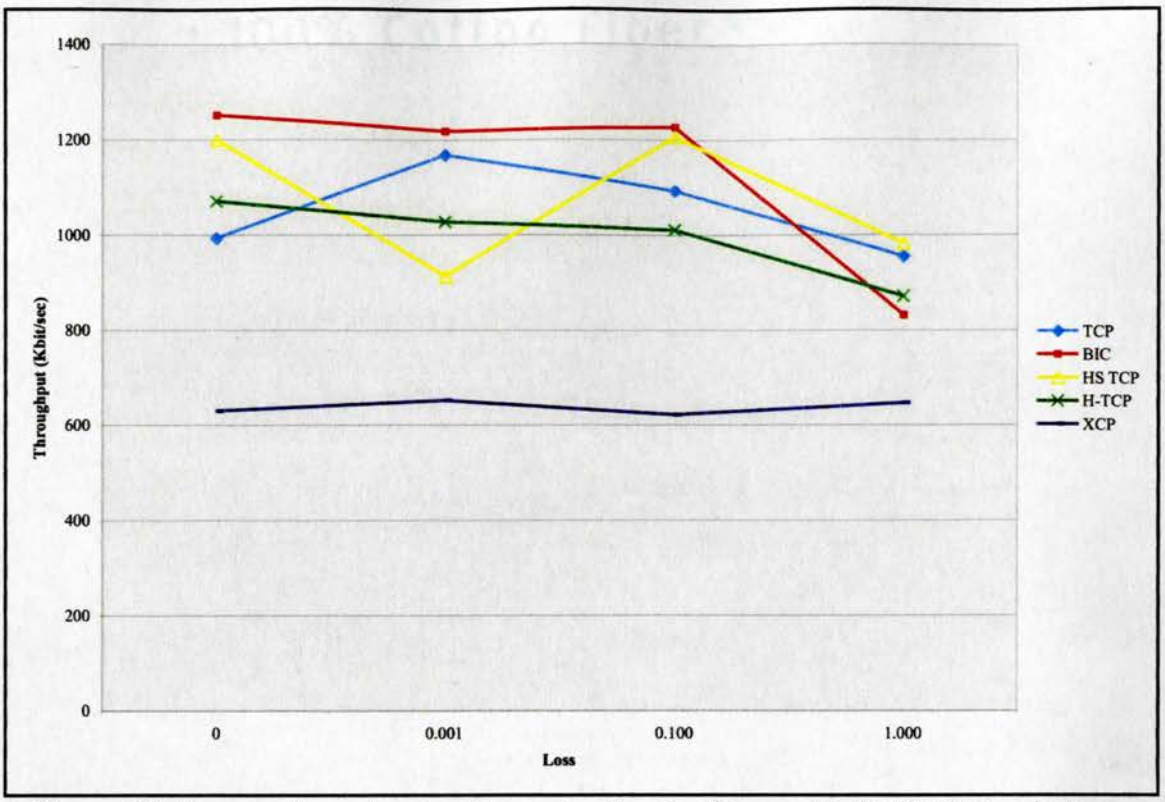

Figure 22: Average throughput with varying levels of loss with 32,000 byte windows.

6.5 Comparison to Other Studies

Rhee [Rhee06] studied the effect of background traffic on the various high-speed transport protocols. They tested flows without background traffic and with background traffic. Without background traffic, as in our experiments, with increasing amounts of delay, utilization was the best for HS-TCP, followed by NewReno TCP, then BIC, and then H-TCP. This is the same as our results seen in Figures 5 and 6. With background traffic, their results were different. Utilization was best for BIC, followed next by HS-TCP, then NewReno TCP and H-TCP. BIC, HS-TCP, and NewReno TCP do much

better at higher delays with background traffic, as well. This highlights the impact background traffic has on these types of tests. The control laws that govern the protocols work differently in the presence of background traffic. Proper models of background traffic are needed so that they can be integrated into protocol evaluations [Rhee06].

Cottrell was able to elicit performance improvements over NewReno TCP from some of the high-speed transport protocols tested in our study. Their results for the closest window size to what we used, 256 KB, were as follows: BIC TCP and NewReno TCP had the best throughput, followed by H-TCP, then HS-TCP. This is the same as the trend seen in our Figures 5 and 6, except BIC did better in their tests than ours and HS-TCP did worse than in ours. They also found the performance of the protocols showed more of a difference at longer distances, which is backed up by our tests with increasing delay. The parameters of their tests were fairly different from ours, though, so it is difficult to make a good comparison. Also, they used a production network, which is a very different context than the test bed we used. This points to the fact that protocols can behave very differently in different contexts and with different configurations. This underscores that tests must be done in simulation, test beds, and in production networks before protocols can be accepted [Cottrell03].

The Hamilton Institute study found performance improvements over NewReno TCP with some of the high-speed transport protocols, as well. Their results are almost the opposite of ours seen in Figures 5 and 6. BIC TCP had the highest throughput, followed by H-TCP, HS-TCP, and then NewReno TCP. Their tests were with two competing flows,

while our tests were with single flows. Also, they were measuring throughput as a function of queue size, while we were testing against loss rates and delay [Leith05].

Zhang and Henderson found their XCP for Linux gave good utilization, low packet drops, and small queues. It outperformed TCP, as it did in our tests as seen in Figures 5 and 6, under controlled conditions. When environmental factors were varied, such as TCP/IP configuration, link contention, and non-congestion losses, performance dropped significantly. For example, if buffers are limited by memory shortage, the XCP routers can increase cwnd. Then when the buffers increase, the sender may be able to send more than the network can actually handle. This is a problem with the XCP control laws. They discovered the same issue revealed in our tests, as seen in our Figures 16 and 17. If the XCP router is even slightly misconfigured, such as having the capacity set too high, it throws off the XCP control laws, and XCP performs poorly. They went on to test situations where the capacity of the network could vary such as in links with contention, showing it can be very difficult to correctly estimate the capacity of a network. They also ran tests with small window size (64 KB) and large window size (640 KB) and found XCP had much worse utilization with the smaller windows, confirming what we saw in Figures 21 and 22 [Zhang05].

Chapter 7

CONCLUSION

A set of experiments was run to evaluate the performance of XCP, BIC TCP, HS-TCP, and H-TCP against NewReno TCP in a network test bed. Network environmental factors such as delay and loss were emulated to create realistic network scenarios. Care was taken to keep network stacks as uniform as possible between the tests of the different protocols. All tests were run on Linux kernel 2.4. The TCP/IP stack was tuned for high-speed network performance. Traffic was generated by a modified version of the ttcp tool. Tests were run with single stream 32,000 byte and 128,000 byte windows for 60s each.

XCP performed the best of all of the high-speed protocols in a controlled environment. It gave the highest average throughput in most instances when it was configured correctly and proper window sizes were used. Nonetheless, a simple router misconfiguration or small windows made XCP perform much worse than the other protocols. A bursty sender also affected XCP negatively. All of the results confirm what other researchers have seen [Cottrell03, Rhee06, Zhang05].

None of the protocols beside XCP performed better than NewReno TCP. This highlights the difficulty in designing, implementing, and deploying a transport protocol. The

congestion control algorithms of NewReno TCP have been developed and tested in the real Internet for more than twenty years between its different versions. It has been proven to perform well and has improved over the years. It is stable and has prevented congestion collapse in the Internet. NewReno TCP does have performance problems in high-speed networks. It is overly conservative, inefficient, RTT unfair, and cannot recognize loss not due to congestion. Even so, newer high-speed transport protocols have a significant challenge to unseat TCP.

To design a new transport protocol, a number of sometimes competing factors must be considered. The measures outlined by the TMRG in their draft specification point to the laundry list of important considerations. The protocol must give acceptable average throughput, low delay, and low packet loss. It must be stable, fair, and TCP-friendly. The protocol should converge, be robust, and be deployable. It must also do this in many different environments, including challenging ones such as satellite or wireless networks. Given the design challenge involved in designing a transport protocol, it is amazing that TCP has performed as well as it has.

All of the high-speed transport protocols that were tested are experimental implementations. Work to improve them continues for all. All of the protocols except for XCP have newer versions for Linux kernel 2.6. In fact, as of Linux 2.6.13 each of the protocols beside XCP is included as part of the Linux kernel so congestion control algorithms can be swapped out at runtime. The paper by Rhee used the 2.6.13 version of the congestion control algorithms [Rhee06].

Work continues on XCP as well. The XCP group at the Information Sciences Institute (ISI) at USC is working on the XCP specification, and is developing a reference implementation for FreeBSD. Many changes have been made to the specification since XCP for Linux was developed. The ISI XCP FreeBSD implementation should be an improvement over XCP for Linux. For XCP to be properly evaluated against the other high-speed transport protocols, though, there will have to be an implementation for Linux. There are challenges to implementing the protocol for Linux, though, which Zhang and Henderson found when they implemented XCP for Linux. These challenges include a lack of floating point arithmetic in the Linux kernel.

None of the high-speed transport protocols performed significantly worse than NewReno TCP. This shows they are all, at least, certainly contenders against NewReno TCP. They have been shown to exhibit other behaviors that are preferable to NewReno TCP such as increased RTT fairness, better convergence, and better stability [Cottrell03, Leith05, Rhee06]. As bugs are worked out and work on their control laws and design continues, significant improvements over NewReno TCP may still be seen. Performance evaluations should be continued. Any protocol that is considered as a replacement for New Reno TCP must be thoroughly evaluated in many different scenarios before it could be accepted. As many have said before, simulations, test bed experiments, and real production network tests must be performed. Correct and thorough TCP models must be tested, taking into consideration both present and future environments [Floyd03, Allman99B]. It is a significant challenge to design, implement, and prove a new transport protocol.

7.1 XCP Deployment Issues

All of the transport protocols beside XCP were almost trivial to deploy. A patch was applied to the Linux kernel, and the kernel was rebuilt. Then when that transport protocol was to be run, the computer was rebooted to that kernel image. This was only necessary on the sending host. Typical TCP applications were run without any modification.

XCP, on the other hand, required changes to the sender and receiver hosts, the router, and required a special application that implemented the XCP socket options. If more routers had been added to the network path, they would have required the XCP module, as well. The network emulation machine had to be set up as a bridge to pass on packets so that it would not require the XCP router module. Getting the router modules compiled, loaded, and working was a non-trivial task. Configuring the router module to work correctly in the test bed environment also required some effort and was prone to misconfiguration.

The extra level of effort required to set up XCP in a simple test bed portends poorly for setting up XCP correctly in a real network environment. In Chapter 3, Katabi and Falk's ideas for XCP deployment were discussed. The level of effort to deploy XCP even in one network, let alone in the entire Internet, should not be minimized. Significant improvements in performance must be seen to justify the additional effort. XCP has quite a way to go before this will be seen.

7.2 Future Work

Additional tests that could be run that would give more useful data would be tests with background traffic. Rhee's experiments with background traffic show that background traffic significantly affects protocol behavior [Rhee06]. It would be useful to have models for background traffic to use in such experiments. Also, other metrics can be collected which fully describe protocol behavior, such as fairness, stability, and convergence.

Better equipment, more memory, faster CPUs, and the like, would probably have an impact on the results of these tests. It would be interesting to rerun the tests on an improved test bed and compare results. Running the experiments on a gigabit network would provide an interesting comparison.

All of the protocols beside XCP are included in Linux kernel 2.6. It would be interesting to rerun the tests with Linux kernel 2.6 and compare the results. If XCP could be implemented for Linux kernel 2.6, it could be included in these tests and results compared.

Others are working on using the ideas of XCP, but in a more realistic way. Stoica has proposed an alternative to XCP in their paper "One More Bit Is Enough." They propose a much simpler protocol called Variable-structure congestion Control Protocol (VCP)

that uses the existing ECN bits for congestion feedback. They have been able to achieve similar performance improvements to XCP. This would be significantly easier to implement and deploy than XCP [Stoica05].

There is continuing work on other high-speed transport protocols, as well. Performance evaluations not only test the value of current proposals but also help advance the work of developing new protocols in general as better understandings are formed of what is needed in high-speed transport protocols.

REFERENCES

Print Publications:

[Allman99A]
Allman, M., V. Paxson, and W. Stevens, "TCP Congestion Control," RFC2581, (1999).

[Allman99B]
Allman, M. and A. Falk, "On the Effective Evaluation of TCP," Computer Communication Review, 29, 5, (October, 1999).

[Cottrell03]
Bullot, H., R. L. Cottrell, and R. Hughes-Jones, "Evaluation of Advanced TCP Stacks on Fast Long Distance Production Networks," Journal of Grid Computing, 1, 4, (December, 2003), pp. 345-359.

[Falk03]
Falk, A., T. Faber, and J. Bannister, "Transport protocols for high performance: Whither TCP?" Communications of the ACM, 46, 11, (November, 2003), pp. 42-49.

[Falk05B]
Falk, A. and N. Jasapara, "Can a Satellite be an Internet Router?" IEEE Globecom 2005, Workshop on Advances in Satellite Communications: New Services and Systems (November, 2005).

[Floyd99]
Floyd, S. and T. Henderson, "The NewReno Modification to TCP Fast Recovery," RFC2582, (1999).

[Floyd02]
Floyd, S., "HighSpeed TCP for large congestion windows," IETF RFC 3649, Experimental (2002).

[Floyd03]
Floyd, S. and E. Kohler, "Internet Research Needs Better Models," Computer Communication Review, 33, 1 (January, 2003), pp. 29-34.

[Floyd05A]
Floyd, S., A. Medina, and M. Allman, "Measuring the Evolution of Transport Protocols in the Internet," Computer Communications Review, 35, 2 (April, 2005), pp. 37-52.

[Jacobson88]
Jacobson, V., "Congestion avoidance and control," Computer Communication Review, 18, 4 (August, 1988), pp. 314-329.

[Katabi02]
Katabi, D., M. Handley, and C. Rohrs, "Congestion Control for high bandwidth-delay product networks," Computer Communication Review, 32, 4 (August, 2002), pp. 89-102.

[Kelly03]
Kelly, T., "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," Computer Communication Review 32, 2 (April, 2003), pp. 83-91.

[Leith04]
Leith, D. and R. Shorten, "H-TCP: TCP for High speed and Long-Distance Networks," Proceedings of PFLDNet 2004, 1, 1, (February 2004).

[Leith05]
Leith, D., Y. Li, and B. Even. "Evaluating the Performance of TCP Stacks for High-Speed Networks," Proceedings of PFLDNet 2006, 1, 1, (February, 2006).

[Low04]
Low, S., C. Jin, and D. Wei, D., "FAST TCP: motivation, architecture, algorithms, performance," Proceedings of IEEE Infocom 2004, 1, 1 (March, 2004).

[Medina05]
Medina, A., M. Allman, and S. Floyd, "Measuring the Evolution of Transport Protocols in the Internet," Computer Communication Review, (April, 2005).

[Postel81]
Postel, J., "Transmission Control Protocol," RFC793, (1981).

[Rhee04]
Rhee, I., L. Xu, and L. Harfoush, "Binary increase congestion control for fast long-distance networks," Proceedings of IEEE Infocom 2004, 1, 1 (March, 2004).

[Rhee06]
Rhee, I., S. Ha, Y. Kim, L. Le, and L. Xu, "A Step toward Realistic Performance Evaluation of High-Speed TCP Variants," Proceedings of PFLDNet 2006, 1, 1, (February, 2006).

[Stoica05]
Stoica, I, Y. Xia, L. Subramanian, and S. Kalyanaraman, "One More Bit Is Enough," Computer Communication Review, 35, 4 (October, 2005), pp. 37-48.

[Zhang05]
Zhang, Y. and T. Henderson, "An Implementation and Experimental Study of the eXplicit Control Protocol (XCP)," Proceedings of IEEE Infocom 2005, 1, 1 (March, 2005).

Electronic Sources:

[Falk05A]
Falk, A., D. Katabi, and Y. Pryadkin, "Specification for the Explicit Control Protocol (XCP)," (October, 2005), http://www.isi.edu/isi-xcp/docs/draft-falk-xcp-spec-01.html.

[Floyd05B]
Floyd, S., "Metrics for the Evaluation of Congestion Control Mechanisms," (October, 2005), http://www.ietf.org/internet-drafts/draft-irtf-tmrg-metrics-01.txt.

[Netem06]
Network Emulator, netem, http://linux-net.osdl.org/index.php/Netem, last accessed April 3, 2006.

[Tierney06]
Tierney, B. "TCP Tuning Guide," http://www-didc.lbl.gov/TCP-tuning/, last revision February 13, 2006, last accessed April 3, 2006.

[Tcptrace06]
Tcptrace, http://www.tcptrace.org, last revision 2003, last accessed April 3, 2006.

[Wei05]
Wei, D., P. Cao, and S. Low, "Time for a TCP Benchmark Suite?" http://www.cs.caltech.edu/~weixl/ research/technicals/benchmark/summary.ps.

[Web06]
Web100, http://www.web100.org, last revision 2006, last accessed April 3, 2006.

VITA

Bridget Hillyer completed her Bachelor's level course work in Computer Science at Florida State University. She also has a Bachelor of Arts degree in Philosophy and Religion from Florida State University. She expects to receive a Master of Science degree in Computer and Information Sciences from the University of North Florida in December 2006. Dr. Sanjay Ahuja is serving as her thesis adviser. Bridget was previously an Application Programmer at ModusNovo, a software startup based in Tel Aviv, Israel.

Bridget is a C/C++ and Java programmer and is interested in distributed systems, networking, network intrusion detection, and software development methodologies. She has extensive experience developing software for both Windows and Linux. Bridget is married to Keith Hayes, and they have one daughter, age four years.