UCRL-TR-208615

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Analysis of Next Generation TCP

K. Halliday, A. Hurst, J. Nelson

December 15, 2004

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Analysis of Next Generation TCP

Kyle Halliday <halliday1@llnl.gov>
Andrew Hurst <abhurst@ucdavis.edu>
Jarom Nelson <nelson99@llnl.gov>

December 13, 2004

Abstract

The Transmission Control Protocol (TCP) has been around for around 30 years, and in that time computer networks have increased in speed and reliability many times over. TCP has done very well to maintain stability and avoid collapse from congestion in the Internet with this incredible increase in speed. But as the speed of networks continues to increase, some assumptions about the underlying network that influenced the design of TCP may no longer hold valid. Additionally, modern networks often span many different types of links. For example, one end-to-end transmission may traverse both an optical link (high-bandwidth, low-loss) and a wireless network (low-bandwidth, high loss). TCP does not perform well in these situations. This survey will examine some of the reasons for this, focusing on high-bandwidth networks, and offer some solutions that have been proposed to fix these problems. This paper assumes basic knowledge of the TCP protocol.

1 Overview

Originally the Transmission Control Protocol was designed to be a reliable, extensible means to transfer data over the ARPAnet, while still leaving room for growth and unseen uses in the future [10]. Considering that it was originally designed 30 years ago and is still in high use, it has done quite a good job of fulfilling its design goals.

Lately, however, cracks have begun appearing in the design of TCP when used over very high speed networks. Some of the latest research by labs and universities produces huge datasets that need to be shared with researchers across the world. For example, the Stanford Linear Accelerator Center has over a petabyte (10^{15} bytes) of stored data [2]. To share all of the SLAC data with other research institutions over an OC12 connection (622 Mbps, relatively high speed by today's standards), would require over 140 days. Increasing the speed to an OC48 (2.4 Gbps) or an OC192 (10 Gbps) would shorten the transfer time to approximately 34 days and 8 days, respectively. Considering how fast many of these research projects generate data, transfer rates that slow will not suffice.

Because of this excess of data to transfer, much research has gone into increasing the speed of the lines that transmit this data. All of the line speed in the world is useless, however, unless it is adequately utilized. As the link capacity increases due to improving technology, Standard TCP is able to use less and less of the available bandwidth (see Figure 1). One of the main causes for this decrease in utilization is the "jagged" way the congestion window

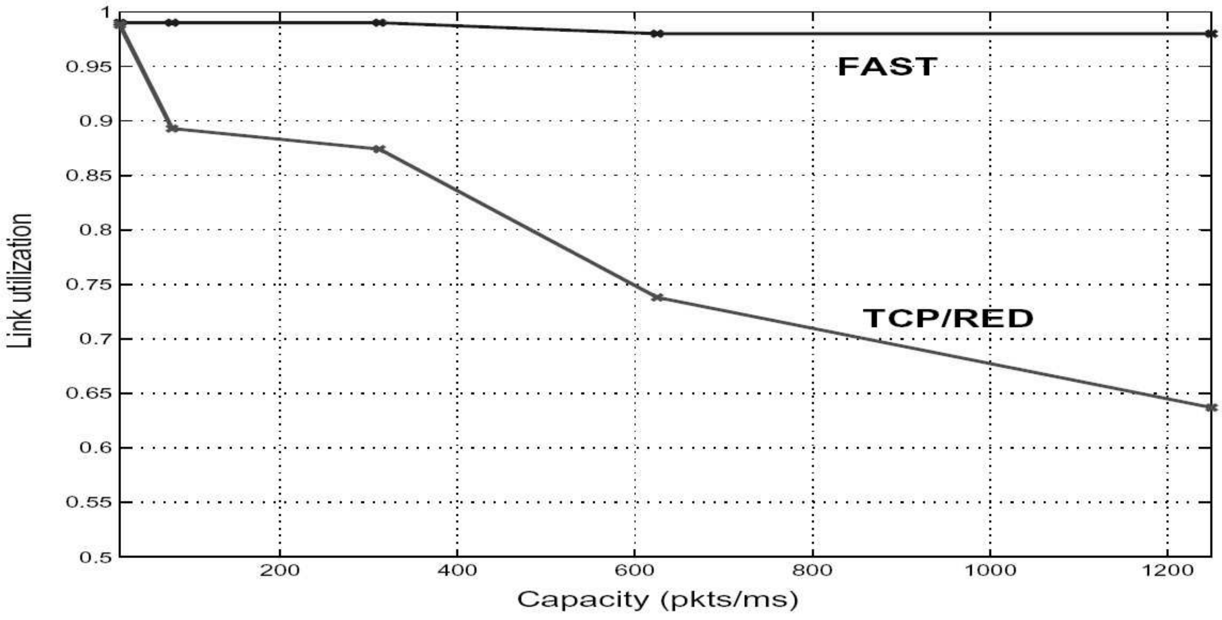

Figure 1: FAST versus TCP/RED performance as network speed increases [2]

changes due to additive-increase and multiplicative decrease in Standard TCP. Multiplicative decrease is much too drastic, while additive increase doesn't increase fast enough. See Section 2.2 for further discussion.

Another problem with the current version of TCP is that it is biased against connections with large round-trip times (RTT). Since the increase of TCP is tied to each RTT, a connection with a smaller RTT will increase its congestion window much faster than a connection with a large RTT, thus responding more quickly to changes in available bandwidth or congestion. In most cases, this will cause the faster of the two competing streams to get the majority of available bandwidth. This can be a big problem when you communicate with many different end-hosts with differing RTTs to each.

Additionally, if there is a large amount of data to send, TCP makes no effort to pace itself and sends it all at once. This can aggravate the congestion problems in the network. Many times this happens when a packet acknowledgment (ACK) is lost, delayed, or corrupted and thus causes a loss event at the sending host. Following that a large number of packets can get resent all at once from the sender (possibly the entire congestion window worth of packets). This can cause congestion because the buffers in the network routers are of finite sizes. To illustrate that congestion windows are big enough to overflow router buffers, consider the example in Section 2.2 where the window size is almost 120 MB at steady state!

Lastly, Standard TCP uses a loss event as a signal of congestion, at which point the sender lowers the congestion window to limit the number of packets to send, thus lessening the congestion in the router's buffers. [8] indicates that a loss is a poor signal of congestion because the loss may not be caused by congestion. If congestion does exist a loss should be used as a "signal of last resort", as it doesn't give an accurate view of the level of congestion. By the time a loss occurs the congestion control has already failed by overloading the capacity of the network, thus causing the loss event. Congestion control should work to avoid congestion, not merely react to it. Of course this assumes that the loss event was not caused by an unreliable network, but by the actions of the protocol.

2 TCP Breakdown

2.1 Connection Setup

In relative terms, connection setup takes very little time compared to the rest of the transfer. For example, connection setup is three packets out of the over 730 million packets sent *just from the sender* when transferring a terabyte of data[1]. Thus its effects will be ignored for the purposes of this paper.

2.2 Reliable Transfer and Congestion Control

TCP is designed to provide reliable in-order transfer of data across an unreliable network. Reliable transfer is related to congestion control because the Standard TCP congestion control algorithm is triggered by a loss event. When a receiving host detects a gap in the received packets' sequence numbers, it re-sends the acknowledgment (ACK) for the last packet received correctly and in order. In TCP Reno, the sending host will re-send the missing data when it receives three duplicate ACK's for a packet, as well as all packets after the missing data. A packet is also considered lost if a timer on the sender times out before the expected acknowledgment for a packet arrives.

When a packet is considered lost, TCP initiates congestion control. For TCP Reno this means cutting its congestion window in half (multiplicative-decrease). To recover from the loss, it increases the congestion window size by one segment every RTT (additive-increase) [10]. In the case of a 10 Gbps connection at full utilization, this means dropping down to 5 Gbps and slowly increasing back up to 10 Gbps, and as mentioned in [12] (and in more detail below) this could take up to 1.5 hours for some connections.

Congestion control after a timeout event is even more drastic, reducing the congestion window to 1 segment, and increasing from there. There is a slow-start phase, however, providing for exponential growth to a threshold level (typically half of the window size before the timeout). Beyond the threshold the additive increase starts again.

Additive-increase multiplicative-decrease (AIMD) is a major source of the problems with conventional TCP in very high speed networks. For example, take the oft-given example of a Standard TCP connection with 1500 byte packets, 100ms round-trip time, and a 10 Gbps connection. To fully utilize the connection at 10 Gbps in steady-state requires a congestion window of 83,333 segments. With the 100ms RTT, to reach full utilization after multiplicative-decrease requires roughly 1.5 hours. To maintain full utilization of the line requires no more than one packet loss per 5 billion packets, far beyond the constraints of today's physical networking technology. [12]

2.3 Flow Control and Sender / Receiver Buffer Size

Each end-host on a TCP connection has a limited amount of buffer space to allocate to TCP. And regardless of how efficient the TCP implementation is, an undersized buffer at either the

[1]Assuming a 1500 byte packet size, and not counting retransmissions

	Router feedback	Delay-based	Loss-based	Receiver modification
TCP Reno	no	no	yes	n/a
HSTCP	no	no	yes	no
STCP	no	no	yes	no
TCPW	no	no	yes	no
FAST TCP	no	yes	no	no
XCP	yes	yes	no	yes

Table 1: Several proposals to improve network performance in high speed networks, and how they compare to TCP Reno.

sender or receiver can restrict performance over any type of network. To ensure that the full bandwidth of a link is used, there needs to be enough buffer space to receive and store the data that has been received by the client but not yet processed by the application. However, an unlimited buffer could quickly consume system resources if the application is not able to process data from the buffer fast enough.

A technique called dynamic right-sizing, which automatically adjusts the buffers allocated to TCP by the operating system, has been developed at Los Alamos National Laboratory (LANL) [4]. Previously these adjustments were done by hand, and cooperation between system administrators at either end of a connection was required to tune the buffers for a particular link. Optimizing the buffers for a connection between LANL and the National Center for Atmospheric Research, researchers were able to obtain a speedup in throughput from 1 Mbps to 15-20 Mbps.

3 Specific Protocols

The remainder of this paper will discuss new TCP-based protocols that aim to solve the problems with congestion control as outlined above. The first few protocols that we mention keep the general TCP congestion control algorithm intact, and simply modify the parameters of how much to grow or decrease the congestion window (i.e. AIMD adjustments). We then look to FAST which replaces the AIMD of Standard TCP with an equation-based approach, which aims to prevent losses before they occur by indirectly monitoring the routers' queuing buffer sizes. Lastly we will examine XCP, a redesign of TCP to take into account feedback from the routers along the route a packet takes between the sender and the client.

3.1 HighSpeed TCP

HighSpeed TCP (HSTCP) [5] proposes a modified congestion control algorithm that results in much better performance and network utilization than Standard TCP. Standard TCP congestion control imposes strict constraints on network reliability for flows with large windows. For example, full utilization of a 10 Gbps network with Standard TCP necessitates a packet loss rate of 1 loss per 5 billion packets. This unrealistic reliability requirement is due to the Standard TCP response function, $w = 1.2/\sqrt{p}$, where w is the size of the average congestion window, and p is the steady-state packet loss rate [5].

The HighSpeed TCP response function uses the same response function as Standard TCP for networks with loss rates between 1 and a certain threshold, `Low_Window`, currently 10^{-3}. For packet loss rates between `Low_Window` and 0, the HighSpeed TCP response function is given by $w = 0.12/p^{0.835}$. This allows for much larger congestion windows and fewer RTTs between losses as the packet drop rate decreases. The choice of setting the `Low_Window` threshold to 10^{-3} is based largely on intuitive estimation of traffic behavior in the commercial Internet. This variable could have a major impact on existing TCP traffic, so more research should be done to determine the consequences of changing its value.

Standard TCP suffers from a jagged steady-state because it constantly probes the network for more available bandwidth. Once a congestion event occurs, the window is halved, and the sender returns to additive increase. HighSpeed TCP mirrors this behavior, but it attempts to smooth out the rough edges. The algorithm increases the window faster and decreases it less drastically than Standard TCP, all in an attempt to gain better utilization.

The increase function (congestion avoidance phase) of Standard TCP increases the window by one segment per acknowledged window. HighSpeed TCP defines a larger increase, based on the formula $w = w + a(w)$. The value of a (size of the increase) is between 1 and 73 segments, and is calculated from a lookup table based on the current window size. Thus, if the window is already large, the increase will also be large.

The HighSpeed TCP decrease function is determined by $w = (1 - b(w))/w$. In Standard TCP, b is .5, which causes the window to be reduced by half when a congestion event occurs. The HighSpeed TCP algorithm makes the reduction in window size proportional to the current window size. The value of b is also determined by a lookup table, and it ranges from .5 to .09. Large windows, rather than being chopped in half, will be reduced by only about 10 percent.

Because one goal of HighSpeed TCP is to increase network utilization, it is also important to ensure fairness with Standard TCP, especially at relatively high packet loss rates. Fairness with Standard TCP is a lower priority on more reliable networks, since Standard TCP is unable to fully utilize bandwidth in networks with drop rates of less than 10^{-6} [5]. The HighSpeed TCP response function can be characterized as MIMD (multiplicative-increase, multiplicative-decrease), whereas Standard TCP is AIMD (additive-increase, multiplicative-decrease). Given the assumption that congestion events are not synchronized, it can be shown that MIMD will converge to fairness [6]. In summary, HighSpeed TCP will not compete fairly with Standard TCP in reliable, high-performance networks, but HighSpeed TCP will compete fairly with other HighSpeed TCP connections. It is also fair to Standard TCP in low-quality or high error-rate networks.

3.2 Scalable TCP

Scalable TCP (STCP) [9] provides another alternative to the Standard TCP response function. Unlike HighSpeed TCP, which increases or decreases the sender window in proportion to the current size of the window, the Scalable TCP algorithm uses constant values for the increase and decrease factors ($a = 0.01$ and $b = 0.875$). Specifically, the window increase function is $w = w + .01$, and the decrease function is $w = 0.875 * w$. As a result, the algorithm fixes the number of round-trip times between losses, and the sending rate doubles every 70 RTTs, regardless of the current sending rate. In that way, the protocol scales well on high bandwidth-delay networks.

3.3 TCP Westwood

The main idea behind TCP Westwood (TCPW) [11] is to estimate the amount of bandwidth available before transmitting. The two methods of calculating bandwidth are Bandwidth Share Estimates (BSE) and Rate Estimates (RE). First, TCPW tries to determine the cause of packet loss in the network (either due to link errors or congestion). Research has found that when loss is caused by link erros, BSE results in better performance, and when packet loss occurs under network congestion, RE is preferable [11].

3.4 FAST

FAST TCP takes a different approach than the previous protocols mentioned. Rather than sticking with the basic congestion control algorithm of Standard TCP, FAST replaces it with a delay-based approach. It works by estimating the queuing delay in the routers used to send its data. Depending on how much queuing delay it detects it will increase or decrease the size of the congestion window [2].

The queuing delay is estimated as the difference between the minimum RTT observed so far and the current weighted average RTT [7]. The congestion window is then modified by using another equation, shown below. This equation is run once every other RTT to resize the congestion window. w is only updated every other RTT, to allow the flow to stabilize to the latest changes.

$$w \leftarrow \min\left\{2w, (1-\gamma)w + \gamma\left(\frac{baseRTT}{RTT}w + \alpha(w,q)\right)\right\} \tag{1}$$

w is the congestion window size; γ is a constant in $(0,1]$ and is usually set to 1; $baseRTT$ is the lowest RTT seen so far, whereas RTT is the weighted average RTT calculated so far; and q is the queuing delay. Lastly, $\alpha(w,q)$ is a function on w and q used to adjust the rate of the window size change, and is generally set to a constant value.

As you can see above in equation 1, FAST requires that the congestion window no more than double every time it is re-calculated. This ensures that out-of-control growth does not occur, and thus lessens the chances of exceeding the bandwidth in a link. As well, you can see by the equation above that the growth of a window is dependent on the ratio of the baseRTT (the fastest RTT seen so far) to the current RTT. For example, if the current RTT is twice that of the baseRTT, then that fraction will evaluate to 0.5, and the window size will increase by $0.5 + \alpha$.

α can be adjusted depending on the size of q. One method put forth in [7] is to have $\alpha(w,q) = kw$ when q is zero (where k is a positive integer) and $\alpha(w,q) = c$ when q is non-zero (where c is a small constant). This allows for multiplicative growth as long as the network can sustain it, and when the routers' queues start to fill up FAST increases the window very slowly, if at all.

Currently upon a loss event FAST uses multiplicative decrease and cuts its congestion window by half. In the future the designers hope to modify the equation above to react to loss events as well.

To prevent overflowing the routers' buffers when a large burst of packets can be sent, FAST employs a burstiness control module that controls how much data can be sent at once based on the measure of the queuing delay. This helps ensure that a bad situation is not made worse by adding more packets to already full router queues.

Figure 2 shows the length of 3 flows used to test FAST, STCP, HSTCP, and TCP Reno. The results are shown in figures 3 and 4. By examining figure 3 (which shows throughput on the top graphs, and congestion window size on the bottom) you can see that FAST is much more stable than STCP, HSTCP, and TCP Reno on the same dataset. As well, figure 4 (showing average queue size, cumulative packet losses, and link utilization) displays that FAST minimizes the average queue size, produces practically no packet losses, and gives the most consistent link utilization of the 4 protocols shown.

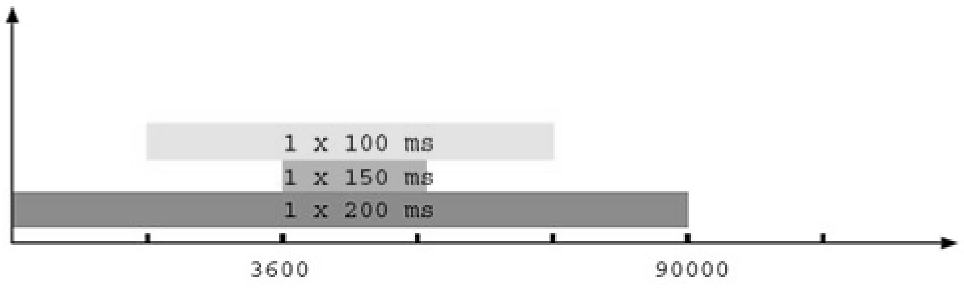

Figure 2: FAST dummynet experiment setup, 3 flows and their lengths [7]

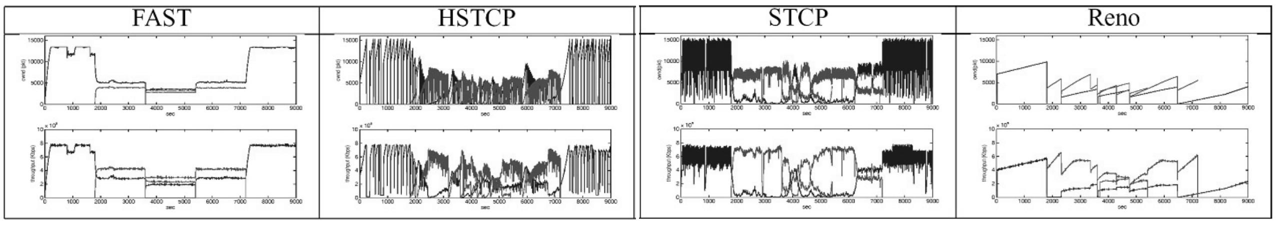

Figure 3: FAST dummynet results. Throughput (top) and congestion window size (bottom) [7]

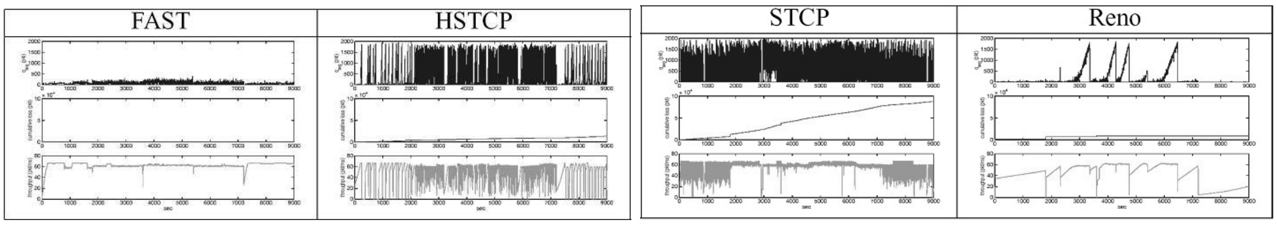

Figure 4: FAST dummynet results. Queue size (top), cumulative lost packets (middle), and link utilization (bottom) [7]

3.5 eXplicit Control Protocol (XCP)

XCP [8] takes a different approach to fixing the pitfalls in TCP. While the other proposed solutions modify how TCP reacts to loss events and controls the congestion window, XCP gets rid of TCP congestion control altogether and develops a new protocol. In XCP, congestion and fairness are controlled through input from the routers along the path. Also, the protocol decouples the management of congestion and fairness into two separate controllers in the routers. The Efficiency Controller monitors the bandwidth utilization of the link and the queue length, and computes the aggregate increase/decrease (feedback) in throughput needed to maximize efficiency. This aggregate throughput feedback is given to the Fairness Controller, which distributes it to all the packets passing through the router. The Efficiency Controller uses MIMD to converge quickly to fairness.

Feedback from the routers is given by adding a Congestion header between the Transport Layer header and the IP header [1]. The most important parts of the header are the following: `RTT` - the sender's measurement of the round trip time, `Throughput` - the sender's measured throughput, `Delta_feedback` - the sender's desired throughput increase (modified by the routers along the path), and `Reverse_feedback` - the value of `Delta_feedback`, copied into the header by the receiver (for feedback on the return path of acknowledgment packets). The fairness controllers in the routers along the path will modify the `Delta_feedback` and `Reverse_feedback` according to the values determined by the routers to maximize efficiency and fairness. To converge to fairness, the routers use AIMD. Since fairness is decoupled from efficiency, other algorithms for fairness and efficiency control are possible, which could be studied in further research.

The performance of XCP in simulation and in experimental implementation is impressive. In both high-bandwidth and high-RTT environments, XCP is able to utilize the bandwidth at above 90% (figure 5), keep queue length minimal, and have virtually no loss events. XCP also is very effective at fairly distributing bottleneck bandwidth (figure 6). These characteristics are demonstrated by simulations in [8] and implementation experiments [3]. Further research is needed to learn how XCP will perform in a large scale environment and how fairly it will compete with TCP and other network traffic flows.

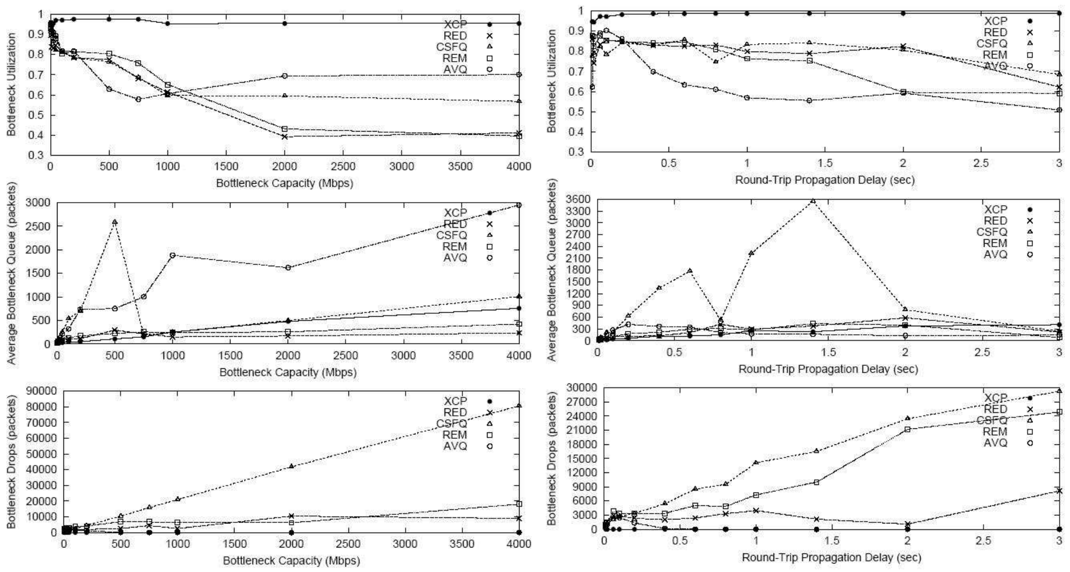

Figure 5: XCP High Bandwidth and High RTT utilization (simulation) [8]

XCP is a very effective protocol for avoiding congestion and achieving fairness across multiple connections, and has been demonstrated to be stable in simulations and experiments. The biggest problem with XCP is the challenge of implementation on a wide scale. [8] suggests two ways to approach implementation. A cloud-based approach in which XCP would be implemented in parts of a network, with border systems providing traffic to non-XCP net-

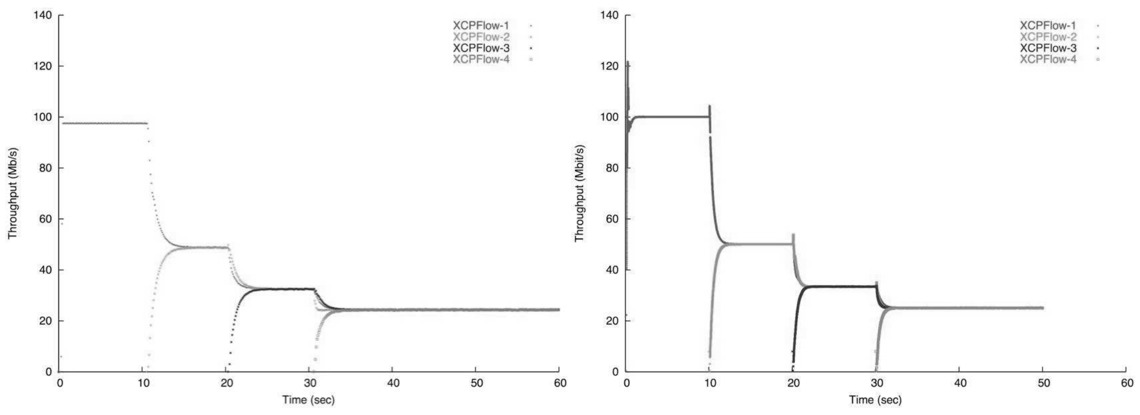

Figure 6: XCP Fairness (measured and simulated) [3]

works. Another approach suggested is to implement XCP routers with TCP support, and detect which routing algorithm to use depending on the type of traffic. There is a lot of room for research before XCP could be implemented on a wide scale, especially since it requires changes to routers. Once the routers are implemented, it would be difficult to change them if it is decided to use different algorithms within the framework of XCP. Also before XCP or another similar protocol could be implemented, there would need to be a lot of research to learn if the given parameters in the congestion header are the best options. Adding congestion control to routers is a significant task. If it is to be implemented, it is important to understand the best approach to giving and receiving feedback.

4 Outstanding Problems / Research Ideas

How do you create a congestion control algorithm that is at the same time backwards compatible, efficient in high-bandwidth networks, and fair with the other connections? Some research focuses on maximizing throughput for a single connection. Other research gives fairness with existing traffic a heavier consideration. Still others seek balance and trade offs between all gaols. Is it possible to ensure that any new protocol performs well on virtually any type of network, from dial-up Internet to high-speed bulk data transfers to wireless networks?

Furthermore there is no standard way to measure the quality of a new protocol as compared to other protocols in several network topologies. One possible area of research is to develop this measure and evaluate the Transport protocols using it. The output of this research could include a set of benchmarks to compare protocols objectively.

Lastly, there appears to be little research into the performance of Transport protocols in mixed-protocol networks.

5 Conclusion

TCP congestion control is a complex system that has many possible methods of modification to improve its performance on very high-bandwidth networks. Methods designed to improve TCP must strike a balance between network congestion control, client and server buffer sizes, fairness, and backwards compatibility. Due to the huge number of computers and routers connected to the Internet, any non-backwards compatible TCP replacement would take a very long time, a huge amount of effort, and lots of money to implement.

As well, any backwards-compatible TCP improvement must "play fair" with the current versions of TCP in use now, or at least not take any more bandwidth from them than they would otherwise use. If a modification of this entrenched protocol can maximize utilization, fairness, and backwards compatibility, it will be well on its way to becoming the next generation TCP.

References

[1] D. Katabi A. Falk. Specification for the explicit control protocol (xcp), 2004.

[2] C. Jin et. al. FAST TCP: From theory to experiments, 2003.

[3] Aaron Falk, Ted Faber, Eric Coe, Aman Kapoor, and Bob Braden. Experimental measurements of the explicit control protocol (xcp), 2004.

[4] Mike Fisk and Wu chun Feng. Dynamic right-sizing in TCP, Oct 2001.

[5] S. Floyd. (rfc 3649) HighSpeed TCP for large congestion windows, December 2003.

[6] S. Gorinsky and H. Vin. Extended analysis of binary adjustment algorithms, 2002.

[7] Cheng Jin, David X. Wei, and Steven H. Low. FAST TCP: Motivation, architecture, algorithms, performance. In *IEEE INFOCOM*, Hong Kong, China, March 2004.

[8] D. Katabi, M. Handley, and C. Rohrs. Internet congestion control for future high bandwidth-delay product environments, 2002.

[9] T. Kelly. Scalable tcp: Improving performance in highspeed wide area networks, 2003.

[10] James F. Kurose and Keith Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* Addison-Wesley Longman Publishing Co., Inc., 2005.

[11] M.Y. Sanadidi Bryan Kwok Fai Ng Ren Wang, Massimo Valla and Mario Gerla. Efficiency/friendliness tradeoffs in tcp westwood. In *In Proceedings of the Seventh IEEE Symposium on Computers and Communications, Taormina, July 2002.*

[12] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control for fast, long distance networks. In *IEEE-Infocom 2004*, March 2004.