Louisiana State University
LSU Digital Commons

LSU Master's Theses Graduate School

2006

Evaluation of TCP Based Congestion Control Algorithms Over High-Speed Networks

Yaaser Ahmed Mohammed
Louisiana State University and Agricultural and Mechanical College

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_theses

Part of the Computer Sciences Commons

Recommended Citation
Mohammed, Yaaser Ahmed, "Evaluation of TCP Based Congestion Control Algorithms Over High-Speed Networks" (2006). *LSU Master's Theses*. 2267.
https://digitalcommons.lsu.edu/gradschool_theses/2267

This Thesis is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Master's Theses by an authorized graduate school editor of LSU Digital Commons. For more information, please contact gradetd@lsu.edu.

EVALUATION OF TCP BASED CONGESTION CONTROL ALGORITHMS OVER HIGH-SPEED NETWORKS

A Thesis
Submitted to the Graduate Faculty of the
Louisiana State University and
Agriculture and Mechanical College
in partial fulfillment of the
requirements of the degree of
Master of Science in Systems Science

in

The Department of Computer Science

by
Yaaser Ahmed Mohammed
B.E., Osmania University, 2004
December 2006

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Seung-Jong Park for showing a lot of patience in answering all my questions throughout my thesis work. I also like to thank him for putting enough trust on me and giving me this opportunity to work for him. I would also like to acknowledge him for his invaluable feedback without which this thesis would never be possible.

I would also like to thank my fellow student Suman Kumar for constantly inspiring and motivating me. I would also extend my thanks to one my other colleagues Michael (Yixin Wu) for all the help in the kernel compiling. My thesis wouldn't have been completed without their help. I would also like to thank CCT (Center of Computation and Technology), LSU for letting me utilize their resources for completing this work. Finally, I also like thank all my friends and family for their moral support.

This research work has been supported by the NSF-CISE for the proposal **"EnLIGHTened Computing: Highly-dynamic Grid E-science Applications Driving Adaptive Optical Control Plane and Compute Resources"**

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS ---------- ii

ABSTRACT ---------- iv

1. INTRODUCTION ---------- 1

2. BACKGROUND ---------- 3
2.1 A Brief Overview of TCP ---------- 3
2.2 High-Speed Networks and Bandwidth-Delay Product ---------- 4
2.3 Problems of TCP over High-Speed Networks ---------- 4
2.4 Proposed Solutions to Overcome Limitations of TCP ---------- 6
2.5 TCP Variants ---------- 8
2.5.1 HighSpeed TCP ---------- 8
2.5.2 Scalable TCP ---------- 9
2.5.3 Binary Increase Congestion TCP ---------- 10
2.5.4 CUBIC ---------- 11
2.5.5 Hamilton TCP ---------- 11

3. EXPERIMENTAL SETUP ---------- 12
3.1 Hardware and Software Configuration ---------- 12
3.2 Emulators and Tools: Overview and Selection ---------- 13
3.3 Linux Tuning for 10G Networks ---------- 16
3.3.1 Setting BDP ---------- 16
3.3.2 Setting other Kernel Parameters ---------- 17
3.3.3 Using Jumbo Frames ---------- 19
3.4 Applications of High Speed Networks ---------- 21

4. PERFORMANCE EVALUATION ---------- 23
4.1 Experimental Evaluation of TCP in High-Speed Scenario ---------- 23
4.1.1 Effect of Using Large Frames ---------- 23
4.1.2 Effect of Using Large BDP ---------- 24
4.1.3 Effect of Using the Window Scaling Option ---------- 26
4.1.4 Effect of Using Parallel TCP flows ---------- 27
4.1.5 Effect of Dynamically Increasing and Decreasing RTT on TCP ---------- 29
4.2 Performance Comparison of TCP Variants ---------- 29
4.2.1 Performance in no Loss Scenario ---------- 30
4.2.2 Protocol Performance over Packet Loss ---------- 31
4.2.3 Protocol Performance over 10^{-5} Packet Loss ---------- 32

5. CONCLUSION ---------- 36

REFERENCES ---------- 38

VITA ---------- 40

ABSTRACT

The Additive Increase Multiplicative Decrease (AIMD) algorithm of the Transport Control Protocol (TCP) had worked remarkably well over low speed networks and had guaranteed fairness to the users all over these years, but at present, the demands for transferring large quantities of data over very high-speed networks are increasing at a tremendous rate. Because of its AIMD algorithm to control its window growth function accompanied by a slow response function which is inadequate over high-speed links, TCP has been proven to underutilize the available network bandwidth and leave a considerable amount of unused bandwidth.

To overcome this limitation of TCP, the network research community came up with a number of TCP variants: HSTCP, STCP, BIC TCP, CUBIC, HTCP, and FAST TCP. All these protocols differ in the window growth policy to utilize the available bandwidth over a high-speed link. Various tests have shown that these protocols successfully utilize the link but at the same time they are not able to guarantee fairness to the other flows in the network. In this work, we aim to explore the following research questions:

- Explore how tuning affects the performance of TCP and over 10G networks.
- Compare TCP variants over a high-loss back-to-back environment

In future, this work can be further extended in exploring the following two questions

- Explore Performance Metrics for fair comparison of protocols over 10G back-to-back links

Move towards designing a congestion control protocol for back-to-back high-speed (Gigabit) links

1. INTRODUCTION

The Transport Control Protocol (TCP) was initially designed in the 1960-70s to operate over links of different speeds. The reason for TCP to become popular was the reliability that it provided and also the fair performance of the Additive Increase Multiplicative Decrease (AIMD) algorithm that it uses to control its window growth policy. The AIMD algorithm had worked remarkably well over low speed networks and at the same time guaranteed fairness to the users all over these years, but at present, the demands for fast transfer of large volumes of data and the deployment of the network infrastructures to support this demand is increasing at a huge rate. As Computer networks are growing exponentially in terms of size as well as bandwidth, TCP underutilizes the network bandwidth as shown in [1,2]. The AIMD window growth policy coupled with the slow response of TCP in fast long distance networks leaves a considerable amount of underutilized bandwidth in such networks.

To overcome the limitation of TCP over high-speed networks, a number of TCP variants were proposed in recent years. These protocols namely, HighSpeed TCP [2], Scalable TCP [3], BIC-TCP [4], CUBIC [5], Fast TCP [6], Hamilton TCP [7] etc. aim to serve this huge ongoing requirement of scalability and bandwidth requirement. Most of these protocols are designed with an aim to replace the standard TCP over the Internet in the coming years. These protocols differ with TCP in terms of their window growth policy and have different characteristics.

These TCP variants can be tested and deployed over a number of high-speed networks which have been developed and deployed over major research institutions. These networks, such as Louisiana Optical Network Initiative (LONI [8]), National

Lambda Rail (NLR [9]), NASA Research and Engineering Network (NREN [10]), Energy Sciences Network (ESnet [11]), Abilene [12], GEANT [13] etc., have large amount of bandwidth which ranges anywhere from 100Mbps to 100Gbps. For example, NLR, a major initiative of the U.S. Research Universities and private sector technology companies, in the coming future may require a channel capacity of 40Gbps to 100 Gbps.

TCP fails to utilize the available bandwidth on these networks and leaves a lot to be desired in terms of its performance.

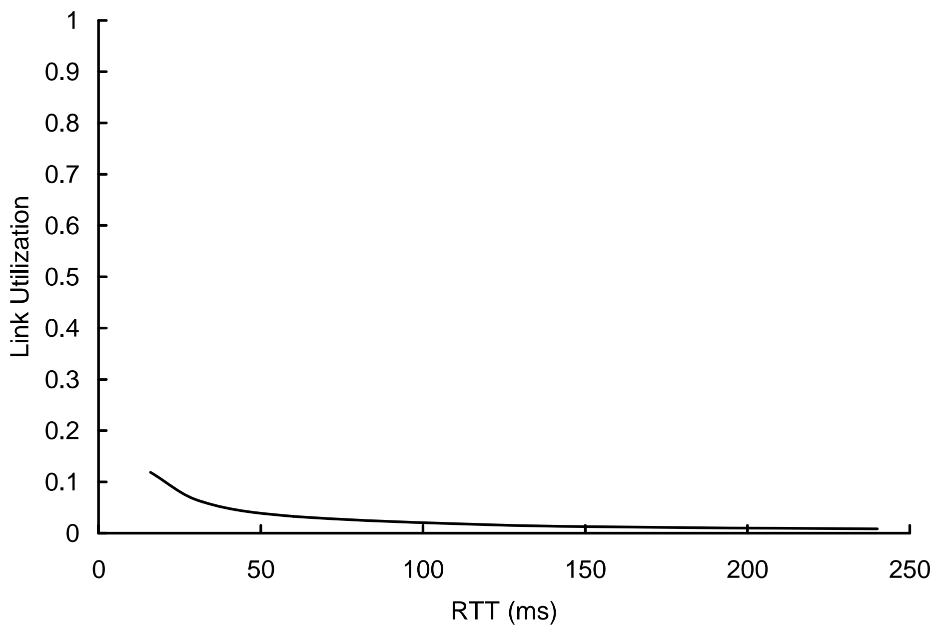

Figure 1.1 Average Throughput of a TCP Flow for 120ms RTT

Figure 1.1 shows the link utilization of TCP when the delay is varied from 16 to 240ms. Default values have been used for the buffer and packet size. As illustrated in the figure, TCP fails to utilize the network over high-speed and long delays and leaves a lot of bandwidth unused.

2. BACKGROUND

2.1 A Brief Overview of TCP

TCP resides in the transport layer of the protocol stack and provides reliable data-transfer between processes which use the protocol on the end host-systems. The TCP uses the AIMD (Additive Increase Multiplicative Decrease) algorithm to control its congestion window. This congestion window (cwnd) is the amount of unacknowledged data in the link between the sender and the receiver. This value is an estimate and has to be adjusted depending on the feedback that the TCP sender gets from the network. The initial value of the TCP congestion window changes either according to the slow start algorithm or according to the congestion avoidance. In slow start, the TCP congestion window increases by one on the receipt of every acknowledgement and in the congestion avoidance phase, the congestion window value increases by one segment every RTT. TCP uses another parameter called ssthresh which determines whether TCP goes into the slow start or congestion avoidance phase, i.e., TCP adjusts its congestion window based on the slow start algorithm and goes into the congestion avoidance phase once it reaches the value of ssthresh.

When the TCP receiver receives an out of order packet, i.e. if a packet is lost, it generates duplicate acknowledgements (dupack) for the lost packet. If the TCP sender receives three duplicate acknowledgements then the sender retransmits the segment and changes its congestion window depending on the value of the current ssthresh.

The TCP's AIMD algorithm works as follows: (i) During the congestion avoidance phase, after every acknowledgement is received, the new congestion window (cwnd) = cwnd + (1/cwnd), i.e., it increases by one for every RTT.

(ii) In case of a loss event, the new cwnd = cwnd/2

Over high-speed long delay networks, this additive increase (AI) nature of TCP is too slow to utilize the entire bandwidth and the multiplicative decrease (MD) nature of TCP is considered to be too drastic.

2.2 High-Speed Networks and Bandwidth-Delay Product

A network can be classified as a high-speed network based on two factors: large delays and huge bandwidths. The high-speed networks are said to have a large Bandwidth-Delay-Product (BDP). This bandwidth delay product determines the amount of data which can be transferred in the network. As the name suggests, the value of the bandwidth delay product equals to the product of the available bandwidth and the latency of the link between the end-host systems. This BDP is also one of the most important factors which should be calculated to tune TCP depending on the type of the network.

2.3 Problems of TCP over High-Speed Networks

1) AIMD Algorithm: The AIMD algorithm, discussed above, takes a very long time to discover the available bandwidth on high BDP link. In TCP, the linear increase by one packet per RTT is too slow and it does not help TCP fully utilize the available bandwidth. Also the multiplicative decrease per loss event i.e., decreasing the congestion window by half, can be considered to be too drastic.

Because of the AIMD algorithm that TCP follows, it may waste many RTTs ramping up to full utilization following a burst of congestion. For example, as shown in [1], a TCP connection with an 1000Byte packet and an 100ms round trip time, filling a 10 Gbits/sec pipe would require a congestion window of W=1,25,000 packets and a packet drop rate of at most one drop every $N=10^{10}$ packets. This means that there is one

drop for every S=8000 seconds which means that in the transfer time is very large, TCP may waste many RTTs ramping up to full utilization following a burst of congestion.

Figure 2.1 shows the average throughput of TCP over a 10Gbps network. The RTT for the experiment is varied from 16 to 240ms. As depicted in the figure, the overhead caused due to the AIMD nature of TCP over large delays does not help it utilize the entire available bandwidth.

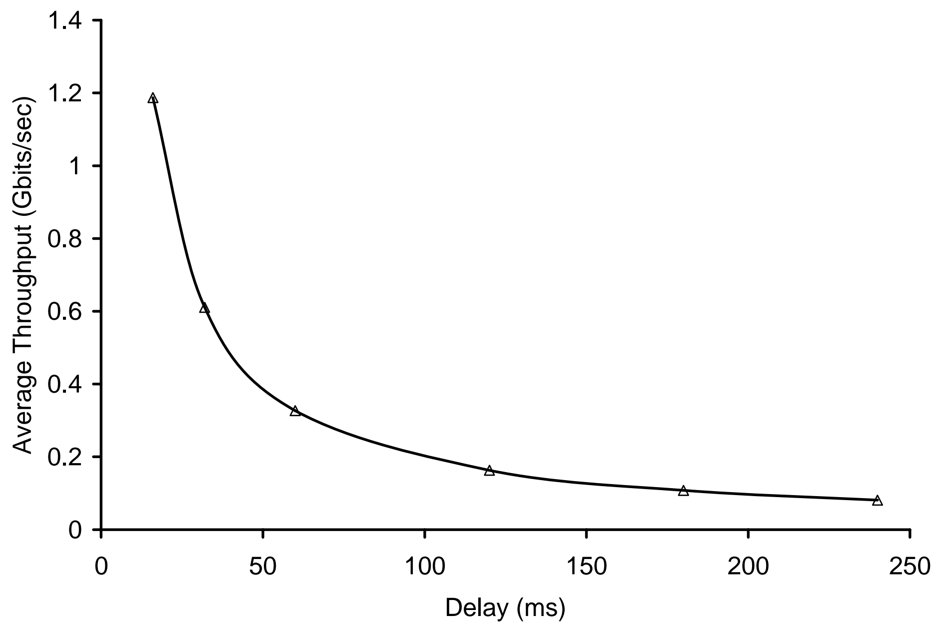

Figure 2.1 Average Throughput of TCP over Varying Delay

2) Unfairness problems with long RTT: Over long delays, as the time required to send the data becomes very small compared to the time the data needs to reach the destination, the overhead of TCP increases to a very large extent. Also, in case of multipoint-to-point communication patterns where multiple flows with different RTTs can compete for the same bottleneck bandwidth, the window of a shorter RTT flow will grow faster than that of a longer RTT flow. This means that the shorter RTT flow will consume a large chunk of bandwidth compared to the shorter RTT flow thus leading to unfairness.

Another problem in this scenario is that as TCP sends data packets after receiving an acknowledgement of the last data packet sent, older information about the link conditions is received at the source and thus the congestion control decision based on such past information may not lead to a proper action.

3) Response Function: As discussed in [2], the steady-state response function of TCP is given by $w = 1.2/sqrt(p)$

where P is the per-packet drop rate, for a TCP sending an acknowledgment packet for every data packet. Based on the response function of TCP, TCP does not have a very dynamic congestion control algorithm.

4) Recovery from Consecutive Time-Outs: From the AIMD algorithm of TCP, if a time out occurs at congestion window W, the TCP sender decreases the window to W/2 and takes considerable time to go back to that state. However, in case of a second timeout at this moment i.e. the retransmitted packet fails to reach the receiver, TCP increases its congestion window by one every time instead of going back to slow start.

2.4 Proposed Solutions to Overcome Limitations of TCP

1) Tuning TCP: The TCP window size can be adjusted to be the product of the bandwidth and the RTT delay of the network. This can be done by tuning the network which is discussed in the later sections.

2) *TCP Striping:* Here multiple paths between the sender and the receiver host systems can be used by the same connection simultaneously. This approach allows connections to enjoy the aggregate bandwidth offered by the multiple paths irrespective of the individual characteristics of the path. PSockets[14] and GridFTP[15] use the same approach over the data middleware or the application layer e.g. etc. As we discuss in our performance

metrics, this approach may require a lot of tuning at the sender side as we may have two paths with very different RTTs from the sender to the receiver.

3) Enhancing TCP: Improving the design of the existing AIMD algorithm of TCP e.g. HSTCP[2], STCP[3], BICTCP[4], HTCP[7] etc. Some of these protocols use the traditional TCP algorithm when the congestion window is small and change the AIMD algorithm only when the congestion window becomes large.

An advantage of using this approach is that the cost of implementing these variants of TCP will be very less when compared to implementing a completely new protocol like XCP[16] which requires additional support from intermediate routers.

4) *Using feedback approach:* Some protocols like FAST TCP and XCP use feedback approach to achieve high per-connection throughput.

5) *Using Parallel TCP:* Instead of using multiple flows between the sender and the receiver over multiple paths as in network striping, multiple flows are sent between the sender and the receiver on the same path [17, 18]. As discussed in the coming sections, this approach also requires a lot of tuning. Increasing the flows to a huge value can result in huge packet losses and overhead both at the sender and the receiver side.

6) *Reducing Overhead*: Making TCP more UDP like to remove the overhead on TCP that is caused by acknowledgements.

Figure 2.2 compares the performance of TCP and UDP for a 120ms delay over 10Gbits/sec. As illustrated in the figure, TCP guarantees reliability on the cost of additional message overheads. Over short RTT, TCP gives a throughput that is comparable to that of UDP but as the delay increases, TCP offers reliability on the cost of

additional message overheads. This fairness property of TCP makes it inefficient for high-speed long delay networks.

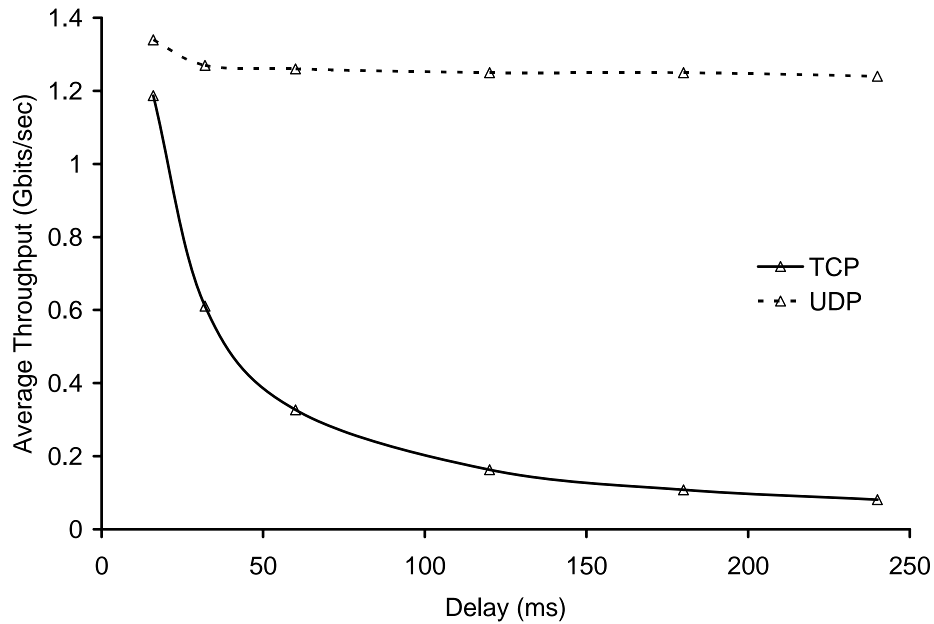

Figure 2.2 Performance of TCP Vs UDP over back-to-back links.

2.5 TCP Variants

2.5.1 HighSpeed TCP (HSTCP)

HighSpeed TCP [2] was developed by Sally Floyd. It is an enhancement to TCP's congestion avoidance algorithm that adjusts the AI and MD parameters of TCP. HSTCP uses a table of values based on the number of acknowledgments and the number of packets currently in the link to predict the new values of AI and MD. The values for AI range from 1 (standard TCP) to a high of 73 packets, and the range of MD is from 0.5 (standard TCP) to a low of 0.09. Consequently, when a congestion event occurs over large BDP networks, TCP does not drop back as much and adds more than one packet per RTT, thus recovering faster.

HSTCP modifies the congestion window as follows:

In case of an Ack: cwnd =cwnd + a(cwnd)/cwnd

Loss: cwnd = (1-b(cwnd))*cwnd)

In the first case, a higher value of cwnd gives a higher value of a(cwnd) and in the second case a higher value of cwnd gives a lower value of b(cwnd). The values of ‘a’ and ‘b’ depends on the current value of the congestion window.

2.5.2 Scalable TCP

Scalable TCP [3] was developed by researchers at Cambridge University. It modifies the AIMD algorithm of TCP such that the increase after the drop is exponential instead of linear as in TCP.

Scalable TCP takes the approach that the multiplicative decrease factor should always be .125 (reduce window by 1/8 on congestion events). Also, instead of doing an additive increase, it does a multiplicative increase beyond a certain threshold. The multiplicative increase is of 5% of the current congestion window. This MIMD nature of STCP has the effect of forcing TCP to always recover in a small number of RTTs.

Another noticeable factor of STCP is that the recovery time after a congestion event is independent of the current window size.

Scalable-TCP modifies the TCP cwnd as follows:

cwnd = cwnd + a, in case of an ACK

cwnd = b * cwnd, in case of a Loss Event

According to them, the suggested values for the parameters “a” and “b” are 0.01 and 0.875, respectively. To make STCP suitable for low speed networks, it behaves like standard TCP when the congestion window is less than a threshold, low window, but

when the congestion window value increases from this threshold; Scalable-TCP's MIMD update rules are applied.

2.5.3 Binary Increase Congestion TCP (BIC TCP)

This protocol has been developed by a team at North Carolina State University (NCSU). From kernel 2.6.15, BIC TCP has replaced TCP Reno as the default protocol in Linux.

BIC TCP [4] uses the concept of Binary Search Increase along with Additive Increase to either increase or decrease the value of the congestion window. Their main idea is to view the congestion control as a searching problem and depending on whether or not an acknowledgement is received they keep changing the "target" window size.

In case of a packet loss, BIC TCP reduces the congestion window by a multiplicative factor (minimum) and then performs a binary search between minimum and the value of the congestion window at the time of loss. In case the congestion window grows past the current maximum, it performs 'max probing' to search for a new maximum window size.

Some of the features of BIC TCP as mentioned in their paper [4] are following:

(i) Scalability: It can scale its bandwidth share to 10Gbits/s around 3.5e-8 loss rates

(ii) RTT fairness: For large windows, its RTT unfairness is proportional to the RTT ratio of the AIMD Algorithm

(iii) TCP Friendliness: It achieves a bounded TCP fairness for all window sizes. Around high loss rates where TCP performs well, its TCP friendliness is comparable to STCP's whose friendliness is comparatively better compared to other protocols.

(iv) Fairness & Convergence: Compared to HSTCP and STCP, it achieves better bandwidth fairness over various time scales and faster convergence to a fair share.

2.5.4 CUBIC

CUBIC [5] is a modification of BIC TCP. CUBIC tries to improve the fairness of BIC TCP. BIC TCP uses the binary search increase strategy for the congestion window whereas CUBIC has a cubic window increase functions. Also CUBIC takes the time of the last congestion event into account rather than the congestion window value.

2.5.5 Hamilton TCP

This variant of TCP maintains its fairness by using RTT-scaling. The congestion window value is increased based on the time between successive congestion events and also the ratio of the minimum RTT observed to the maximum RTT observed. At a loss event, Hamilton TCP adopts an adaptive back-off strategy and reduces its congestion window based on the ratio.

3. EXPERIMENTAL SETUP

3.1 Hardware and Software Configuration

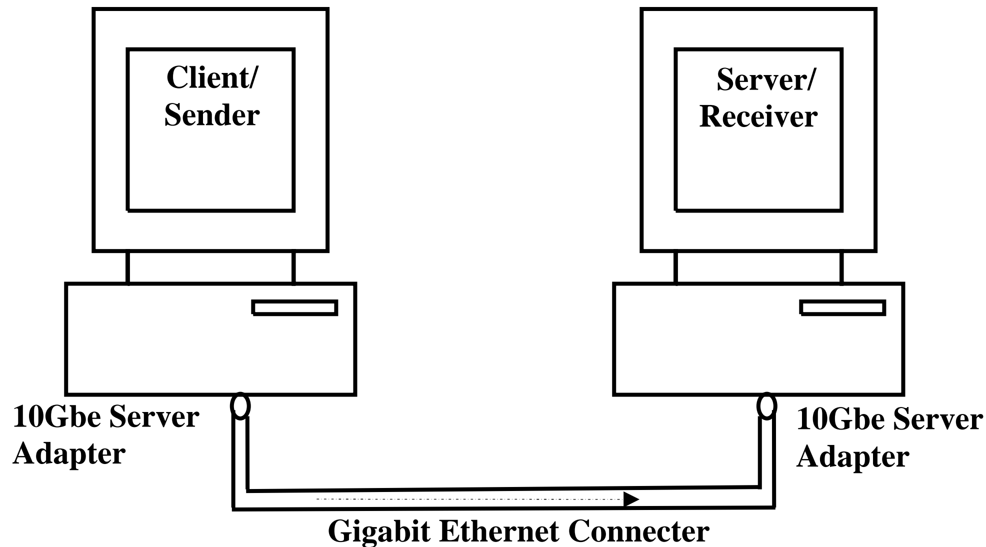

Figure 3.1 Experimental Setup

Figure 3.1 shows the experimental setup we have used for all the experiments. The two machines are connected back to back using a 10G cross over cable. This setup would be similar to two systems that are connected back to back over a Layer 2 switch. The machines are equipped with Chelsio N210 10Gbe Server Adapter [19] on the PCIX 133Mhz/64 bit slot. Both the systems were having similar hardware and software configurations. The summary of the configurations of the end systems is as follows:

Motherboard: Tyan S2895A2NRF Dual Opteron/SATA/8X/GB/1394

CPU: AMD Opteron 252 2.6GHz processor with heat sink and fan

Memory:

8x2GB PC2700 ECC Registered DDR 256 x 72 (2GB x 4)

8x512MB PC2700 ECC Registered DDR 64X72 (512MB x 4)

Hard Drive:

2x80GB Serial ATA 7200rpm 8MB Hard Drive (O/S Drive)

10x147GB SCSI U320 15K rpm 80pin 8MB Hard Drive (Configured RAID 5, 4+1)

Network Card: Chelsio N210-SR 10Gbe Server Adapter

Operating System: Linux Kernel 2.6.17

For introducing delay between the two systems we installed NIST NET [20] on the receiver side. More information about the emulator is given in the next section. The delay between the two links was adjusted to 16ms, 40ms, 80ms, 120ms, 180ms 240ms, and 320ms using NISTNet. The kernel also came with all the congestion control protocols that we discussed in Chapter 2.

3.2 Emulators and Tools: Overview and Selection

We used a network emulator at the receiver side between two machines connected back-to-back. The benefit of using emulators over simulators like ns2 [21] is that they actually make use of the physical machines and interfaces, which in turn will help us to overcome issues and problems which we can find in the real scenario. Another advantage is that we can be assured that the high-speed protocols would also behave in a similar fashion. Also, by configuring the delay and drop, we were able to observe the behavior of the CPU, hard disk, the NIC card, the performance of the PIC bus etc which is not possible using a simulator. Another problem with simulators like ns2 [21] and OPNET [22] have large overhead problems in simulating Gigabit links. Also, things like parameter tuning, deciding how many parallel flows to use etc, is a tedious task and requires a lot of tuning to reach the optimal value was made simple with the use of emulators.

The basic working of emulators involves building a queue usually between the Ethernet card and the IP layer where all these operations take place. As we set different

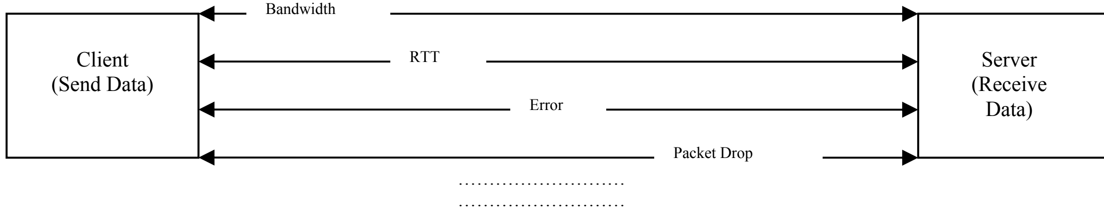

Figure 3.2 Different Functionalities of an Emulator

values of delay, bandwidth, jitter, packet loss, etc., the packets entering or leaving the emulator are processed accordingly. There are two emulators which could be used for this purpose namely NIST Net [20] and NetEM [23]. Further results showed us that NetEM showed some instability and had a large over head compared to NIST Net.

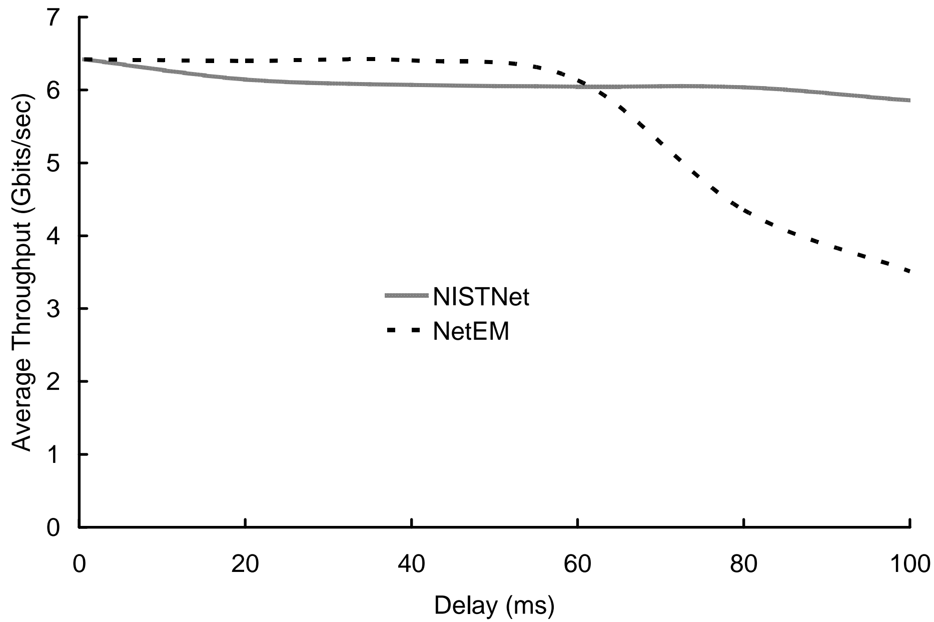

Figure 3.3 NISTNet Vs NetEM over Different Delay

Figure 3.3 above shows the performance of NIST Net and NetEM over different delay. Based on our experiments, NISTNet performed much better than NetEM over large RTT and is also much more stable than NetEM. NetEM shows a lot of variations and performs abnormally in some cases. We believe that this may happen because NetEM creates a queue at the sender side and holds the data packet in that queue. Using jumbo frames over high-speed large delay network causes the data packet to be very large and hence adds to the overhead on the end systems thereby reducing the overall throughput.

We used both Iperf [24] and NutTCP [25] for generating traffic and calculating the overall average throughput. Iperf also helped in the tuning of the TCP connections over a specific client-server path by allowing us to change the TCP window size which controls how much data can be in the network at any given time. For optimal performance, this value should be equal to the bandwidth delay product of the network as discussed.

We encountered problems using Iperf in our case. In Figure 3.4, for both the cases the BDP has been set to 500MB. As depicted in the figure, Iperf shows a less throughput than NutTCP for the same overall scenario. This is because of the intrusive nature of TCP which makes it saturate the entire path and this it increases the overall path delay and jitter. Iperf also utilizes 100% of the CPU which may be another reason for showing a lesser throughput than NutTCP which comparatively has a less overhead.

One of the limitations of both Iperf and NutTCP is that both these applications, after creating a dummy data at the sender side, discard it as soon as the receiver receives it. This may not be the case for a real application which even though may be memory-to-memory transfer yet the receiver, after receiving the data will have to use the data. Also

hard-disk to hard-disk transfers may have a very less throughput because of the hardware bottleneck.

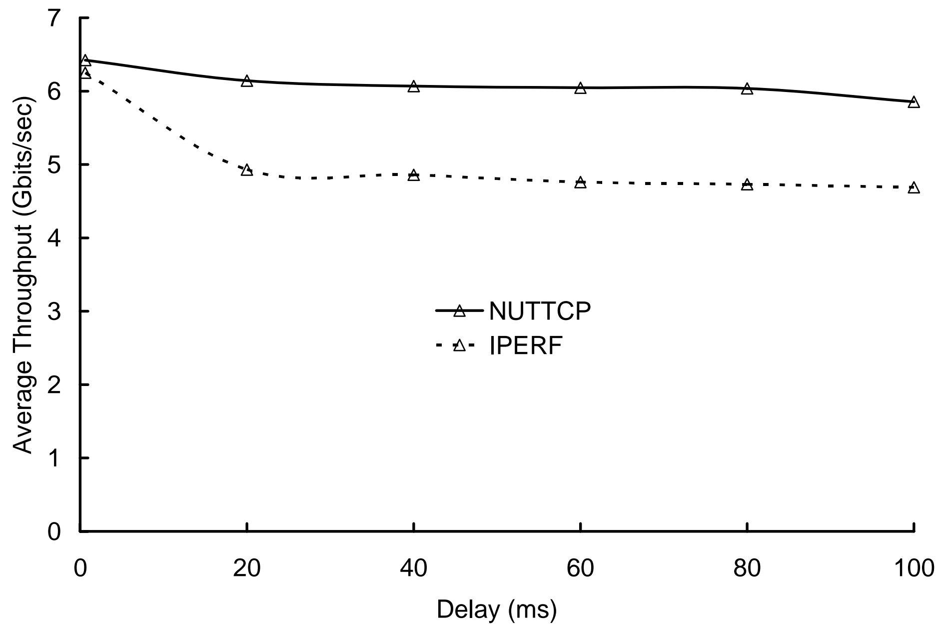

Figure 3.4 Performance of NutTCP and Iperf over a Similar Scenario

3.3 Linux Tuning for 10G Networks

During the entire course of this work, we intended to provide a common platform for all the experiments so that we can have a fair comparison of all the transport protocols that we tested. In this section, we deal with the parameters that help in utilizing the maximum bandwidth. Any Linux kernel comes with the default TCP related parameter values, which may not be suitable for the high-speed scenario. In this section we give a brief description of these parameters and also the optimal values for these parameters.

3.3.1 Setting BDP

As discussed in section 2, the bandwidth delay product of the link determines the amount of data which can be in transit in the network. For an overall bandwidth of 10Gbits/sec and an RTT of 200ms, the BDP can be calculated as follows:

*BDP = 10,000,000,000/8GB/sec * 240/1000 sec = 250,000,000Bytes = 250Mbytes*

For tuning the network to get optimal performance, the TCP send and receive socket buffer sizes should be set to a large value to completely utilize the high-speed link. This of the buffer sizes helps TCP to eliminate buffer size restrictions on TCP throughput thus increasing the overall throughput. Also an important point to note is that by setting high values for the buffers, we are restricting TCP to go beyond a particular value and thus avoid the drastic multiplicative decrease part to come into picture. We carried out several experiments to tune the following parameters which resulted in achieving the maximum throughput which could be offered by our end systems, the hardware being the bottleneck.

3.3.2 Setting other Kernel Parameters

Receive and Send window (rmem_max and wmem_max): These values determine the maximum receive and send window which corresponds to SO_SNDBUF and SO_RCVBUF. In other words, these values determine the amount of data which can be transferred before the server stops and waits for acknowledgements of received packets. It is generally advisable to keep these values equal to somewhere between two times or same as the bandwidth delay product (BDP) of the link. The default values was 64K which is far too less for the high-speed case scenario. For all our experiments which required tuning we set this value to 500MB based on the BDP calculations above.

Socket Buffer Memory (tcp_rmem and tcp_wmem): With new values of the sender and receive window, the values for socket buffer memory should also be increased than what was originally configured by the kernel. There are three values which correspond to the minimum, default, and the maximum number of bytes to use. The default values for the

receive window are 4096, 87380, and 4194304 and for the send window, the default values are 4096, 16384, and 4194304. We set the minimum, default, and the maximum value to be 500MB for all the experiments.

Transmission Queue Length (txqueuelen): This parameter is used in the kernel to regulate the size of the queue between the kernel and the Ethernet layer. This size of the queue has to be carefully selected. A high value of the transmission queue length may result in an increase in the number of duplicate packets which should be avoided. The default value of this is only 100 and this should be changed to at least 10,000.

Receiver Queue Size (netdev_backlog): This parameter determines the receiver queue size. If this value is set to a low value then the queue will build up in size when the interface receives packets faster than the kernel can process. The default value is only 300 and for a 10G network, to minimize the number of lost packets on the receiver side rather than on the network, this value should be set to at least 3000.

TCP Cache Parameter: The kernel caches the TCP network transfer statistics and tends to use it for the next connection. This value should be cleared before every connection and it ensures that the ssthresh value need not be discovered for every connection.

Route Cache Feature: The route cache 'feature' for high-speed links should be eliminated for a better overall performance.

Disabling SACK: For high-speed links, if there are too many packets in flight and a SACK event occurs, it takes a very long time to detect the SACKed packet which results in TCP timeout which in turn changes the values of the congestion window to 1. For a better performance and to avoid this problem on high speed links, this SACK option has

to be disabled. An important point to note is that the SACK event only occurs when the TCP window size is greater that 20MB.)

TCP Segmentation offload: The TCP segmentation offload should be disabled for making TCP more stable over high-speed networks.

There are a number of other parameters which need to be tuned for a better performance like increasing the PCI latency timer for the NIC card, using SMP affinity in case of dual CPUs so that the number of interrupts are reduced and all the processes related to the card are handled by a single CPU, changing the sizes of the default buffer sizes for the non-TCP sockets, increasing the total amount of memory available for the socket input/output queues etc. These values can be changed but do not affect the performance significantly.

3.3.3 Using Jumbo Frames

Ethernet, since the 1980s has been using 1500 byte frame sizes. Even today the default size of the Ethernet frame has been kept the same. Over gigabit networks, this MTU size is considered to be very less and has to be increased to at least 9000Bytes. Another point to note is that IPv4 can support frames to up to 64KB but the 32 bit CRC limit of the Ethernet will loose its effectiveness over 12KB frame sizes [26].

With High-speed networks have enough capacity to handle large packets because of the large bandwidth delay product of the link. For all the experiments which required tuning, we set the packet size as 9K. This is the present limitation of Linux kernel we are using (2.6.17). Using large frame sizes results in reduced fragmentation overhead and hence translates to a lower CPU overhead on hosts. Also, because of more aggressive TCP dynamics, it results in also leads to a greater throughput and a better response to certain type of losses.

After applying the settings discusses in the above sections, we noticed a considerable change in the performance of TCP over high-speed networks as shown in Figure 3.5 below. The MTU used for the experiment was 9K. More results after tuning are discussed in the next section.

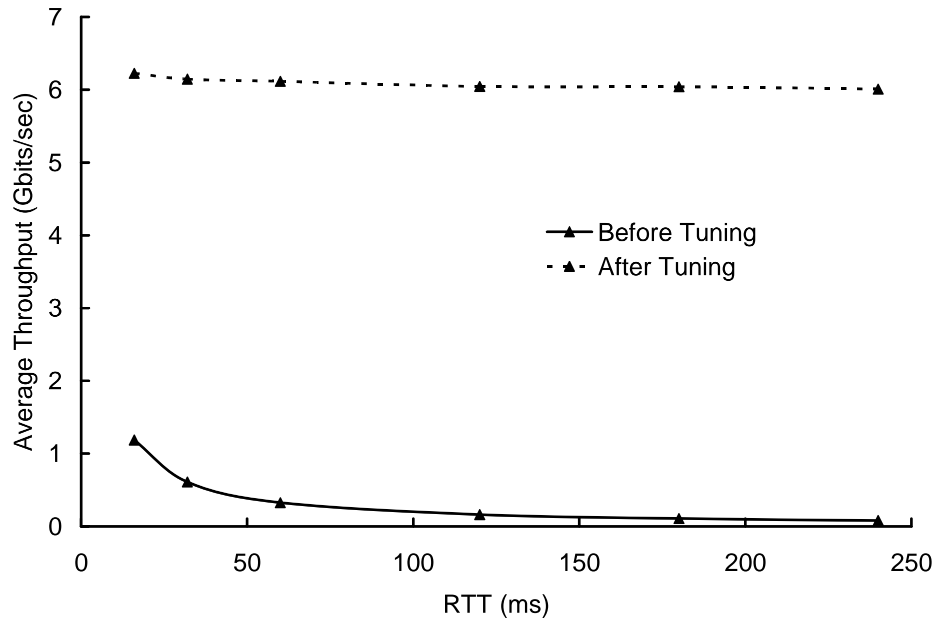

Figure 3.5 Performance of TCP before and after Tuning

Another point to note is that our end systems have good processing speeds and were capable of handling data over 10Gbits/sec but the PCIX slot where the 10GbE card was connected was the bottleneck. PCIX 133Mhz is known to support maximum speeds of only up to 7.5Gbits/sec. Another bottleneck was the AMD [26] 8131 chipset that we have on our end systems. In this chipset, the number of outstanding PCI split transactions and the write burst size is limited. Any attempt made by us to change these settings resulting in system crashes and hangs so we ended up using the default values. Because of this, we were restricted to a maximum speed of up to 6.51Gbits/sec which we achieved over less RTT.

3.4 Applications of High Speed Networks

With the emergence of video conferencing, high-performance grids, generation of huge amounts of simulation data which at times need to be visualized at real-time between locations which may be hundreds and thousands of miles away; the necessity of transferring huge amounts of data at least amount of time possible has become a necessity. These applications involve communication patterns which may vary from point-to-point to multipoint-to-point. An example of a multipoint-to-point (or point-to-multipoint) scenario can be video conferencing. Data visualization over remote sites in which the server processes data from remote clients can also be considered as an example of a multipoint-to-point link. A point-to-point communication pattern may exist in data transfer.

All these applications have different requirements in terms of the transport protocol characteristics. For example, retransmission of lost packets may be necessary in some cases whereas in other applications like video conferencing, retransmission of lost packets will only make the situation worse. These characteristics may also differ based on the communication pattern used in the application. As discussed in [28], all these applications which use TCP have the following characteristics:

a) Connection Oriented vs. Connection less (TCP vs. UDP)

b) Reliability vs. unreliability (large file transfers vs. video conferencing, real-time streaming, and Visualization)

c) Latency and Jitter: (small message passing vs. large file transfers)

d) Sequenced Vs Unsequenced transfer: (application-level message passing vs. large file transfer)

e) Rate Shaping Vs Best Effort (based on throughput requirements)

f) Fairness (among different transport protocols)

g) Congestion Control (some applications like video conferencing may not require congestion control)

4. PERFORMANCE EVALUATION

4.1 Experimental Evaluation of TCP in High-Speed Scenario

In this section we observe the behavior of TCP Reno under different experimental conditions with an aim to improve its performance over high BDP networks after tuning the Linux parameters.

Note that all the experiments discussed in this chapter are based on memory to memory transfers and have been run for a total time of 200 seconds.

4.1.1 Effect of Using Large Frames

This section talks about the results that we got by using large frame sizes which was discussed in section 3.3.3. In Figure 4.1 below, we compare the performance of TCP by

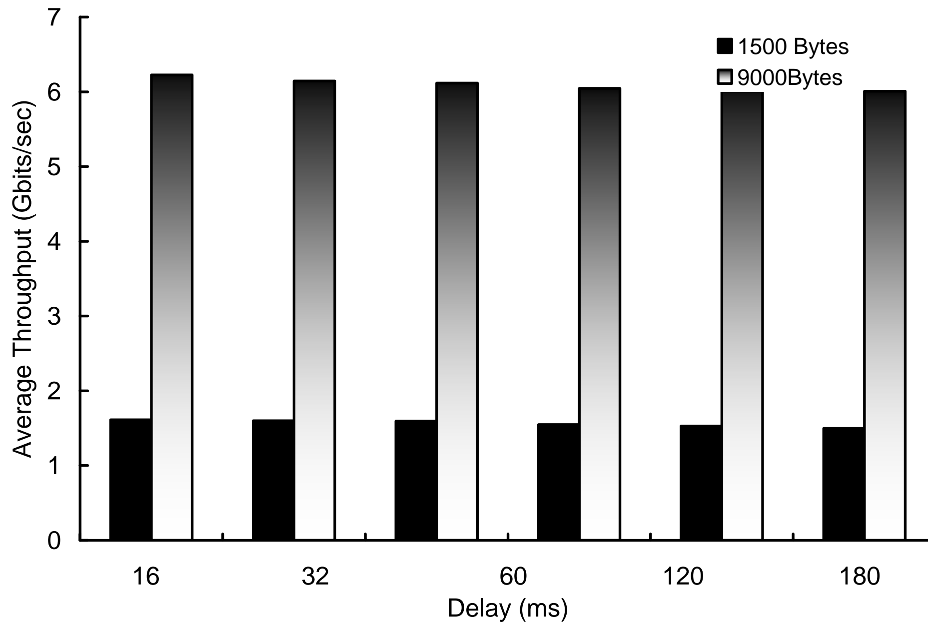

Figure 4.1 Performance Comparison of TCP by using 1500 and 9000 MTU

varying the size of MTU over different delay. The sender and receiver buffer sizes of TCP have been tuned to 500MBytes. As illustrated in the figure, there is a considerable

amount of improvement in the throughput of TCP in the case of 9KBytes frames compared to the 1.5KBytes frame size.

We performed another experiment to study the effect of jumbo frames over the performance of TCP. Figure 4.2 shows the performance of TCP as we increase the frame size from 1500 to 9000Bytes in steps of 1500. The delay has been kept constant and has been set to 120ms. The TCP sender and receiver sizes have been tuned to 500MBytes for best performance. As expected, the performance of TCP improves as we increase the frame size. This shows that for high-speed networks, it is mandatory to use frames of larger size instead of using the default size of 1500Bytes.

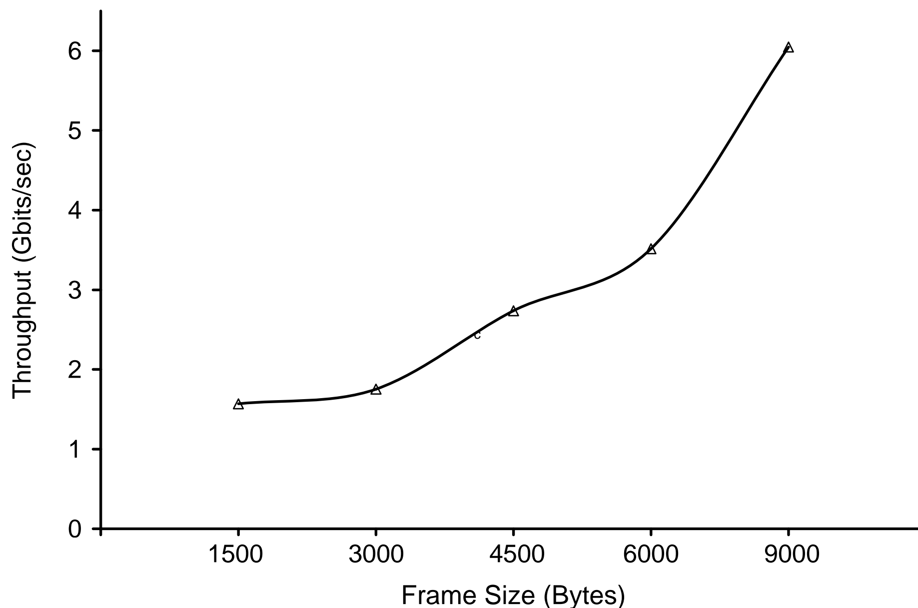

Figure 4.2 TCP Performance over Different Frame Sizes

4.1.2 Effect of Using Large BDP

This sections deals with the effects of using a large sender and receiver buffer size (BDP) as discussed in section 3.3.1. Figure 4.3 below shows the effect of the sender and receiver buffer size on the average throughput of TCP which is set by calculating the BDP of the

link as shown in the last chapter. We calculated the average throughput for flows having 120ms and 240ms delay over buffer sizes of 10, 50, 100, 200, 500, and 1GB. The 120ms flow shows a peak performance for any receiver buffer size greater than 400MBytes. The TCP flow with a delay of 240ms shows peak performance for a receiver buffer size between 500 and 600MBytes. After that, the performance degrades because of the overhead caused on the end systems. From this figure, we can see the importance of tuning the buffer sizes to a right value to get the optimal performance. Based on this result, we have set the buffer size to 500MBytes for all our experiments which involve tuning.

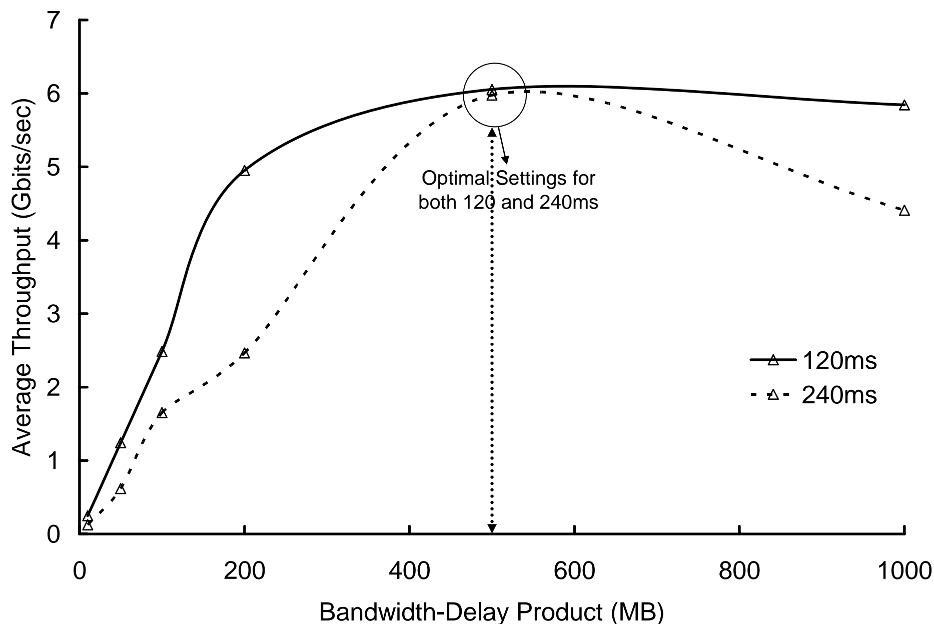

Figure 4.3 TCP Performance over Different Buffer Sizes

In Figure 4.4 below, we compare the performance of three TCP flows having different receiver buffer sizes over RTT. The delay for the experiment has been kept constant and set to a value of 120ms. The two flows with better performance have receiver buffer sizes of 100MB and 500MB. The flow with the default value of the buffer

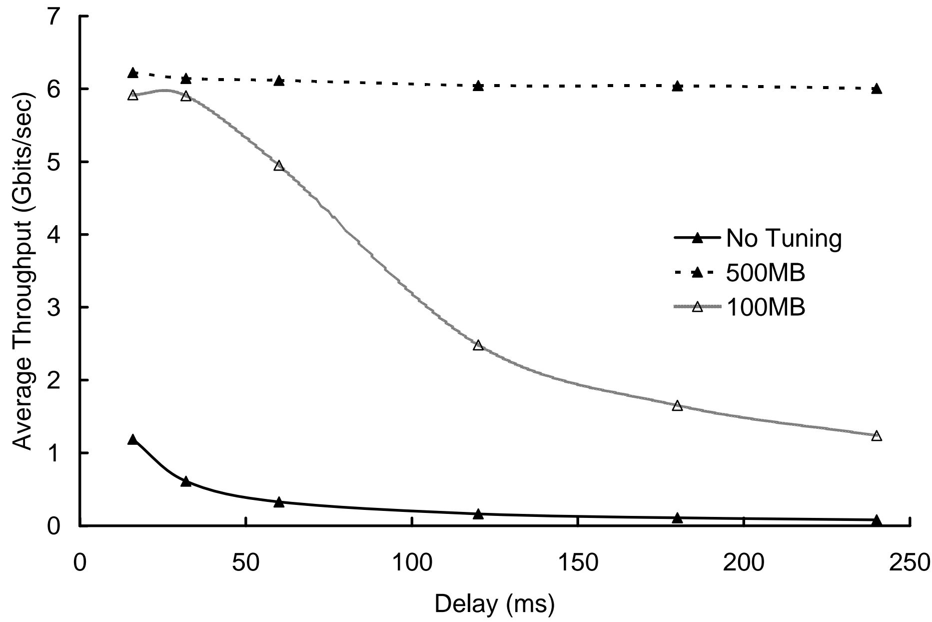

Figure 4.4 TCP Performance over Different Delay

size (64KB) does not show good performance. It again proves that the default parameter values in the kernel are not suitable for the high-speed scenario and needs to be changed to a higher value in order to get a high throughput.

4.1.3 Effect of Using the Window Scaling Option

Figure 4.5 shows the average throughput of TCP after we disable the window scaling option which was first defined in RFC 1323 [27]. Window scaling allows us to scale TCP windows over high-speed networks. The default size of the TCP window, as defined in its header, cannot be larger that 2^{16} bytes (~65kb). With window scaling option enabled, the window size can be scaled to a larger size thus reducing the bandwidth losses. For the experiment, the overall delay is varied from 16ms to 240ms. From the figure it can be observed that there is a huge improvement in the overall TCP performance when the option is enabled (the window scaling option is enabled by default).

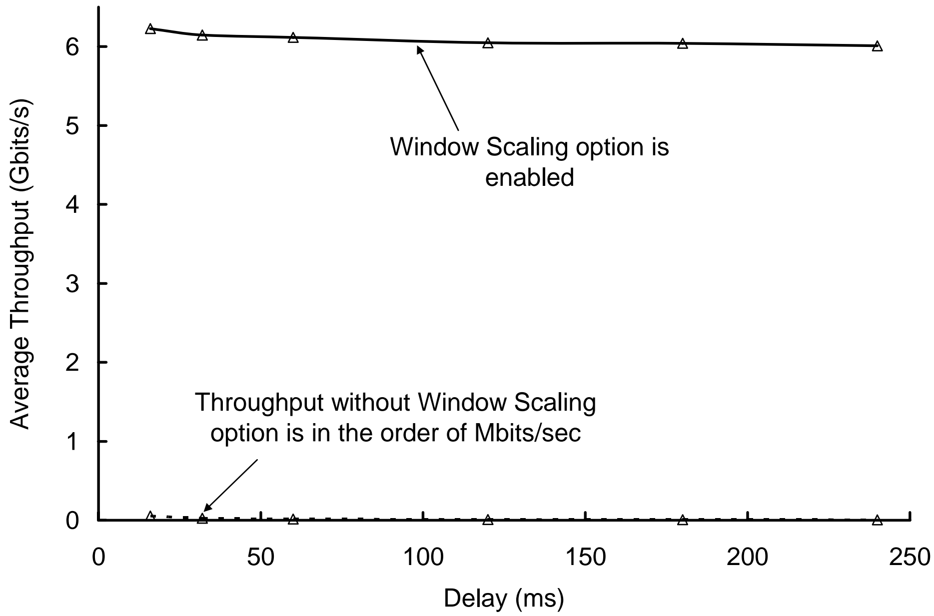

Figure 4.5 TCP Performance after Enabling the Window Scaling Option

4.1.4 Effect of Using Parallel TCP Flows

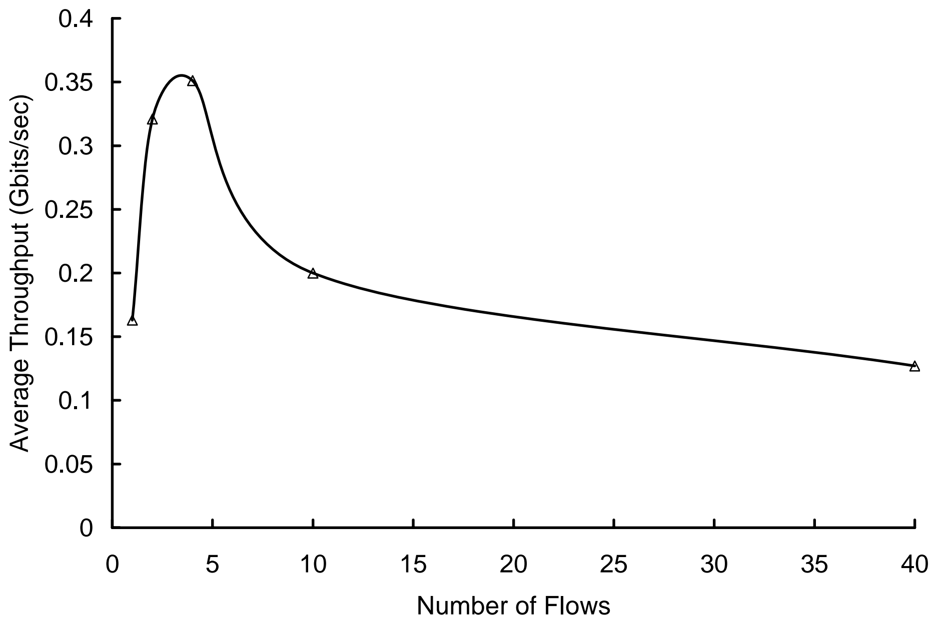

Figure 4.6 TCP Throughput with Increasing Number of Parallel Flows

Figure 4.6 shows the effect of increasing the number of flows on the average throughput of TCP. The overall delay for all the flows has been set to 120ms. Also, we ran this

experiment with the default values of the Linux kernel i.e. without tuning any buffer or frame size. As depicted in the figure, TCP shows the best performance with when the flows are between 4 and 6 in this scenario. After that, the average throughput begins to decrease. One of the reasons for the decrease could be the overhead on the end system which is caused due to the increase in the number of flows.

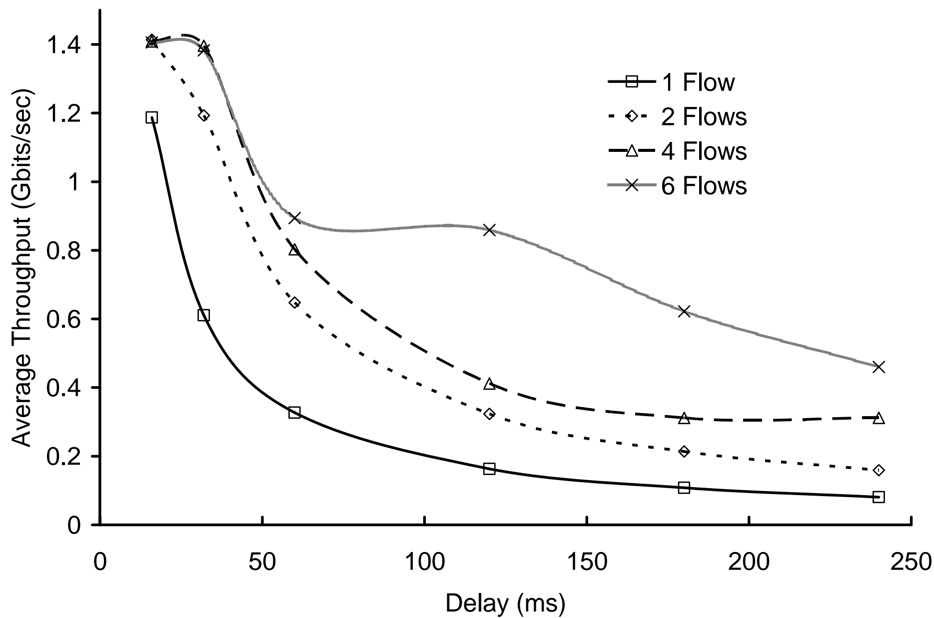

Figure 4.7 Average Throughput Vs Delay for Different Number of TCP Flows

Figure 4.7 shows the average throughput of TCP over varying delay after increasing the number of flows. Again, we have not done any parameter tuning and the throughput values are based on the default parameters. The RTT has been varied from 16ms to 240ms and the frame size has been kept to a default value of 1500Bytes. This figure is an extension of the earlier graph and as expected, the increase in the number of flows results in the increase in the overall average throughput. This figure also shows that the optimal number of flows may vary for different scenarios and the number of flows used for a particular scenario should be tuned accordingly.

4.1.5 Effect of Dynamically Increasing and Decreasing RTT on TCP

We also emulated a scenario in which we could have a case where an RTT from the sender to the receiver could change dynamically. This may be the case for a multipoint-to-point scenario where a client may send data to remote locations which may be separated by different distances and thus have different RTTs. For example, in data visualization, the client may have to send data to a remote machine which could be located in Japan (~200ms delay) and a remote machine somewhere in the US (~60ms delay). This could also be the case in which there may be a huge background traffic or overhead on the end systems and the latency and jitter keep varying.

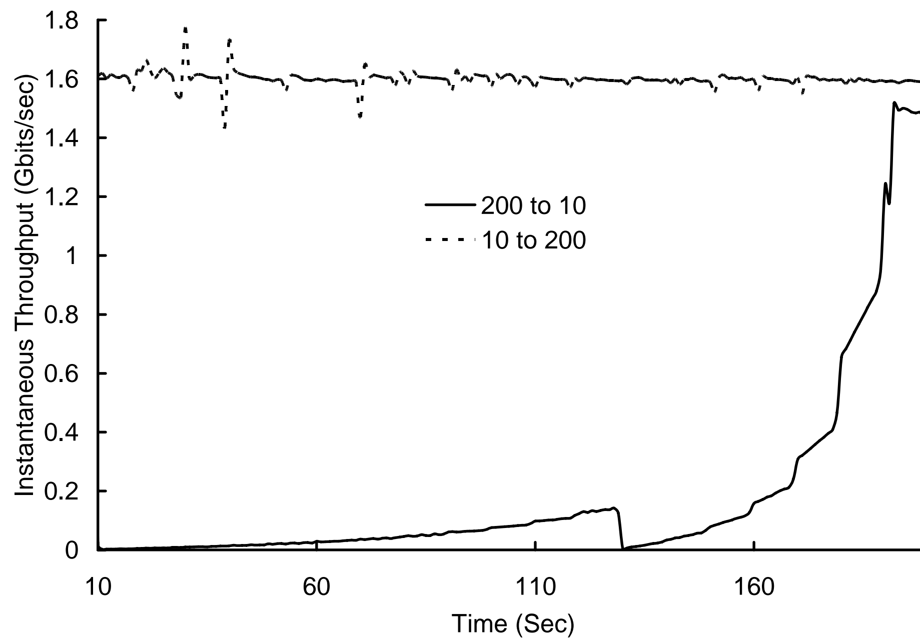

Figure 4.8 Instantaneous Throughput of Two Flows with dynamically varying RTTs

Two cases have been shown in figure 4.8. For the 200 to 10 case, the RTT has been decreased from 200 to 10 after every interval of 10 seconds. TCP is not able to utilize the link in this case in spite of having 9000Byte buffer size and also a receiver buffer size of 500MB. There is an improvement in the throughput after the RTT becomes

approximately 40ms. Interestingly, in the other case, when the RTT varies from 10 to 200ms the instantaneous throughput is constant which shows that TCP takes a long time to discover the available capacity.

In this section, we have seen that the performance of TCP can be greatly improved by tuning the parameters to an optimal set of values. TCP is able to show a stable average throughput in most of the cases due to a large receiver buffer size and the nature of the memory to memory transfer that we have used to conduct the experiments. The data at the receiver side is read (in our case it is discarded as soon as it reaches the receiver) at the same rate as the data is sent by the sender. Therefore, no losses can be observed at the receiver side and thus the congestion window value also remains the same.

4.2 Performance Comparison of TCP Variants

In this section, we compare the various TCP based high-speed protocols over different scenarios. Also, all the protocols have their own default setting and no changes have been made.

4.2.1 Performance in No Loss Scenario

Figure 4.9 shows the performance of the different TCP variants over varying delay. The receiver buffer size for the cases has been fixed to 500MB. 9000Bytes frame size has been used for all the cases and the experiment has been emulated for 200 seconds. Interestingly, we can see that all protocols give a very similar performance. As discussed before, in this case, for all the protocols we have restricted the amount of buffers the protocols may use. This makes the protocol have a fixed value of the congestion window thus restricting the protocol to use large size congestion windows and hence avoid any

form of packet loss. For larger delays, CUBIC and BIC show a little less throughput compared to the other protocols because of their slow congestion window growth policy which may also be a result of their less aggressiveness which in turn means that they are fairer towards the other protocols.

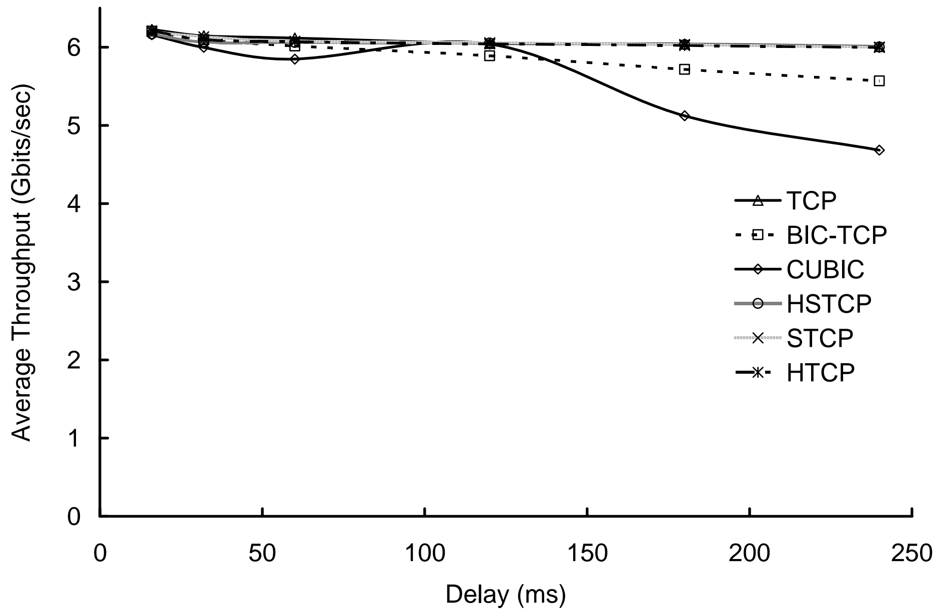

Figure 4.9 Average Throughput Vs Delay for different of TCP Variants

4.2.2 Protocol Performance over Packet Loss

Figure 4.10 shows the effect of loss on the behavior of different TCP based congestion control algorithms. High loss environments over high-speed networks can be considered to be a very rare scenario in real networks, but we had to force losses to observe the protocol behavior more closely. Having only a back-to-back link restricted us to observe the protocol behavior without any loss case. As shown in the Figure 4.10, for a loss of 10^{-3}, all the protocols perform very poorly as expected. But as we move from a packet loss of 10^{-3} to 10^{-5}, the protocols show an improvement, in particular, STCP which shows

a far better performance compared to the other protocols. This is because of the MIMD algorithm it uses. TCP, because of its AIMD approach, is not able to scale its congestion window to a larger value. All the other protocols also do not perform well in this scenario showing that they are less aggressive in their window growth function which in turn means that they are fairer towards other flows which may not be needed in the case of back-to-back links.

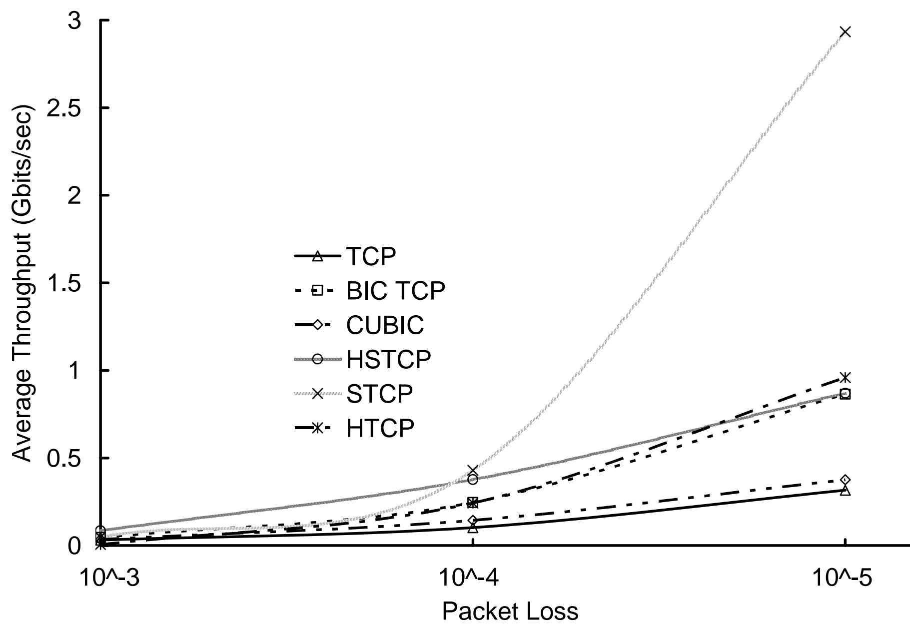

Figure 4.10 Average Throughput Vs Packet Loss for different of TCP Variants

4.2.3 Protocol Performance over 10^{-5} Packet Loss

This scenario is presented in Figure 4.11. Even in this case, STCP achieves a better throughput compared to the other protocols though it degrades in its performance as the delay increases to 240ms. HSTCP and BIC TCP show a good performance for shorter delays; but as the delay is increased, their performance becomes comparable to that of TCP. All the other protocols show a relatively bad performance. Further, the effect of loss on the congestion window behavior is shown in figures 4.12 and 4.13.

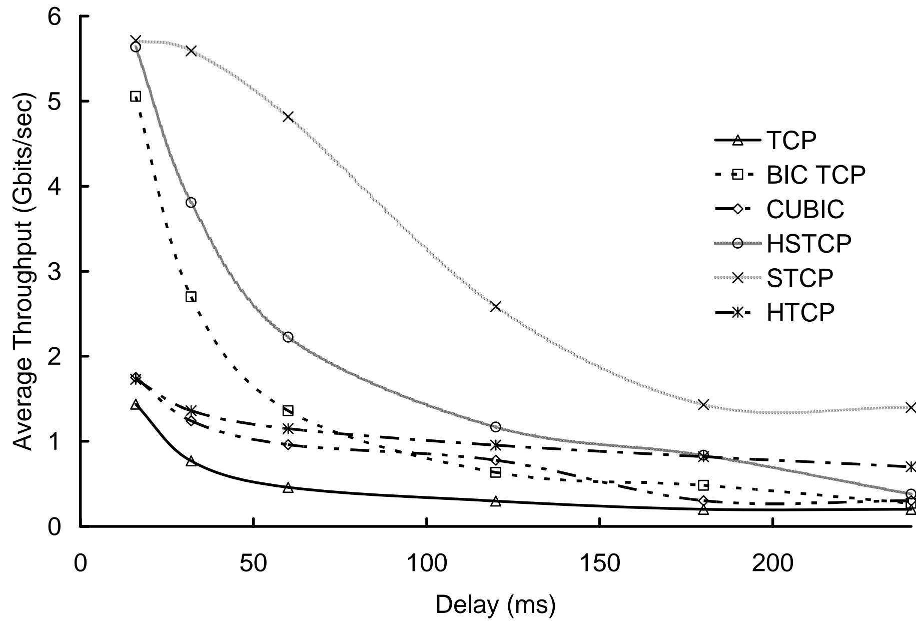

Figure 4.11 Average Throughput Vs Delay for Different of TCP Variants

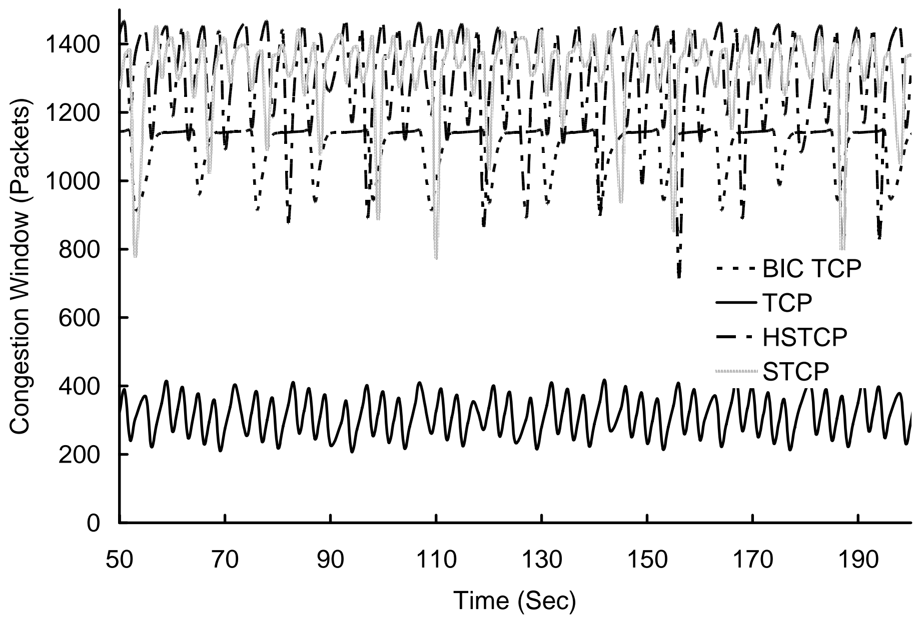

Figure 4.12 Congestion Window Vs Delay for Short Delay

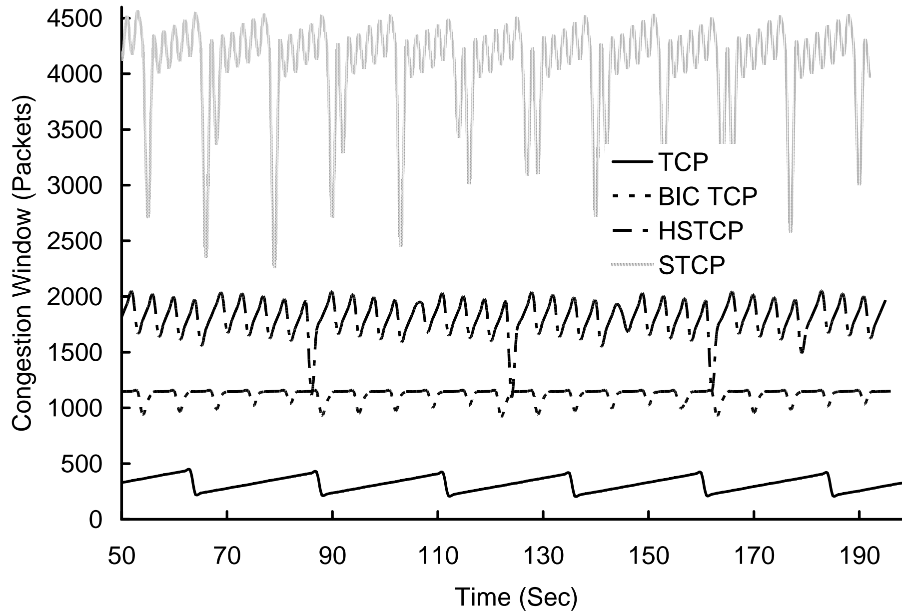

Figure 4.13 Congestion Window Vs Delay for Long Delay

We have plotted the congestion window for only TCP, BIC TCP, HSTCP, and STCP to make it more readable. At short delays (16ms) the average throughput of BIC TCP, HSTCP, and STCP are equal as shown in Figure 4.11 and hence their congestion window seem to overlap. TCP's AIMD approach restricts its congestion window from reaching a high value thus restricting the overall throughput. Figure 4.13 shows the congestion window behavior of the protocols over long RTT (120ms). The congestion window of Scalable TCP scales to a large value compared to the other protocols which can be attributed towards its MIMD approach. BIC TCP, because of its binary search approach, is not aggressive compared to other protocols like HSTCP and STCP.

In the high-speed applications which use the back-to-back topology, fairness may not be a major concern compared to the aggressiveness in utilizing the link. As seen from the above discussion, Scalable TCP can be a better option but it has been shown in [4] that Scalable TCP is not fair compared to the other protocols. This motivates us to come

up with a better algorithm over back-to-back links which follows the multiplicative increase approach and also behaves in a fair manner at the same time.

5. CONCLUSION

Our first goal was to maximize the performance of TCP on 10Gbits/sec link by tuning the default Linux TCP parameters. We were successful in achieving the maximum physical limit of achievable throughput with the PCIX 133Mhz bus and the AIMD 8131 chipset on our host systems restricting the bandwidth to 6.51Gbits/sec. All the other TCP variants we tested too reached the maximum available throughput. BIC and CUBIC took a longer time than the other protocols because of their less aggressive window growth policy. Our next aim was to make TCP and other protocols sustain the same maximum throughput over long delays. We were able to do successfully do that by having a large value of the sender and receiver buffer. We tested the protocols over long RTT (240ms) and most of them, except for CUBIC, were able to fully utilize the link capacity (6.51Gbits/sec in our case).

We also explored the effect of using parallel TCP flows over high-speed links. Interestingly, the number of flows which gave the maximum performance had to be carefully chosen as increasing the number of flows over a certain threshold added to the overhead at the host systems. We also tested the protocols over different rates of packet loss to study their window growth policy. We understand that packet loss in high-speed networks is a rare phenomenon but we were restricted by having only back-to-back machines in our case and emulated packet loss was the only option to study the window growth policy of the protocols. Under a high packet loss scenario, STCP was able to give the best performance because of the MIMD algorithm that it follows. TCP in this case gave the worst performance because of its AIMD nature.

We like to conclude that though back-to-back topology may not be adequate to compare the protocols, high-speed links over networks like NLR and LONI can also be considered as back to back and studying the performance and behavior of these variants over such links is very important and can help us utilize the bandwidth over such links in an effective manner. In the future, we would like to test these variants over high-speed multipoint-to-point networks and also define a complete set of performance metrics for high-speed networks which can help compare all the TCP variants fairly.

REFERENCES

[1] S. Floyd, S. Ratnasamy, and S. Shenker, "Modifying TCP's Congestion Control for High Speeds," http://www.icir.org/floyd/hstcp.html, May 2002

[2] S. Floyd, "HighSpeed TCP for Large Congestion Windows," IETF, INTERNET DRAFT, draft-floyd-tcp-highspeed-02.txt, 2002

[3] T. Kelly, "Scalable TCP: Improving Performance in HighSpeed Wide Area Networks," Submitted for publication, December 2002

[4] L. Xu, K.Harfoush, and I. Rhee, "Binary Increase Congestion Control for Fast Long-Distance Networks," in IEEE INFOCOM, Mar. 2004.

[5] Injong Rhee and Lisong Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," in PFLDnet, 2005.

[6] C. Jin, D. X. Wei and S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," In *Proceedings of IEEE INFOCOM 2004*, March 2004

[7] R. Shorten, and D. Leith, "H-TCP: TCP for High-Speed and Long-Distance Networks," *Second International Workshop on Protocols for Fast Long-Distance Networks*, February 16-17, 2004, Argonne, Illinois USA

[8] LONI, "http://www.loni.org/loni/loniweb.nsf/index, "

[9] NLR, "http://www.nlr.net/"

[10] NREN, "http://www.nren.nasa.gov/"

[11] ESnet, "http://www.es.net/"

[12] ABILINE, "http://abilene.internet2.edu/"

[13] GEANT, "http://www.geant.net/"

[14] H. Sivakumar, S. Bailey, and R. L. Gorssman. ***Psockets****: The case for applicaiton-level network striping for data intensive applications using high speed wide area networks*. In Proc. of SC2000.

[15] The Globus Project. ***GridFTP****: Universal data transfer for the Grid*. http://www-fp.globus.org/datagrid/deliverables/C2WPdraft3.pdf, Sept. 2000.

[16] Dina Katabi, Mark Handley, and Charles Rohrs, "*Internet Congestion Control for Future High Bandwidth-Delay Product Environments*." ACM Sigcomm 2002, August 2002. URL "http://ana.lcs.mit.edu/dina/**XCP**/".

[17] D. Lu, Y. Quao, P. Dinda, and F. Bustamante, "*Modeling and taming **parallel TCP** on the wide area network*," in Proceedings of the 19th IEEE International **Parallel** and Distributed Processing Symposium, Apr. 2005.

[18] Thomas J. Hacker, Brian D. Noble, and Brian D. Athey, "*Improving throughput and maintaining fairness using **parallel tcp***," in Proceedings of the IEEE INFOCOM, 2004.

[19] Chelsio Communications, "http://www.chelsio.com/"

[20] NIST NET, "www-x.antd.nist.gov/**nistnet**/"

[21] The Network Simulator, "www.isi.edu/nsnam/ns/"

[22] OPNET Technologies, "http://www.opnet.com/"

[23] Network Emulator,"http://linux-net.osdl.org/index.php/**Netem**"

[24] IPERF, "dast.nlanr.net/Projects/**Iperf**/"

[25] NUTTCP, "ftp://ftp.lcp.nrl.navy.mil/pub/nuttcp/"

[26] Advanced Micro Devices, "www.amd.com"

[27] RFC 1323 – TCP Extensions for High Performance, "ww.ietf.org/**rfc**/**rfc1323**.txt"

[28] S. Iren, P. Amer, and P. Conrad, "The Transport Layer: Tutorial and Survey." ACM Comp. Surveys, vol. 31, no. 4, pp. 360-405, Dec 1999.

VITA

Yaaser Ahmed Mohammed was born in Andhra Pradesh, India on 22nd June 1981. He earned his primary and secondary education from Hyderabad Public School in Hyderabad, Andhra Pradesh. After finishing his high school, he wrote a state wide competitive entrance examination for engineering and was ranked among the top 3% in his entire state. After qualifying this examination he got admission to, department of Information Technology, Osmania University. He received his Bachelor of Engineering in Information Technology from Osmania University, Hyderabad, India, in spring 2004. After his graduation, he came to the United States of America to pursue his master's degree. He then joined the graduate program at Louisiana State University, Baton Rouge, in August 2004. He is a candidate for the degree of Master of Science in Systems Science to be awarded at the commencement of Fall, 2006.