2004

Performance analysis of TCP/IP over high bandwidth delay product networks

Subodh Kerkar
University of South Florida

Follow this and additional works at: http://scholarcommons.usf.edu/etd

Part of the American Studies Commons

Scholar Commons Citation
http://scholarcommons.usf.edu/etd/1107

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
Performance Analysis Of Tcp/Ip Over High Bandwidth Delay Product Networks

by

Subodh Kerkar

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science Department of Computer Science & Engineering College of Engineering University of South Florida

Major Professor: Miguel Labrador, Ph.D.
Srinivas Katkoori, Ph.D.
Dewey Rundus, Ph.D.

Date of Approval:
July 6, 2004

Keywords: highspeed, fast-tcp, optical, evaluation, reno

© Copyright 2004 , Subodh Kerkar
# TABLE OF CONTENTS

## LIST OF FIGURES ii

## ABSTRACT iv

### CHAPTER 1 INTRODUCTION 1

1.1 Motivation for Present Work 1
1.2 Contribution of this Thesis 5
1.3 Outline of this Thesis 5

### CHAPTER 2 LITERATURE REVIEW 7

2.1 The Transmission Control Protocol (TCP) 7
2.2 TCP Versions 10
  2.2.1 Old Versions 10
  2.2.2 TCP for High Bandwidth-Delay Product Networks 23
2.3 Router Queueing Techniques 26
  2.3.1 Droptail 26
  2.3.2 Random Early Detection (RED) 28

### CHAPTER 3 METHODOLOGY 31

3.1 Simulation Topology for a HBDP Link (Single source/single sink) 31
3.2 Simulation Topology for Fairness to Itself and Others Scenario (Two Sources/ Two Sinks) 33

### CHAPTER 4 PERFORMANCE EVALUATION 35

4.1 Single Source – Single Sink Topology 35
4.2 Two Sources – Two Sinks Topology (Fairness to Others) 46
4.3 Two Sources – Two Sinks Topology (Fairness to Itself) 57

### CHAPTER 5 CONCLUSIONS 64

## REFERENCES 66
LIST OF FIGURES

Figure 1. Variation of TCP’s Congestion Window With Time 10
Figure 2. Scalable TCP Congestion Window Properties 26
Figure 3. Response Curves of Scalable TCP and Regular TCP 27
Figure 4. Drop Probabilities of Different RED Modes 29
Figure 5. Simulation Topology for One Source – One Sink 32
Figure 6. Simulation Topology for 2 Sources – 2 Sinks (Fairness to Itself) 33
Figure 7. Simulation Topology for 2 Sources – 2 Sinks (Fairness to Others) 34
Figure 8. Normalized Throughput of Protocols Under Consideration 37
Figure 9. TCP Sequence Numbers for Link Bandwidth 1Gbps 37
Figure 10. Congestion Windows of All the Protocols When Link Speed 1Gbps 39
Figure 11. Slowstart Times 42
Figure 12. Packet Loss Rate 43
Figure 13. Recovery Time 44
Figure 14. Performance of HSTCP Against TCP Protocols 48
Figure 15. HSTCP Fairness to Vegas and Westwood 49
Figure 16. Performance of TCP Westwood 51
Figure 17. Behavior of TCP Westwood’s Cwnd and Ssthresh when the Bottleneck Link is Set to 1Gbps 51
Figure 18. Performance of Vegas 52
Figure 19. Performance of Scalable TCP With Other Protocols 55
Figure 20. Performances of Tahoe, Newreno, Reno and SACK TCP 57
Figure 21. Fairness to Itself (Newreno, Reno, SACK, Scalable, Tahoe) 62
Figure 22. Fairness to Itself (Vegas) 63
PERFORMANCE ANALYSIS OF TCP/IP OVER HIGH BANDWIDTH DELAY
PRODUCT NETWORKS

Subodh Kerkar

ABSTRACT

In today’s Internet scenario, the current TCP has performed reasonably well. As the Internet has scaled up in load, speed, size and connectivity by the order of six over the past fifteen years, the TCP has consistently avoided severe congestion throughout this same period. Applications involving high performance computings such as bulk-data transfer, multimedia Web streaming, and computational grids demand high bandwidth. These applications usually operate over wide-area networks and, hence, performance over wide-area networks has become a critical issue. Future applications will need steady transfer rates in the order of gigabits per second to support collaborative work. TCP, which is the most widely used protocol, is expected to be used in these scenarios. It has been shown that TCP doesn’t work well in this new environment, and several new TCP versions have been developed in recent years to address this issue.

To date, there has not been a performance evaluation of various TCP protocols. In this thesis, various TCP versions — Tahoe, Reno, Newreno, Vegas, Westwood, Sack, Highspeed TCP, Scalable TCP — have been evaluated for their performance over high
bandwidth delay product networks. It was found that the flow and congestion control
mechanism used in TCP was unable to reach full utilization on high-speed links. Also
discussed in this Thesis are fairness issues related to these new protocols with respect to
themselves and with others.
CHAPTER 1
INTRODUCTION

1.1 Motivation of Present Work

Today’s Internet scenario is on the brink of testing the widely used communication protocols, the Transmission Control Protocol (TCP). Applications needing connections with steady transfer rates are consistently showing up on the horizon. In the first International Workshop on Protocols for Fast Long-Distance Networks [1], several presentations put forward the point of future network applications requiring steady transfer rates in the order of gigabits per second to support collaborative work. Although the raw transmission bit rate of next generation networks will definitely support these high speeds, it is unknown whether the communication protocols will become the performance bottleneck. This is the case of TCP when running over optical networks using the Dynamic Bandwidth on Demand (DBOD) service or high bandwidth-delay product networks. Many communities use such networks and need to distribute a substantial amount of data over them. The large datasets collected by the High Energy Physics, Bioinformatics and Radio Astronomy communities require global distribution for the data to be analyzed effectively. This is one example of such a network. Internet paths operating in this region are usually referred to as ‘Long Fat Pipes’. High capacity packet satellite channels — e.g. DARPA’s wideband net — are called LFNs. Terrestrial
fiber-optical paths also fall into the LFN class, which is moving out of the domain for which TCP was originally crafted.

Since TCP is expected to be used in this scenario, its most important aspect — Flow and Congestion control mechanism — should perform as the network evolves. As the bandwidth or the delay goes on increasing, this mechanism causes problems as TCP reacts adversely in such cases. According to Sally Floyd [3], TCP faces three main difficulties: Bit Error Rate (BER), slow-start mechanism, and congestion avoidance mechanism. First, the Bit Error Rate (BER) of the links on which high data transfer is expected should be very small, much smaller than the current BER. Secondly, The slow-start phase of the TCP’s congestion control mechanism sees the congestion window increase exponentially. In high bandwidth-delay product scenarios, the congestion window increases to a very large value and a large number of packets is dropped once the channel capacity is filled. And finally, TCP has been shown to waste a considerable amount of bandwidth in its congestion avoidance phase when the window increases linearly. Hence, in high capacity link and long propagation delays, it will take TCP a very long time to fill up the whole pipe.

As the high-bandwidth network becomes more widely used, the problems of the Additive Increase Multiplicative Decrease (AIMD) algorithm of the TCP becomes more apparent. For many years, network research has been seeing improvements in TCP efficiency and stability. As a result, different versions of TCP including Tahoe, Reno, Vegas [6], Sack [7,8] and Westwood [9] have been developed. These variants brought about a significant amount of improvement, as a result of their improved congestion control mechanism, selected acknowledgement, and fast recovery; but all these variants
had the same unchanged window-based algorithm as is specified in RFC 2581 [10].

There have been many other TCP variants that employ techniques other than the window-based algorithm. Mechanisms that employ rate-based techniques involve the controlling of the congestion window, based on feedback received from routers. But these kinds of techniques are unlikely to be incorporated in the future, as they require the modifications of routers by the ISPs. In the last two to three years, researchers have come upon many proposals for the modification of TCP on the sender’s side for its use on High Bandwidth-Delay Product links. A few examples are Sally Floyd’s High-speed TCP [3], Kelly’s Scalable TCP [4], Caltech’s FAST protocol [5].

Today, there have been many TCP versions showing significant performance improvement over the original versions. These versions have been tested on a stand-alone basis or, at most, with the classic TCP Reno version. The performance of recent TCP versions like TCP Westwood and FAST TCP has only been compared with TCP Reno [4]. Research on TCP Westwood has explored its improvement on high-bandwidth networks and its friendliness with just two TCP versions, Reno and Vegas. The Caltech group at UCLA has conducted research on FAST TCP and has compared its results with respect to TCP Reno. Scalable TCP, which is based on Highspeed TCP, is still wide open for exploration [6]. Explicit Control Protocol (XCP) [11] also compares its performance with TCP Reno. In our discussions, we will categorize the protocols under study as ‘old’ and ‘new’ protocols. The ‘new’ protocols for the High Bandwidth-Delay Product networks include the HSTCP and the Scalable TCP; the rest fall under the ‘old’ category. Apart from the research mentioned above, there has not been a performance evaluation of any kind conducted where the old protocols and the new have been compared together
under similar conditions. So far the matter of how a set of new and old protocols would behave separately if run on a link of varying capacity has not been tested. Features like Packet Loss Ratio, slow-start time, sequence number, congestion window, and throughput have not been compared or analyzed. In this thesis, a performance evaluation of these old and new protocols is conducted and the results analyzed.

It is also known that TCP’s throughput is inversely proportional to the round trip time (RTT). Hence, fairness issues come into play. Connections with larger RTT take a longer time to fulfill the available bandwidth over high-speed links; connections with shorter RTT, which share the same segment of link, obtain more bandwidth resources. In addition, the TCP congestion control is dependent on the number of flows. Suppose there are N connections sharing the same link, all the connections will increase the sending rate by one segment every RTT, so the overall increase of all TCP flows is a function of N. As a greater number of flows compete for a fair share of bandwidth, fairness to each flow becomes an important factor. Unfairness is bound to result when more than one flow having different RTTs are competing for the same bottleneck link. These were the problems that existed in the older versions of TCP. It has yet to be seen whether these problems still persist with the new protocols. As mentioned earlier, these protocols have been tested and compared mostly with TCP Reno; they have not been tried against their new high-speed counterparts. So far, no analysis has been done on the friendliness issues of these protocols. An analysis for fairness of these protocols in various combinations like Highspeed TCP — Scalable TCP, FAST TCP — XCP, etc., would result in a very interesting discussion. It is not known how these protocols react with one another over high bandwidth delay product links; this is still an area to be explored. In this thesis, the
friendliness of these protocols when contended with the old and new ones over a common channel will be analyzed.

1.2 Contribution of this Thesis

This thesis makes the following contribution, meant to address the above-mentioned aspects.


2. A study of the fairness of new protocols when sharing the same bottleneck link with its peer protocols as well as with themselves.

3. Analysis of the effect of Droptail and RED queuing techniques on performance and fairness.

1.3 Outline of this Thesis

This thesis is organized as follows:

Chapter 2 offers an elaborate view of the various TCP protocols under consideration. It is categorized into 4 parts: 1). The first part describes the basic operation of the TCP protocol and the phases of a congestion window; 2). The second part offers a detailed description of the eight protocols is given with respect to their congestion control mechanisms and the research conducted in relation to them so far; 3). These protocols are classified into two parts: the older version of TCP and the more recent versions for HBDPN. TCP Tahoe, Reno, Newreno, SACK, VEGAS and Westwood have been
explained under the former, while HSTCP and Scalable TCP have been explained under the latter section; 4). The fourth part explains the two router queue management techniques, Droptail and RED, which are used in the simulations.

Chapter 3 explains the three simulation topologies used in the experiments conducted to evaluate the performance of these protocols when running on a stand alone basis, when running against other protocols, and when running against each other for issues concerning fairness. The router queuing techniques and the parameters used in the simulation scenarios are explained in detail.

Chapter 4 discusses the results of the simulations conducted along with graphs for throughput, congestion window, slow start time, packet loss ratio, recovery time, and throughput ratios. The behavior of these protocols over the network topologies mentioned above is explained in detail. Performance of these protocols when competing with other protocols over a bottleneck link is analyzed. These protocols are also evaluated and analyzed when competing with themselves over a bottleneck link.

Chapter 5 concludes the thesis, pointing out the best protocol for the HBDPN. It also discusses future work and additional experiments that could be conducted in this direction.
CHAPTER 2
LITERATURE REVIEW

Congestion avoidance, slow-start, fast retransmit and fast recovery mechanisms of TCP are very well known and studied in vast detail [7],[10],[11],[13],[15], they are considered the building blocks of regular TCP - Reno, Newreno, Tahoe and SACK [11], [12], [13], [14]. In this discussion, TCP Vegas, Westwood, High-speed TCP and Scalable TCP are also put forward for comparison. In this section, the mechanism of TCP is discussed, and a few terms are explained in brief. The above-mentioned TCP protocols are discussed with respect to their congestion control mechanisms and the research done on them to date. Later in the chapter the performance of the above-mentioned protocols on high bandwidth delay product networks is studied.

2.1 The Transmission Control Protocol (TCP)

In order that effective communication take place between the sender and the receiver, TCP uses error, flow and congestion control algorithms. These include the Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery mechanisms. Tahoe is the oldest and the simplest of all the TCP versions. The rest of this section details the mechanisms on which a TCP Tahoe version performs.

A TCP receiver uses cumulative acknowledgements to specify the sequence number of the next packet the receiver expects. The generation of acknowledgements
allows the sender to get continuous feedback from the receiver. Every time a sender sends a segment, the sender starts a timer and waits for the acknowledgement. If the timer expires before the acknowledgment is received, TCP assumes that the segment is lost and retransmits it. This expiration of the timer is referred to as a timeout. If the acknowledgement is received, however, TCP records the time at which the segment was received and calculates the Round Trip Time (RTT). A weighted moving average of the RTT is maintained and used to calculate the timeout value for each segment.

TCP uses a sliding window mechanism to achieve flow control that allows multiple packets to be present in flight so that the available bandwidth can be used more efficiently. This keeps the sender from overwhelming the receiver’s buffers. However, the most important variation of TCP’s sliding window mechanism over other sliding window mechanisms is the variation of the window size in TCP with respect to time. If the receiver is unable to send acknowledgements at the rate at which the sender is sending data, the sender reduces its sending window. The sender and receiver agree upon the number of packets that a sender can send without being acknowledged, and upon number of packets the receiver is able to receive, before its buffers become overwhelmed. This is accomplished by the Advertised Window (AWND) parameter, which is the receiver side estimate of the number of packets it is able to receive without overflowing its buffer queues.

TCP also includes several variables for performing congestion control. The CWND variable defines the number of consecutive packets that a sender is able to send before receiving an acknowledgement and the variable is changed based on network conditions. At any given point in time the sender is allowed to send as many consecutive
packets as provided by the minimum of CWND and AWND, thereby considering the condition of the receiver and the network simultaneously. At the connection startup time, CWND is started at 1 and incremented by 1 for every acknowledgment received thereafter. This leads to an exponential growth of the transmission rate and is referred to as the Slow Start algorithm. The growth continues until the Slow Start Threshold (SSTHRESH) is reached. After that, the CWND is increased by 1 for every RTT, presenting a linear growth characteristic in the Congestion Avoidance Phase. This is the additive increase mechanism of congestion control in TCP, as the transmission rate additively increases for every successful packet transmission. The Congestion Avoidance phase continues increasing the CWND until a packet is lost, in which case the congestion window is reduced to 1 and TCP enters the Slow Start phase. This is a multiplicative decrease since CWND reduces to a value of 1 as shown in Figure 1. The loss of a packet in the congestion avoidance state leads to a timeout in Tahoe.

TCP includes error control mechanisms to provide a reliable service. TCP detects packet losses by means of the retransmission time out or the reception of 3 duplicate acknowledgements (DUPACKS). Upon the receipt of 3 DUPACKS asking for the retransmission of the same packet, TCP assumes that the segment is lost due to congestion. At this point, TCP retransmits the missing packet instead of waiting for a timeout to occur. This is called the Fast Retransmit algorithm. A TCP Tahoe sender has these three main algorithms available to perform error and congestion control. Because of the drastic reduction of its CWND, TCP Tahoe has been shown to provide very low throughput.
2.2 TCP Versions

The various TCP versions that are to be studied are classified into 2 parts: old versions and new versions [TCP for High Bandwidth Delay Product Network (HBDPN)]. While the older versions include Tahoe, Reno, Newreno, SACK, Vegas and Westwood, the more recent ones, which address the performance issues of TCP over high bandwidth delay product networks, include High-speed TCP, and Scalable TCP. The details of each of these are investigated in the following sections.

2.2.1 Older Versions

The TCP versions falling under this category have been performing very well over not-so-large bandwidth networks over a long period of time. These are the

---

**Figure 1.** Variation of TCP’s Congestion Window with Time.
versions that have faced many challenges when run on very high bandwidth delay
product networks. These categories of protocols are discussed one by one as follows.

TCP Tahoe

Early TCP implementations followed a go-back-n technique using cumulative
positive acknowledgement, and required a retransmit timer expiration to resend data lost
during the flight. These TCPs did very little to handle congestion. TCP Tahoe added a
number of new algorithms and refinements to earlier implementations. The new
algorithms include slow-start, congestion avoidance, and fast-retransmit [15]. One of the
major refinements was the modification of the roundtrip time estimator used to set
retransmission timeout values. Initially, it was assumed that lost packets represented
congestion. Therefore, it was assumed by Jacobson that when a packet loss occurred, the
sender should lower its share of the bandwidth.

The mechanism of TCP Tahoe is the same as explained in section 2.1. TCP Tahoe
does not deal well with multiple packet drops within a single window of data. The two
phases in increasing the congestion window, the slow-start and the congestion avoidance
phases can be summed up with the following equations.

\[ \text{Slow-start phase:} \]
\[ cwnd = cwnd + 1 \quad \text{if } cwnd < ssthresh \]

\[ \text{Congestion avoidance phase:} \]
\[ cwnd = cwnd + 1/cwnd \quad \text{if } cwnd \geq ssthresh \]
where ssthresh is the threshold value at which TCP changes its phase from slow-start to congestion avoidance. When a segment loss is detected, the cwnd and ssthresh are updated as follows.

\[ ssthresh = \frac{cwnd}{2} \]
\[ cwnd = 1 \]

During the time when TCP Tahoe came up, the network environment and the applications that were being used did not demand high bandwidth links. Hence, this variant of TCP did not have to face the challenge of scaling to the high bandwidth delay product network. Studies done in [16] reflect that TCP Tahoe has major drawbacks as a means of providing data services over a multimedia network, since random loss resulting from fluctuations in real-time traffic can lead to significant throughput deterioration in the high bandwidth delay product network. The results of these studies conclude that the performance is degraded when the product of the loss probability and the square of the bandwidth-delay product is large. Also, for the high bandwidth delay product network, TCP is extremely unfair towards connections with higher propagation delays.

TCP Reno

The TCP Reno [19] implementation modified the sender to incorporate a mechanism called fast recovery. Unlike Tahoe, Reno does not empty the pipe unnecessarily on the receipt of a few numbers of dupacks. Instead, with the mechanism of fast recovery the congestion window is set to half its previous value. The idea is that the only way for a loss to be detected via a timeout and not via the receipt of a dupack is when the flow of packets and ACKs has completely stopped, which would be an
indication of heavy congestion. But if the sender is still able to receive an ACK, then it should not fall back into slow-start, as it does in the case of TCP Tahoe. This case does not imply heavy congestion, since the flow still exists, but the sender should send with relatively less vigor, utilizing a lower amount of resources. The mechanism of fast recovery comes into picture at this stage. After receiving a certain number of dupacks, the sender will retransmit the lost packet; but, unlike Tahoe, it will not fall back into slow-start. It will rather take advantage of the fact that the currently existing flow should keep on sending, albeit using fewer resources. By using fast recovery the sender uses a congestion window that is half the size of the congestion window present just before the loss. This factor forces Reno to send less packets out until it knows that it is feasible to send more. Therefore, it has indeed reduced its utilization of the network. Although Reno TCP is better than Tahoe in cases of single packet loss, Reno TCP is not much better than Tahoe when multiple packets are lost within a window of data [15], [17]. Fast recovery ensures that the pipe does not become empty. Therefore, slow-start is executed only when a packet is timed out. This is implemented by setting ssthresh to half the current congestion window size and then setting the congestion window to 1 segment, causing the TCP connection to slow-start until the ssthresh is reached; then it goes into the congestion avoidance phase like in the case of Tahoe. The Reno TCP represented in equation form looks like this.

**Slow-start phase**

\[
cwnd = cwnd + 1
\]

When a segment is detected, the fast retransmission algorithm halves the congestion window.
\[ ssthresh = cwnd/2 \]
\[ cwnd = ssthresh \]

TCP Reno then enters fast recovery phase. In this phase, the window size is increased by one segment when a duplicate acknowledgement is received; and the congestion window is restored to ssthresh when a non-duplicate acknowledgement corresponding to the retransmitted segments is received.

The basic problem in TCP Reno is that fast retransmit assumes that only one segment was lost. This can result in loss of ACK clocking and timeouts if more than one segment is lost. Reno faces several problems when multiple packet losses occur in a window of data. This usually occurs when fast retransmit and fast recovery is invoked. It is invoked several times in succession leading to multiplicative decreases of cwnd and ssthresh impacting the throughput of the connection. Another problem with Reno TCP is ACK starvation. This occurs due to the ambiguity of duplicate ACKs. The sender reduces the congestion window when it enters fast retransmit; it receives dupacks that inflate the congestion window so that it sends new packets until it fills its sending window. It then receives a non-dupack and exits fast recovery. However, due to multiple losses in the past, the ACK will be followed by 3 dupacks signaling that another segment was lost; this way, fast retransmit is entered again after another reducing of ssthresh and cwnd. This happens several times in succession and during this time the left edge of the sending window advances only after each successive fast retransmit; and the amount of data in flight eventually becomes more than the congestion window. When there are no more ACKs to be received, the sender stalls and recovers from this deadlock only through
timeout, which causes slow-start. There are two solutions available for the above problems: Newreno and TCP SACK.

TCP Newreno

The TCP Newreno [11] modifies the fast retransmit and fast recovery mechanisms of Reno TCP. These modifications are implemented to fix the drawbacks of TCP Reno. Here, the wait for the retransmit timer is eliminated when multiple packets are lost from a window. Newreno is the same as Reno but applies more intelligence during fast recovery. It utilizes the idea of partial ACKs. When there are multiple packet losses, the ACK for the retransmitted packet will acknowledge some but not all the packets sent before the fast retransmit. In Newreno, a partial ACK is taken as an indication of another lost packet and as such the sender transmits the first unacknowledged packet. Unlike Reno, partial ACKs do not take Newreno out of fast recovery. This way Newreno retransmits 1 packet per RTT until all lost packets are retransmitted, and avoids requiring multiple fast retransmits from a single window of data. This Newreno modification of Reno TCP defines a fast recovery procedure that begins when three duplicate ACKs are received and ends when either a retransmission timeout occurs or an ACK arrives that acknowledges all of the data up to and including the data that was outstanding when the fast recovery procedure began [18]. The Newreno algorithm can be explained in the following steps:

- On the receipt of the third dupack, if the sender is not already in fast recovery procedure, then set ssthresh to no more than the value below [19].

\[ ssthresh = \max(\text{flightsize}/2, 2\times\text{MSS}) \]
Also, remember the highest sequence number transmitted in a variable.

- Retransmit the lost packet and set cwnd to ssthresh + 3*MSS. This artificially inflates the congestion window by the number of segments that have left the network and that the receiver has buffered.
- For each additional dupack received, increment the congestion window by MSS.
- Transmit a segment, if allowed by the new value of cwnd and the receivers advertised window.
- When an ACK arrives that acknowledges new data, this ACK could be the acknowledgement elicited by the retransmission from step 2, or one elicited by a later retransmission.

TCP Vegas

In 1994, Brakmo, O'Malley and Peterson came with a new TCP implementation called Vegas that achieves between 40% and 70% better throughput and 1/5 to 1/2 the losses when compared with TCP Reno. TCP Vegas [20] also had all the changes and modifications on the sender side. In Reno, the RTT is computed using a coarse-grained timer, which does not give an accurate estimate of RTT. Tests conducted conclude that for losses that resulted in a timeout, it took Reno an average of 1100ms from the time it sent a segment that was lost, until it timed out and resent the segment; whereas less than 300ms would have been the correct timeout interval had a more accurate clock been used. TCP Vegas fixes this problem using a finer coarse-grained timer. Vegas also changed the retransmission mechanism. The system clock is read and saved each time a segment is sent; when an ACK arrives, the clock is read again and the RTT calculation is computed
using this time and the timestamp recorded for the relevant segment. With the use of this accurate RTT, retransmission is decided as follows: When a dupack is received, Vegas checks to see if the new RTT is greater than RTO. If it is, Vegas retransmits the segment without having to wait for the 3rd dupack. Whereas, when a non-dupack is received, if it is the first or second one after a retransmission, Vegas checks again to see if $\text{RTT} > \text{RTO}$; if so, then the segment is retransmitted. This process catches any other segment that may have been lost previous to the retransmission without requiring a waiting period to receive a dupack. Vegas treats the receipt of certain ACKs as a trigger to check if a timeout should happen, but still contain Reno’s timeout code in case this mechanism fails to recognize a lost segment.

Vegas' congestion avoidance actions are based on changes in the estimated amount of extra data in the network. Vegas defines the RTT of a connection as its BaseRTT when the connection is not congested. In practice, it is the minimum of all measured roundtrip times and mostly it is the RTT of the first segment sent by the connection before the router queues increase. Vegas uses this value to calculate the expected throughput. Secondly, it calculates the current actual sending rate. This is done by recording the sending time for a segment, recording how many bytes are transmitted between the time that segment is sent and its acknowledgement is received, computing the RTT for the segment when its acknowledgement arrives, and dividing the number of bytes transmitted by the sample RTT. This calculation is done once per round trip time. Thirdly Vegas compares actual to expected throughput and adjusts the window accordingly. Difference between the actual and expected throughput is recorded. Vegas defines two thresholds, $\alpha$ and $\beta$, which roughly correspond to having too little and too
much extra data in the network, respectively. Following is the mechanism of the congestion control in equation form. \(Diff\) is the difference between actual and expected throughput.

\[
\begin{align*}
    Diff < 0 & : \text{ change BaseRTT to the latest sampled RTT} \\
    Diff < \alpha & : \text{ increase the congestion window linearly} \\
    Diff > \beta & : \text{ decrease the congestion window linearly} \\
    \alpha < Diff < \beta & : \text{ do nothing}
\end{align*}
\]

To be able to detect and avoid congestion during slow-start, Vegas allows exponential growth only every other RTT. In between, the congestion window stays fixed so a valid comparison of the expected and actual rates can be made. When the actual rate falls below the expected rate by the equivalent of one router buffer, Vegas changes from slow-start mode to linear increase/decrease mode.

A couple of problems with TCP Vegas that could have a serious impact on its performance, are the issues of rerouting and stability. Rerouting a path may change the propagation delay of the connection; Vegas uses the connection to adjust the window size and it can affect the throughput considerably. Another issue of TCP Vegas is its stability. Since each TCP connection attempts to keep a few packets in the network when their estimation of the propagation delay is off, this could lead the connection to inadvertently keep many more packets in the network causing a persistent congestion.

Research on TCP Vegas to date consists primarily of analyses of the protocol, improving its congestion avoidance and detection techniques. [21][22]. Most of the studies involving TCP Vegas consist of its performance evaluation with respect TCP Reno. [23][24]. Recent research at Caltech is exploring a new Vegas version, which
Caltech claims is a stabilized version of Vegas [25]. This stabilized version of Vegas is completely source-based and requires no network support. They further suggest that this stabilized Vegas be deployed in an incrementing fashion when a network contains a mix of links (some with active queue management and some without). Also, the performance of TCP Vegas is compared against that of TCP Reno on high performance computation grids [26] by Eric Weidge and Wu-chon Feng at Ohio State University. With the help of real traffic distributions Weidge and Feng show that Vegas performs well over modern high performance links and better than TCP Reno, provided that the TCP Vegas parameters $\alpha$ and $\beta$ are properly selected.

TCP SACK

TCP throughput can be affected considerably by multiple packets lost from a window of data. TCP’s cumulative acknowledgement scheme causes the sender to either wait for a round trip time to find out about a lost packet, or to unnecessarily retransmit segments that have been correctly received. With this type of scheme, multiple dropped segments generally cause TCP to lose its ACK-based clock, which reduces the overall throughput. Selective Acknowledgement (SACK) [27] is a strategy that rectifies this behavior. With selective acknowledgement, the data receiver can inform the sender about all segments that have arrived successfully, so that the sender need retransmit only those segments that have actually been lost. This mechanism uses two TCP options: the first is an enabling option, ‘SACK-permitted’ which can be sent in a SYN segment to indicate that the SACK option can be used once the connection is established; the second is the SACK option itself, which may be sent once permission has been given by SACK-
permitted. In other words, a selective acknowledgement (SACK) mechanism combined with a selective repeat retransmission policy can help to overcome these limitations. The receiving TCP sends back SACK packets to the sender TCP indicating to the sender data that has been received. The sender can then retransmit only the missing segments [28]. The congestion control algorithms present in the standard TCP implementations must be preserved. In particular, to preserve robustness in the presence of packets reordered by the network, recovery is not triggered by a single ACK reporting out-of-order packets at the receiver. Further, during the recovery, the data sender limits the number of segments sent in response to each ACK. Existing implementations limit the data sender to sending one segment during Reno-style fast recovery, or two segments during slow-start. Other aspects of congestion control, such as reducing the congestion window in response to congestion, must similarly be preserved. The use of time-outs as a fallback mechanism for detecting dropped packets is unchanged by the SACK option. Because the data receiver is allowed to discard SACKed data, when a retransmit timeout occurs the data sender must ignore prior SACK information, when determining which data to retransmit.

Studies regarding TCP SACK include issues concerning aggressiveness of the protocol in the presence of congestion in comparison to other TCP implementations. Also, the issues concerning current TCP implementation performance in a congested environment when competing against TCP implementations with SACK have been explored. [29]. TCP SACK has also been used to enhance performance of TCP in satellite environments. In [40], TCP with selective acknowledgement is examined and compared to traditional TCP implementations.
TCP Westwood

TCP Westwood is a scheme [30] employed by the TCP source to estimate the available bandwidth and use the bandwidth estimation to recover faster, thus achieving higher throughput. It is based on two concepts: the end-to-end estimation of the available bandwidth and the way such an estimation is used to set the slow-start threshold and the congestion window. Also, it is important to note that the feedback is merely end-to-end and does not depend on any intermediate nodes at the network level. The TCP Westwood (TCPW) source continuously estimates the packet rate of the connection by properly averaging the rate of returning ACKs. This estimate is used to compute the allowable congestion window and slow-start threshold to be used after a congestion episode is detected — that is after three duplicate acknowledgements or a timeout. Unlike TCP Reno, which simply halves the congestion window after three dupacks, TCPW attempts to make a more intelligent decision. It selects a slow-start threshold and a congestion window that are consistent with the effective connection rate at the time of congestion. These types of techniques for bandwidth estimation have been proposed before, (packet pair [31] and TCP Vegas [32]) but, due to technical reasons they have not been deployed onto the network. The key thing about TCPW is that it probes the network for the actual rate that a connection is achieving during the data transfer, not the available bandwidth before the connection is started. TCPW offers a number of features that are not available in TCP Reno or SACK. The knowledge of the available bandwidth can be used to adjust the rate of a variable rate source. In the TCPW the sender continuously computes the connection Bandwidth Estimate (BWE) that is defined as the share of bottleneck
bandwidth used by the connection. After a packet loss indication, the sender resets the congestion window and the slow-start threshold based on BWE as

\[ cwnd = BWE \times RTT. \]

Another important element of this procedure is the RTT estimation. RTT is required to compute the window that supports the estimated rate BWE. Ideally, RTT should be measured when the bottleneck is empty. In practice, it is set equal to the overall minimum roundtrip delay (RTTmin) measured so far on that connection. In TCPW, congestion window increments during the slow start and congestion avoidance remain the same as in Reno — that is they are exponential and linear, respectively. In case of 3 dupacks, TCPW sets the congestion window and slow-start threshold as follows:

\[
\text{ssthresh} = \frac{(BWE \times RTTmin)}{MSS}
\]

\[
\text{if(cwnd > ssthresh) /*congestion avoidance*/}
\]

\[
\quad cwnd = \text{ssthresh};
\]

\[
\text{endif}
\]

In the case of a packet loss being indicated by timeout expiration, cwnd and ssthresh are set as follows:

\[
cwnd = 1;
\]

\[
\text{ssthresh} = \frac{(BWE \times RTTmin)}{MSS};
\]

\[
\text{if(ssthresh < 2)}
\]

\[
\quad \text{ssthresh} = 2;
\]

\[
\text{endif;}
\]

Recent research on TCPW’s performance over large bandwidth pipes includes modifications of TCPW to TCP Westwood with Bulk Repeat (TCPW BR) [33]. TCPW BR has three sender-side modifications, namely Bulk Repeat, fixed Retransmission timeout, and intelligent window adjustment to help a sender recover from multiple losses in the same congestion window and to keep window size reasonably large when there is
no congestion along the path. TCPW BR also uses a loss differentiation algorithm (LDA), which is based on two schemes: spike and rate gap threshold, which is used to differentiate between losses due to congestion and losses due to error. In losses due to congestion, ‘congestion loss mode’ TCPW BR works in the same way as TCPW. In cases of error losses, ‘error loss mode’ TCPW BR relies on the three sender-side modifications discussed above. This protocol has shown significant performance improvement in heavy loss environments. TCPW has also been modified for its performance over large bandwidth networks. Techniques like Adaptive restart (Astart) [34], paced-Westwood [35], TCP-Westwood with easy-RED [36], and TCP Westwood with rate estimates [37] explore the fairness issues, efficiency, friendliness issues, and performance issues of TCPW over high bandwidth-delay product networks that have small buffers.

2.2.2 TCP for High Bandwidth-Delay Product Networks

As the next generation of applications will require network links with steady transfer rates in the order of gigabits per second to transfer huge data in a reliable amount of time, the widely used TCP protocol will become the bottleneck. Since the window-based mechanism of current TCP implementations is not suitable for achieving high link utilization, many researchers have proposed modifications to TCP to improve the performance that it presents in very high bandwidth-delay product links. Here we discuss two protocols proposed to be efficient over high bandwidth links, namely High speed TCP and Scalable TCP.
Highspeed TCP

High-speed TCP [3] is a modification to TCP’s congestion control mechanism to be used with TCP connections that have large congestion windows. The congestion control mechanism of the current standard TCP constrains the congestion windows that can be achieved in realistic environments. For example, for a standard TCP connection with a 1500 byte packets and a 100msec round trip time, achieving a steady state throughput of 10Gbps would require an average congestion window of 83,333 segments and a packet drop rate of, at most one congestion event every 5000,000,000 packets, which is a very unrealistic constraint. High-speed TCP is designed to have a different response in environments with a very low congestion event rate. It is also designed to have the standard TCP response in environments with packet loss rates of, at most, $10^{-3}$. Since HSTCP leaves TCP’s behavior unchanged in environments with mild to heavy congestion, it does not increase the risk of congestion collapse. In environments with very low packet loss rates HSTCP presents a more aggressive response function. The high-speed TCP response function is specified using three parameters: low_window, high_window, and high_p; low_window is used to establish a point of transition. The HSTCP response function uses the same response function as regular TCP, when the current congestion window is at most low_window; it uses the high-speed TCP response function when the current congestion window is greater than low_window; high_window and high_p are used to specify the upper end of the high-speed TCP response function [36]. The high-speed TCP response function is represented by new additive increase and multiplicative decrease parameters. In congestion avoidance phase the behavior of the congestion window can be given by the following equations:
\[ \text{ACK: } \text{cwnd} = \text{cwnd} + a(\text{cwnd})/\text{cwnd} \]

\[ \text{Drop: } \text{cwnd} = \text{cwnd} - b(\text{cwnd}) \times \text{cwnd} \]

Research related to HSTCP includes work done by Evandro De Souza [36], Deb Agarwal, and Sally Floyd [37]. In [36], HSTCP is tested for deployment issues. According to De Souza, HSTCP is appropriate to bulk transfer application because it is able to maintain high throughput in different network conditions, and because it is easy to deploy when compared with other solutions already in use. Another study conducted by Sally Floyd in collaboration with Evandro de Souza and Deb Agarwal, which is a part of HSTCP proposal, involves the limited slow-start mechanism for TCP with large windows. [37]

Scalable TCP

Scalable TCP [38] is a simple change to the traditional TCP congestion control algorithm; it claims to improve TCP performance in high-speed wide area networks. Scalable TCP changes the algorithm to update TCP’s congestion window as follows. For each acknowledgement received in a round trip time during which congestion has not been detected:

\[ \text{cwnd} = \text{cwnd} + 0.01 \]

And on the first detection of congestion in a given round trip time:

\[ \text{cwnd} = \text{cwnd} - [0.125 \times \text{cwnd}] \]

Figure 2 shows the main difference in the scaling properties of traditional and scalable TCP. Traditional TCP probing times are proportional to the sending rate and the round
trip time. Scalable TCP probing times are proportional only to round trip time making the scheme scalable to high-speed networks.

Scalable TCP has been designed from a strong theoretical base to ensure resource sharing and stability while maintaining agility in conjunction with prevailing network conditions. The response curves for both a traditional TCP connection and a scalable TCP connection is shown below. The scalable TCP algorithm is only used for windows above a certain size. By choosing the point at which the response curves intersect, good resource sharing with traditional connections can be ensured.

**Figure 2.** Scalable TCP Congestion Window Properties [4].

This allows scalable TCP to be deployed incrementally. Scalable TCP builds directly on the high-speed TCP proposal and works on engineering stable and scalable TCP variants.

2.3 Router Queuing Techniques

2.3.1 Droptail
The drop tail [42] scheme is the traditional and the simplest technique for managing router queue lengths. Droptail does not selectively drop packets; it drops them when there is no buffer space available. After setting a maximum length to the queue, Droptail accepts packets for the queue until the maximum length is reached, and then drops subsequent incoming packets until the queue decreases (as a packet from the queue has been transmitted). This technique is also known as "tail drop", since most recently arrived packet, which is at the tail of the queue, is dropped when the queue is full.

Connections sending more traffic will get more system resources (although not necessarily better performance).

This method has served the Internet well for years, but it has two important drawbacks. In some situations tail drop allows a single connection or a few flows to monopolize queue space, preventing other connections from finding room in the queue. This "lock-out" phenomenon is often the result of synchronization or other timing effects.

Figure 3. Response Curves of Scalable TCP and Regular TCP [4].
The tail drop discipline allows queues to maintain a full status for long periods of time, since tail drop signals congestion only when the queue has become full. It is important to reduce the steady-state queue size, and this is perhaps queue management's most important goal.

2.3.2 Random Early Detection (RED)

There are two basic parts to RED [43]: detecting congestion, and responding to congestion. The algorithms for both of these tasks are simple, efficient in terms of both time and space, and easy to implement. To track the congestion level, an Exponentially Weighted Moving Average (EWMA) of the queue length is kept. RED recalculates the average queue length $avg$ each time a packet arrives in order to have an up-to-date estimate of the current congestion level when determining what to do with the incoming packet. If the queue is not empty, then the $avg$ is calculated using the following equation:

$$avg = (1-w_q)avg + w_qq$$

where $q$ is the instantaneous queue length given by the number of packets currently enqueued, and $w_q$ is an operator-set parameter called the queue weight, which determines how quickly $avg$ can change. If the queue is empty, then the equation used to update $avg$ depends upon the amount of time the queue was idle before the packet arrived, $q_{time}$, and the number of small packets that could have been transmitted by the gateway during that time. Thus,

$$m = (time - q_{time})/s$$

$$avg = (1 - w_q)^mavg$$
where \( time \) is the current time, and \( m \) is simply a temporary value representing the number of packets that could have been transmitted during the idle time \( (time - q_{time}) \), and \( s \) is the time needed to transfer a typical small packet. Once an estimation of the congestion level has been calculated, the gateway uses the value to determine what to do with the incoming packet. For this purpose RED queues are configured with two values, \( min_{th} \) and \( max_{th} \), which represent minimum and maximum thresholds for calculating a random drop probability. At \( avg \) values below \( min_{th} \), the incoming packet will simply be enqueued, while at values above \( max_{th} \) it will be “marked”. This marking can consist either of dropping the packet or performing some action such as setting a bit in the packet’s encapsulated transport header to indicate a congestion event to the flow. If \( avg \) falls between \( min_{th} \) and \( max_{th} \) however, the gateway randomly marks the packet with a probability \( p \), which it generates internally. Further RED experiments [44] have led Floyd to recommend using what is known as the “gentle” RED variation, in which the only difference from standard RED is the probability that a packet will be dropped that varies from \( max_{p} \) (maximum drop probability) to 1, when \( avg \) is between \( max_{th} \) and \( 2 \times max_{th} \).

![Diagram of Drop Probabilities]

(a) (b)

Figure 4. Drop Probabilities of Different RED Modes. (a) Normal Drop Probability  (b) Gentle Drop Probability.
The goal of this algorithmic adjustment is to allow more leeway away from optimal values in the network operator’s selection of $max_p$ and $max_{th}$, without severely influencing performance. The differences between normal and gentle modes are illustrated by Figures 4(a) and 4(b), respectively.
CHAPTER 3
METHODOLOGY

This chapter explains in detail, the simulation scenarios used in the performance evaluation of these eight protocols. This chapter is divided into two sections. The first section describes the simulation setup and the parameters used for evaluating the performance of the TCP protocols on a stand-alone basis over a single link. The second section describes the simulation setup for evaluating the fairness of these protocols to themselves as well as to other variants. The simulation topology for evaluating these two types of fairness is the same. All the simulations are conducted using both DropTail and RED (Gentle) router queuing techniques. The packet size in all the simulations is taken as 1000 bytes. Also, the application, which is used to send data, is FTP in all scenarios. All the simulations use the NS-2 [39] simulator. The parameters for RED queuing technique are set to default in the tcl script so that NS-2 configures them automatically.

3.1 Simulation Topology for a HBDP Link (Single source/single sink)

This section includes the description of the simulation topology and parameters used to evaluate the performance of TCP Tahoe, Reno, Newreno, SACK, Vegas, Westwood, HSTCP and Scalable TCP over high bandwidth-delay product networks. The network topology used consists of one TCP source, one TCP sink node (destination), and
two routers connected by a bottleneck link as shown in Figure 5. The simulations were carried out using network simulator NS-2 [39]. The maximum values of the congestion window for the TCP versions are set such that the connections could achieve full link utilization. The propagation delay of the bottleneck link is fixed and set to 25msec (two-way).

The bandwidth of the link is varied from 1.5Mbps to 1000Mbps with values of 10, 100, 250, 550 and 800Mbps in between. In this way the bandwidth-delay product increases; this increase is used to show the performance of these protocols. The queue limit of the bottleneck link is set to 200 packets to absorb part of the sudden congestion. The experiments performed are used to show the bandwidth utilization, congestion window behavior, packet loss rate during the slow-start phase, and recovery time after a packet drop event.

Figure 5. Simulation Topology for One Source – One Sink.
3.2 Simulation Topology for Fairness to Itself and Others Scenario (Two Sources/Two Sinks)

The simulation topology used is a dumbbell with a single bottleneck link as shown in Figure 6. The bottleneck link is connected on either sides by two TCP sources and two TCP sink nodes. The queue limit is kept at 200 packets for both scenarios; fairness to itself and fairness to other protocols. The link bandwidth is varied from 10Mbps to 1000Mbps with values of 100, 250, 550 and 800 Mbps in between. The link delay is 25msec both ways and is kept fixed. The maximum window size is kept large enough so as not to impose any limits.

In the case of experiments regarding ‘fairness to itself’ of a TCP protocol, the propagation delay of one TCP source is varied and its value is kept as the multiples of the propagation delay of the other TCP source. For example, if S1’s delay is 10msec, then S2’s delay is taken as 20msec through 60msec with multiples of 10 for the various
topology to observe how they perform when competing with themselves.

The topology for the performance of a TCP protocol, when competing with another TCP protocol, is shown in Figure 7. The only difference in these simulations is that S1 and S2 are two different TCP sources and the propagation delays of both sources are the same (in our case 10msec). All eight protocols are tested against one another under this scenario and their performance is evaluated with respect to the throughput ratios they achieve.
CHAPTER 4
PERFORMANCE EVALUATION

As mentioned in the previous chapter, the simulations explained here are categorized into three sections. First, each protocol is evaluated separately on a single source single sink topology with a varying bandwidth. Second, each protocol is evaluated against every other protocol for fairness issues. And third, each protocol is evaluated for issues concerning fairness when sharing a bottleneck link with itself. Simulations in each section have been performed using Droptail as well as RED (gentle) router queuing techniques.

4.1 Single Source - Single Sink Topology

In this scenario, while evaluating the results, it is important to have the knowledge that Droptail and RED queuing techniques will virtually have the same effect. RED in this case with a single source (connection) will mark all the packets for dropping when the average value of the queue goes above the maximum threshold. Since it is a single source scenario, the whole queue (buffer) is for this connection, and hence RED will drop packets as the buffer is full, which is similar to the mechanism Droptail follows. The effective buffer in this case will be less than 200 packets, which is also used in Droptail. This buffer value is automatically configured in the NS-2’s [39] RED mechanism. Hence
in the following section, we see that the results of droptail and RED queuing techniques are similar.

Figure 8 shows the utilization achieved by the protocols under consideration as a function of the bandwidth of the bottleneck link. As the figure shows, there are considerable differences among these protocols. As a general trend, it can be seen that the performance of most protocols degrades as the bandwidth increases, revealing clear scalability problems. This is expected behavior, and it confirms what other researchers have found [41]. Only TCP Vegas, Scalable TCP, and HighSpeed TCP seem to perform well and scale better to higher speeds. The importance of this graph is the addition of other TCP versions as well as Vegas and Westwood, which had not been compared previously. The TCP versions also perform as expected with Tahoe presenting the worst performance followed by Reno, Newreno, and SACK, in that order. This sequence reflects the behavior of these protocols according to their reaction to packet losses and multiple packet losses from the same congestion window. Finally, TCP Westwood improves over the regular TCP versions but still below HighSpeed TCP, Scalable TCP and Vegas.

(a)
Figure 8. Normalized Throughput of Protocols Under Consideration. (a) Using Droptail (b) Using RED.

Figure 9. TCP Sequence Numbers for Link Bandwidth 1Gbps. (a) Using Droptail (b) Using RED
Figure 9 also shows the throughput achieved by all these protocols while they are using the TCP sequence numbers in the case of a 1 Gbps link. As it can be seen from the figures, the results relate very well to each other. The throughput performance of the protocols can be explained by looking at the behavior of the congestion window variable. Figure 10 plots the $cwnd$ of the protocols over time, where the bottleneck bandwidth is set to 1 Gbps. With the exception of Vegas, the figure shows the expected sawtooth pattern of TCP. It can be seen that TCP Tahoe is the only protocol reducing its $cwnd$ to 1, while the other TCP versions only reduce it to half or less than half of the current value. Reno presents deeper and longer reactions, while Newreno and SACK are very similar. Interesting behaviors are experienced by TCP Vegas, Westwood and HighSpeed TCP. TCP Westwood achieves better throughput because its $cwnd$ does not drop as deep as the regular TCP versions, guided by the Fair Share Estimate (FSE). It will be seen later that TCP Westwood goes through a rather long Congestion Avoidance phase. The case is the same.

![Congestion Windows of All the Protocols when Link Speed 1Gbps. (a) Using Droptail (b) Using RED](Continued.)
Figure 10. Congestion Windows of All the Protocols when Link Speed 1Gbps. (a) Using Droptail (b) Using RED
with the congestion window of Scalable TCP. Its window behavior is similar to that of TCP Westwood, but Scalable TCP achieves higher throughput because it does not take the initial computation time that TCP Westwood takes to compute the estimated bandwidth. The \( cwnd \) of HighSpeed TCP takes values similar to TCP Westwood, but it achieves better throughput since it manages to transmit more packets, particularly at the beginning of the connection. It also achieves better throughput because it increases the congestion window faster. HighSpeed TCP presents the oscillatory behavior also experienced in the simulation results in [36], when only one source is in the system. TCP Vegas’s behavior is the best as the \( cwnd \) is rather steady after the Slow Start.

Two conclusions are important at this point. First, it is definitively impossible to achieve full bandwidth utilization using the window-based approach utilized by current TCP implementations. The behavior of the \( cwnd \) shows that it takes TCP too much time to reach the maximum window size, and too little time to reduce its size in the presence of packet losses. Furthermore, the reduction of the \( cwnd \) is very drastic. The second conclusion is more important and has to do with Vegas’ behavior. If full utilization is to be achieved, the mechanism used by Vegas needs to be improved upon further. In the case of the bottleneck bandwidth being set to 1 Gbps, for example, it is known that the theoretical value of the congestion window required to achieve full link utilization is approximately 3325 packets, given by the bandwidth-delay product of the network and the buffer size. It can be observed from Figure 11 that this is the maximum value achieved by all protocols, and that TCP Vegas’ congestion window is very steady and close to 3325 after the Slow Start phase, indicating that TCP Vegas is very good at estimating the available bandwidth. The main problem with Vegas lies in the first
Congestion Avoidance phase; it takes Vegas a rather long amount of time to initially reach the 3325 value.

Next, the performance of the protocols were evaluated during the Slow Start phase. Here, the Slow Start time is a primary area of interest, as is the Packet Loss Rate (PLR) during that period of time. The Slow Start time is important because the longer it takes, the more wasted capacity results. The PLR is an indication of how efficient the Slow Start mechanism is. Obviously, the higher the PLR, the worse. The PLR was measured as the number of packets lost, divided by the total number of packets sent during the Slow Start phase. From Figure 11 it can be seen that all protocols have a similar and very short Slow Start duration. This is expected since they all employ the same exponential mechanism. Vegas has a slightly longer duration because it increases the cwnd exponentially every other RTT; however, its slow-start time of 0.4 seconds is still a very short time. Figure 12 shows the PLR achieved by the different protocols during the Slow Start phase. As can be seen, most TCP versions show a similar and steady PLR as the bandwidth is increased. This is expected because the buffer at the bottleneck link fills out at the same time regardless of the
link capacity. This is in contradiction to other studies that say that one of the problems of current TCP versions is the very large value of $cwnd$ that is achieved during Slow Start and resulting the high PLR. The explanation is in the buffer size of the bottleneck link. If the buffer size is set to the bandwidth-delay product of the link, the $cwnd$ will in fact grow to very large values (in the order of 10000 in this study’s 1 Gbps case). In reality, however, the system can only absorb around 6250 packets; but if the buffer size is set to more realistic values, as in this study’s example, the $cwnd$ will grow to modest values and the number of packets dropped will not be substantial. For instance, the PLR was in the order of 6% for this study.

An interesting point to mention here is the fact that TCP Vegas and HighSpeed TCP were the only protocols with zero PLR. While TCP Vegas’ Slow Start phase takes a little longer than the other protocols, its Slow Start procedure is rather effective in avoiding packet losses during this time. This is in complete alignment with the design goals of Vegas as explained in [20]. The performance of the protocols over the
Congestion Avoidance phase is also investigated in the current study. Here, interest is
mostly in the Congestion Avoidance phase time, called the recovery time, or the time that
it normally takes the congestion window to reach its maximum value after a drastic reduction
resulting from a packet drop. Figure 13 shows this time in seconds for the different

Figure 12. Packet Loss Rate. (a) Using Droptail (b) Using RED
protocols as the bandwidth of the channel is increased. As was expected, the recovery time takes longer as the channel capacity increases. From the graph, it can be seen that the recovery time for regular TCP versions is around 70 seconds, or 2800 RTTs, while the recovery time of TCP Westwood and Vegas is around 10 seconds longer. Also, it can be observed that the recovery time of HighSpeed TCP is very small. This is due to the oscillatory behavior presented by this protocol as observed in Figure 10. For this experiment, only

![Graph](image)

**Figure 13.** Recovery Time (a) Using Droptail (b) Using RED
the first Congestion Avoidance phase was utilized. Another interesting point is related to TCP Westwood. It was found that the bandwidth calculation during the initial phase is not very accurate, and therefore, after the initial loss of packets, TCP Westwood sets the $cwnd$ and $ssthresh$ at very low values. Figure 17 shows the values of $cwnd$ and $ssthresh$ over time in the case where the bottleneck link is set to 1 Gbps. The figure clearly shows the bandwidth estimation problems of Westwood experiences during the initial phase of the connection and how the Congestion Avoidance phase starts with very small $cwnd$ and $ssthresh$ values. As a result, Westwood stays in that phase for a very long time, wasting a lot of bandwidth. In fact, the $cwnd$ was set equal to 41 and grew to 3325 in a linear manner. A similar case was found in Vegas where the first Congestion Avoidance phase started at a $cwnd$ of 72. At the beginning of the Slow Start phase, the expected bandwidth was a high value because the network is empty. However, the actual bandwidth decreased substantially, since the exponential increase of the $cwnd$ quickly filled the buffers. At this point, Vegas lost some packets, reduced its $cwnd$, and then entered into Congestion Avoidance with a very low value of $cwnd$. Under realistic network conditions with normal buffer sizes this situation is rather unavoidable. Since the expected bandwidth is very close to the link speed, the $cwnd$ starts increasing linearly until the actual bandwidth equals the expected bandwidth, and, at that time, the $cwnd$ remains steady until the end of the simulation, achieving full utilization. The problem is that this initial Congestion Avoidance phase is very long and increases with the bandwidth of the bottleneck link. Modifying the slow-start phase procedures or the algorithms that drive the Congestion Avoidance phase can solve this problem.
4.2 Two Sources – Two Sinks (*Fairness to Others*)

For the current study, the four recent and very important TCP modules are analyzed separately: Highspeed TCP, TCP Westwood, TCP Vegas, and Scalable TCP. Currently these TCP versions seem to have the potential to make their way through the high bandwidth delay product barrier. The following sections analyze the behavior of Highspeed TCP, TCP Westwood, TCP Vegas and Scalable TCP when they are competing with one another and with regular TCP protocols. Finally, the regular TCP protocols are discussed together. For the simulations, a dumbbell topology is employed as shown in Figure 7. Two different TCP versions are attached to two sources to evaluate their performance over the bottleneck link. The analysis is done by taking the throughput ratio of these protocols. All the graphs plot the throughput ratio against the bandwidth for these protocols. For example, if the two protocols under study are HSTCP and TCP Reno, then their throughputs are calculated over the varying bandwidth and their ratios are taken and represented as “High/Reno”.

Highspeed TCP (HSTCP)

In Figure 14, as was seen in the previous section, HSTCP is shown to be very aggressive grabbing a high percentage of bandwidth share when competing with other TCP protocols.

HSTCP increases its congestion window quickly in the slow-start phase because it transmits more packets at the start of the connection. This behavior of HSTCP makes it grab more bandwidth; hence, it achieves on average, 25-50 times higher throughput than other protocols. The fairness tends to improve as the bandwidth continues to increase.
This is due to the fact that as the bandwidth of the link increases, the buffer size of 200 tends to become small for this bottleneck link; hence the congestion status changes to heavy from moderate. The packet loss ratio increases which causes the HSTCP to follow the highspeed response function with fairness being improved according to [41]. Figure 15 shows the throughput ratios of HSTCP when competing with TCP Westwood and TCP Vegas. These two protocols suffer drastically with respect to HSTCP, as a result of their congestion control mechanisms and bandwidth estimation process, which will be considered in more detail in the following section.
Figure 14. Performance of HSTCP Against TCP Protocols (a) Using Droptail (b) RED

These protocols always tend to be dominated by other protocols and never perform well in competent environments although as seen in the previous section, these protocols achieve a very high throughput when run on their own.
The effect of the router queuing techniques was minimal and existed only when the link capacity was small. The relative fairness between HSTCP and other protocols is less in the case of Droptail, than in the case of RED. This is because with RED the fraction of packet drops received by each flow should be roughly proportional to that flow’s share of link bandwidth, while this property is no longer true in case of Droptail queue management. As the link capacity continues to increase, the router queue management technique does not have effect on the results.

In today’s Internet scenario, there are not a lot of TCP connections that are operating effectively with congestion windows containing thousands of packets. Therefore, the benefits of the HSTCP would outweigh the unfairness that would be experienced by regular TCP protocols.
TCP Westwood (TCPW)

The TCPW uses a bandwidth estimation mechanism with which it manipulates its ssthresh and congestion window after every RTT. Unlike other TCP protocols, TCPW does not reduce the ssthresh to half during a congestion event but converges it to a steady value as shown in Figure 17. This takes a fair amount of time and during this time, another TCP version with which TCPW is competing (as it was in this study) grabs more than its fair share of bandwidth.

As Figure 16 shows, all the protocols dominate over TCP Westwood; and, as the bandwidth increases, this dominance tends to increase. The router queuing techniques do not have a significant effect on the issue of fairness.
Figure 16. Performance of TCP Westwood

Figure 17. Behavior of TCP Westwood’s Cwnd and Ssthresh when the Bottleneck Link is Set to 1 Gbps.

TCP Vegas

According to Figure 11, TCP Vegas has a slightly longer slowstart time than do other protocols. The reason for this is that Vegas increases its congestion window exponentially every other RTT. The performance of Vegas can be seen in Figure 18. The
plots suggest that TCP Vegas does suffer when it shares a bottleneck link with other TCP protocols, due to its bandwidth estimation technique, which prolongs the slowstart phase. This is because the other protocols use most of the buffer space, causing the TCP Vegas connection to back off, since it interprets this as a signal of network congestion.

Figure 18. Performance of Vegas (a) Using Droptail (b) Using RED.
In Figure 18, the prominent difference between the two graphs during the small bandwidth phase can be seen. The reason for this is based on the assumption that the buffer occupancy decreases as the bandwidth increases. This study analyzed the Droptail and RED cases. When Droptail is used, Vegas due to its less aggressive mechanism of increasing its congestion window, can not occupy much buffer space; thus, the other protocols sharing the bottleneck link with Vegas will occupy more buffer space and a larger share of the bottleneck link. As the bandwidth increases, however, the buffer tends to become less occupied; hence, Vegas receives a much better share of the buffer. In addition to the buffer share, the bandwidth estimation technique of Vegas outweighs the effect of regular AIMD techniques used in traditional TCP. In the case of RED, Vegas receives a better share of the bottleneck link when compared to Droptail. This is because of the mechanism of RED. RED gives fair share of the buffer to each flow. Hence, Vegas, during the small bandwidth links receives its share of buffer space; and more packets from the other TCP protocols that are sharing the bottleneck link with Vegas are dropped in this stage, increasing fairness. As the bandwidth increases, the buffer tends to become empty and Vegas always receives its share of the buffer (if required), and so does not end up suffering.

Scalable TCP

Scalable TCP’s performance when competing with the rest of the protocols is shown in the Figure 19. The graphs clearly indicate that except for the Scalable/Vegas combination, all other protocols receive a fair share of bandwidth when run with Scalable TCP, irrespective of the router queuing technique used; the Scalable/Westwood
combination is not represented in the graph because of its high magnitude. Scalable TCP uses up most of the bandwidth when sharing a bottleneck link with Westwood, due to TCP Westwood’s congestion window mechanism (see Figure 17). The difference in the (a) and (b) graphs of Figure 19 is the Scalable/Vegas combination. Again, here, the effect of router queuing management is seen only on the low-bandwidth links, and RED increases the fairness towards Vegas when the channel capacity is low. The reason for this was explained in the previous section. The Scalable TCP, being more aggressive, occupies much of the bottleneck link in the smaller bandwidth links (40 times more than Vegas). As the link speed increases, as seen in the previous section, Vegas receives a fair share of the bottleneck link. Here, in this case, because of the aggressive nature of Scalable TCP, it dominates over Vegas fractionally more than the other TCP protocols.
Other TCP protocols [TCP Tahoe, Reno, Newreno and SACK]

These protocols do not show any drastic differences in fairness when competing with one another. Their behavior is shown in Figure 20: all protocols act fair to one another without any noticeable aggressiveness. The reason for this being all these protocols use the same traditional window-based algorithm.
4.3 Two Sources – Two Sinks (*Fairness to Itself*)

This section explains the results obtained from the simulations conducted with the topology shown in Figure 6. These simulations concern the issues regarding fairness shown when each of these protocols shares a bottleneck link with itself. The two sources S1 and S2 in Figure 6 are two TCP sources of the same version. While the propagation delay of one source is kept constant at 10ms, the propagation delay of the second source varies in multiples of 10 - 60ms. The graphs show the RTT scale plotted on the x-axis and Throughput plotted on the y-axis. The throughputs of the S1 and S2 flows are plotted on the graphs for the link capacities of 10, 100, 550 and 1000Mbps. TCP Tahoe, Reno, Newreno, SACK, and Scalable TCP are analyzed.
together because of the similar results they present. Analysis of HSTCP, TCP Westwood and TCP Vegas is done separately.

Analysis of Tahoe, Reno, Newreno, SACK and Scalable TCP

These protocols present some common behavior when they share a common bottleneck link, where one source has a larger propagation delay than the other. The first important and basic thing that can be observed from the graphs in Figure 21 is that, as the propagation delay of one source increases, that source tends to occupy less bandwidth. The reason for this is that, as the propagation delay of a TCP sender increases, it takes longer for it to receive its acknowledgment and increase its congestion window by sending more packets into the bottleneck link. Hence, it fills up a lesser percentage of the shared link than do TCP senders with lesser propagation delay.

As the bandwidth of the bottleneck link increases, the Droptail and RED queuing techniques produce similar results because, at the low bandwidth links, the buffer is utilized the most and, as the bandwidth increases, the buffer tends to be less occupied. Hence, during the Droptail scenario for all the protocols, when the bandwidth of the bottleneck link is 10Mbps, a prominent fairness issue exists, which is absent in the RED technique. This is because in the case of Droptail, the TCP source with the shorter propagation delay fills up the buffer faster than the connection with the longer propagation delay, which means packets are dropped of the latter connection and less of the link is utilized by this source. Another observation is that, as the bandwidth of the link goes on increasing, the total link utilization of the bottleneck link decreases.
Figure 21. Fairness to Itself (Newreno, Reno, SACK, Scalable, Tahoe)

Analysis of TCP Vegas

TCP Vegas shows a comparatively higher total link utilization than the previous protocols, but the fairness factor suffers. The TCP Vegas that has a longer propagation delay is severely affected in this scenario as is shown by the growing gap in Figure 22. This wider gap can be explained via the TCP Vegas congestion window mechanism, which increases its congestion window every other RTT. As the propagation delay of one Vegas source increases this source will take effectively double the time to increase its congestion window than other TCP protocols, which increases congestion window every RTT would take. Hence the broader gap.
Figure 22. Fairness to Itself (Vegas)
CHAPTER 5
CONCLUSIONS

This chapter is divided based on the experiments that were described in the previous chapter. The results and the observations of each experiment are summarized and the protocol is pointed, with advice as to which is best suited for the Internet, which is a large-scale combination of the experiments performed.

From the experiments conducted over a single source single sink topology, TCP Vegas, Scalable TCP, HSTCP and TCP Westwood performed fairly well over the TCP Reno, Newreno, Tahoe and SACK, as the bandwidth of the bottleneck link increased. This experiment offers insight into the performance of these protocols on an individual basis, although the environment in which these protocols were simulated was only remotely similar to the real world environment. A conclusion, which can be made from this set of experiments, is that, as the bandwidth of the link increases, the throughput decreases irrespective of the router queuing technique used.

From the experiments conducted for “fairness to others” it can be summarized that the traditional TCP protocols — Reno, Tahoe, SACK, Newreno are the fairest to one another. HSTCP is the most aggressive protocol and it is not recommended that it be incorporated into the Internet. TCP Vegas, although it is the best when run alone, suffers when it shares a bottleneck link with others, as does Westwood. Scalable TCP, on the
other hand, is a protocol that when sharing a bottleneck link with other protocols is comparatively less aggressive than HSTCP, and slightly more aggressive than regular TCP protocols.

Additionally, when these protocols share a bottleneck link with themselves, Scalable TCP is not very aggressive, and doesn’t allow one source to grab more bandwidth, like Vegas and HSTCP do. Scalable TCP acts similar to traditional TCP protocols in this scenario. Considering the observations of the three scenarios mentioned above, we come to a conclusion that Scalable TCP, among the protocols under study, is the one with a better performance, with regards to throughput and overall fairness.
REFERENCES


