3-30-2004

Performance Evaluation of TCP over Optical Channels and Heterogeneous Networks

Jianxuan Xu
University of South Florida

Performance Evaluation of TCP over Optical Channels and Heterogeneous Networks

by

Jianxuan Xu

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

Major Professor: Miguel A. Labrador, Ph.D.
Wei Qian, Ph.D.
Rafael Perez, Ph.D.

Date of Approval:
March 30, 2004

Keywords: congestion control, High bandwidth-delay product network

TABLE OF CONTENTS

LIST OF FIGURES

PERFORMANCE EVALUATION OF TCP OVER OPTICAL CHANNELS AND HETEROGENEOUS NETWORKS

Jianxuan Xu

ABSTRACT

Next generation optical networks will soon provide users the capability to request and obtain end-to-end all optical 10 Gbps channels on demand. Individual users will use these channels to exchange large amounts of data and support applications for scientific collaborative work. These new applications, which expect steady transfer rates in the order of Gbps, will very likely use either TCP or a new transport layer protocol as the end-to-end communication protocol.

This thesis investigates the performance of TCP and newer TCP versions over High Bandwidth Delay Product Channels (HBDPC), such as the on demand optical channels described above. In addition, it investigates the performance of these new TCP versions over wireless networks and according to old issues such as fairness. This is particularly important to make adoption decisions. Using simulations, it is shown that 1) the window-based mechanism of current TCP implementations is not suitable to achieve high link utilization and 2) congestion control mechanisms, such as the one utilized by TCP Vegas and Westwood are more appropriate and provide better performance. Modifications to TCP Vegas and Scalable TCP are introduced to improve the performance of these versions over HBDPC. In addition, simulation results show that new TCP proposals for HBDPC, although they perform better than current TCP versions, still perform worse than TCP Vegas. Also, it was found that even though these newer versions improve TCP's performance over their original counterparts in HBDPC, they still have performance problems in wireless networks and present worse fairness problems than their old counterparts. The main conclusion of this thesis is that all these versions are still based on TCP's AIMD strategy or similar and therefore continue to be fairly blind in the way they increase and decrease their transmission rates. TCP will not be able to utilize

the foreseen optical infrastructure adequately and support future applications if not redesigned to scale.

CHAPTER 1

INTRODUCTION

1.1 Introduction and motivations

Next generation optical networks are expected to offer a Dynamic Bandwidth on Demand (DBoD) service that customers will use to establish connections over all optical links with very high bandwidth, very low bit error rates and rather long propagation delays. This can be the case of one user connected to a gigabit Ethernet or 10 Gbps Ethernet switch which at the same time is connected to the optical network by means of a Multiservice Provisioning Platform or MSPP device using the GMPLS family of protocols. In this scenario, end users can establish an obtain an end-to-end all optical channel at OC-48 or OC-192 rates on demand to satisfy their communication needs. This DBoD service will support foreseen applications allowing the transfer of huge files or the real-time exchange of very large amounts of data required to do scientific collaborative work.

Several applications already envisioned will need this type of service and infrastructure. In the First International Workshop on Protocols for Fast Long-Distance Networks [1] held early 2003, several presentations made the case about foreseen requirements for end-to-end steady transfer rates in the order of several gigabits per second to support collaborative work and the transfer of huge amounts of data generated by high energy physic projects such as CERN's Large Hadron Collider. As stated in [2], although it is expected that communication networks, storage technologies and powerful computers will support these transfer rates, communication protocols will become the bottleneck if we don't redesign them to scale. This is the case of TCP when running over all optical next generation networks, or similarly, over high bandwidth-delay product channels.

Ideally, we should be able to modify current protocols and provide a smooth transition to the new environment. This is in fact the approach taken by several researchers who have proposed modifications to the most widely used transport layer protocol, TCP. However, these proposals still face important challenges, and more research is needed to find an appropriate solution. In [3], Sally

Floyd explains TCP's three main challenges. First, in order for TCP to achieve transfer rates in the order of gigabits per second, links should have bit error rates considerably smaller than current possibilities. Furthermore, even if these BER were achievable, TCP congestion control mechanism is expected to present problems since congestion signals will have very large interarrival times. The second problem is related to the Slow Start mechanism, which increases TCP's congestion window exponentially. During Slow Start, a TCP connection over a high bandwidth-delay product channel will increase its congestion window to a very large value and once the connection is about to fill the channel's available bandwidth many packets will be dropped. Finally, TCP has been shown to waste too much bandwidth because of its Congestion Avoidance mechanism. Considering the very high link capacities and long propagation delays of DBoD channels, it will take TCP a very long time to fill the entire link and achieve full utilization.

Several modifications have already been proposed to address the problems of TCP over high bandwidth-delay product channels, mainly based on modifications to the underlying Additive Increase Multiplicative Decrease (AIMD) strategy of TCP [4, 5, 6, 3, 7, 8]. However, we don't know much about these new versions in several aspects yet. First, there is no study where all these versions are compared and analyzed together. Second, they have not been compared against all current TCP versions. Third, it is completely unknown how these new versions perform in old environments, such as "normal" wired and wireless scenarios, which is important for adoption decisions. Finally, it is also unknown how these new versions perform regarding to known TCP issues, such as fairness.

1.2 Contributions of the thesis

This thesis includes the following contributions to the state of the art in transport layer protocols.

- The thesis includes a performance evaluation of TCP in HBDPC where the most important new proposals for these types of networks and old and current TCP versions are analyzed together for the first time.
- Modifications to TCP Vegas and Scalable TCP are proposed to improve their performance in HBDPC.

- New TCP versions for HBDPC are also evaluated over wireless networks and normal wired networks. This is important for users that are thinking about adopting these new versions based only on the performance shown in HBDPC.

- The thesis also addresses the issue of fairness of these new TCP versions to clarify the unknown aspect of whether these versions also confront the same fairness problems of old versions or not.

1.3 Outline of the thesis

The thesis is organized as follows. Chapter 2 discusses in detail the current versions of the TCP protocol's congestion control algorithm and also the new TCP proposals for high bandwidth delay product channels (HBDPC). A performance evaluation of all TCP versions in HBDPC is included in Chapter 3, including proposed modification to TCP Vegas and Scalable TCP to improve their performance in this environment. In Chapter 4 the performance of new TCP versions for HBDPC is analyzed in wireless networks along with the fairness of these new versions. Finally, Section 5 includes the conclusions and points out directions for future research.

CHAPTER 2

LITERATURE REVIEW

2.1 The Transmission Control Protocol(TCP)

Unrestricted access to common resource may result in poor performance in the form of long delays in data delivery, low network utilization, high packet loss rates and even a possible collapse of the communication system. This phenomenon, known as network congestion, occurs when the aggregate demand for a resource (e.g., link bandwidth) exceeds the available capacity of the resource. Network congestion in the current Internet is detected and controlled by the TCP protocol. TCP interprets packet loss as a signal of congestion and reduces its sending rate to alleviate it. If no packet loss occur, TCP increases its transmission rate until it fills the channel capacity and packet loss occur.

TCP follows the Additive Increase/Multiplicative decrease (AIMD) strategy to obtain better performance and deal with congestion. TCP uses a variable called the congestion window to change the output rate of the connection in a manner consistent with the AIMD strategy. As such, the congestion window variable is increased in an additive manner if there is no congestion in the network while it is decreased in a multiplicative manner in the event of packet loss.

TCP also has a flow control mechanism to avoid overflowing the buffers of the receiver and control the way information is injected into the network. Once the source sends all allowed packets as given by the current value of the congestion window, it has to wait for acknowledgments (*ACKs*) in order to increment the congestion window and be able to send new packets. As a result, TCP is usually said to be ”self-clocking”. The source automatically slows down the source when the network becomes congested because acknowledgments are delayed. This feature was the only congestion control mechanism in the Internet before Van Jacobson’s proposal in 1988 [9]. Current TCP versions which detect congestion and adjust their rate differently are reviewed next.

2.2 Current TCP versions

Current widely deployed TCP protocols are TCP Tahoe, TCP Reno and TCP Newreno [9, 10, 11]. The flow and congestion control procedure has two phases: Slow-start and Congestion Avoidance. the Slow-Start phase is used when the connection is initialized and its main goal is to fill the available capacity as quickly as possible. During this phase the congestion window is increased exponentially. It starts with a congestion window of 1 and then increases it by 1 for every ACK received. This continues until the congestion window reaches a threshold called the Slow-Start threshold (*ssthresh*), which is set at the beginning of the TCP connection. Once the congestion window reaches *ssthresh*, the Slow-Start phase ends, and the congestion avoidance phase begins.

In the congestion avoidance phase the congestion control mechanism adopts the standard AIMD (Additive Increase, Multiplicative Decrease) strategy: When no losses are observed, it gradually increases the congestion window variable in an additive manner (additive increase) augmenting the congestion window by $1/\lfloor cwnd \rfloor$ for every incoming ACK, which is roughly equivalent to increasing the window by 1 every *RTT*. When packet losses are detected, the algorithm reduces the congestion window in a multiplicative manner, which is necessary to avoid congestion. Different TCP versions reduce the congestion window in different ways in reaction to packet losses. In the original TCP Tahoe, upon a packet loss, the algorithm reduces the (*ssthresh*) to half, sets the congestion window to 1 and starts the Slow-Start phase again. Figure 2.1 shows the behavior of the congestion window of TCP Tahoe and the two phases just described.

In Reno and its offspring Newreno, the mechanism is modified and the Fast Retransmit/Fast Recovery procedure is included. Under this new procedure, when a packet loss is detected the (*ssthresh*) is set to *ssthresh =max(cwnd/2,2)* and the congestion window to *cwnd=sshtresh+3* avoiding the slow-start phase after each packet loss (or duplicated packet). This new behavior of the congestion window is shown in Figure 2.2 and clearly shows that it offers better performance to TCP compared to the original behavior.

TCP SACK was developed by Floyd and Fall [12] to take care of the inefficiencies of Reno to handle multiple packet drops and the problem of unnecessary retransmissions and long recovery times in Newreno. As mentioned in RFC 2018 [12], both the sender and the receiver must come to an agreement to implement SACK. The receiver is also able to indicate to the sender using Selective

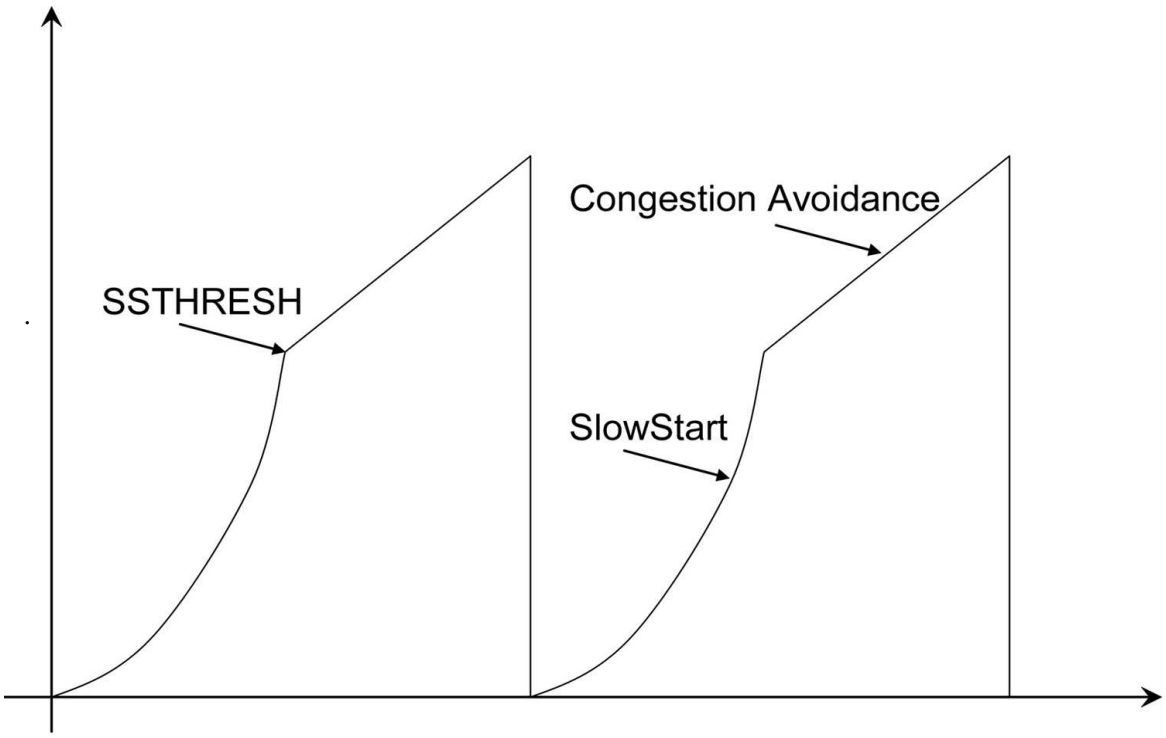

Figure 2.1 Behavior of the congestion window variable in TCP Tahoe

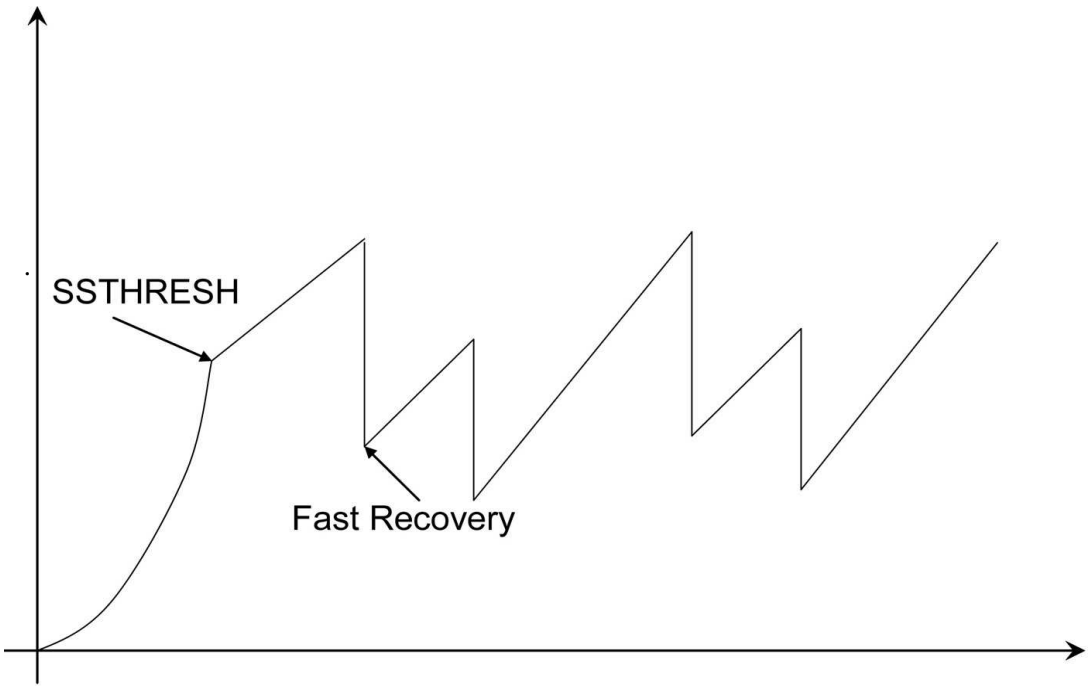

Figure 2.2 TCP Reno and NewReno FR/FR procedure

Acknowledge(SACK) blocks the exact sequence number of the packets that have been received and those missing. This gives the sender then benefit of retransmitting only those packets that are lost. With this accurate information the TCP SACK sender not only retransmits the missing packets but also accomplishes the retransmission during only one RTT, reducing this time compared to the other versions.ting only those packets that are lost. With this accurate information the TCP SACK sender not only retransmits the missing packets but also accomplishes the retransmission during only one RTT, reducing this time compared to the other versions.

TCP Vegas was first introduced by Brakmo et al. in [13]. It introduces several changes compared to the old-fashioned TCP. First, the congestion avoidance mechanism that TCP Vegas uses is quite different from that of TCP Tahoe or Reno. TCP Reno uses the loss of packets as a signal of congestion in the network and has no way of detecting any incipient congestion before packet losses occur. Thus, TCP Reno reacts to congestion rather than preventing it. TCP Vegas, on the other hand, uses the difference between the estimated throughput and the measured throughput of the connection as a way of estimating the congestion state of the network. Now note that this mechanism used in Vegas does not purposely cause any packet loss and therefore removes the oscillatory behavior and achieves higher average throughput and efficiency.

TCP Vegas keeps track of the minimum round-trip propagation delay seen by the connection, which is denoted by the BaseRTT. In NS-2 [14] Vegas uses the first RTT as the initial value for the BaseRTT and replaces its value with any shorter RTT during the connection lifetime. So the BaseRTT at time *t* is estimated to be the smallest RTT seen by the source. With the BaseRTT, TCP Vegas calculate the Expected Throughput using the formula below which represent the maximum throughput that the connection can achieve.

$$Expected \quad Throughput = CWND/BaseRTT \tag{2.1}$$

TCP Vegas also compute the Actual Throughput, which is the throughput that the connection is actually obtaining. This calculation is made using the Equation 2.2 below, where RTT is the normal RTT of the packets during the current congestion window.

$$Actual \quad Throughput = CWND/RTT \tag{2.2}$$

Based on these two values, TCP Vegas calculates the difference between them and increase or decrease the congestion window according to the following formulas.

$$Diff = (Expected - Actual) * BaseRTT \tag{2.3}$$

$$cwnd = \begin{cases} cwnd + 1 & \textit{if Diff} < \alpha \\ cwnd - 1 & \textit{if Diff} > \beta \\ \textit{No change} & \textit{Otherwise} \end{cases} \tag{2.4}$$

The α and β parameters are two thresholds of Vegas, with $\alpha < \beta$. If $Diff < \alpha$, Vegas increases the congestion window linearly during the next RTT. If $Diff > \beta$, then Vegas decreases the congestion window linearly during the next RTT. Otherwise, it leaves the window size unchanged. The rationale behind Vegas is very simple. If the actual throughput is much smaller than the expected throughput, then it suggests that it is likely that the network is congested. Thus, the source should reduce the flow rate. On the other hand, if the actual throughput is too close to the expected throughput, then the connection may not be utilizing the available capacity, and hence it should increase the rate. Therefore, the goal of TCP Vegas is to keep the rate of the connection between the Expected and the Actual rate. Another interpretation is that Vegas maintains a certain number of packets or bytes in the queues of the network [13] given by the α and β parameters. In NS [14], α is set to 1 and β is set to 3, representing the use of at least one but no more than three buffers of the output queue of the bottleneck link [13].

Another difference between TCP Vegas and Reno is the retransmission mechanism. In TCP Reno, a rather coarse grained timer is used to estimate the RTT and the variance, which results in poor estimates. Vegas extends Reno's retransmission mechanism as follows. TCP Vegas records the system clock each time a packet is sent. When an ACK is received, Vegas calculates the RTT and uses this more accurate estimate to decide the retransmission of TCP packet [10]. If it receives a duplicate ACK, Vegas checks to see if the RTT is greater than round trip timeout(RTO). If it is, then without waiting for the third duplicate ACK, it immediately retransmits the packet.

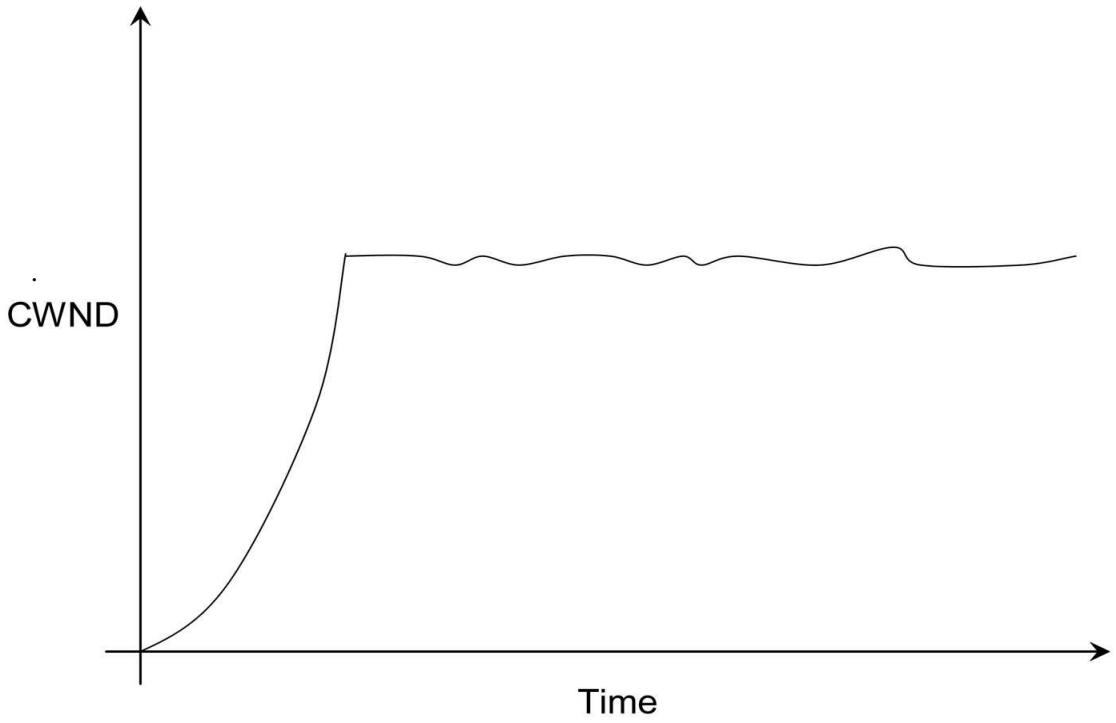

Figure 2.3 Behavior of the congestion window of TCP Vegas

Vegas also modifies the Slow-Start mechanism of TCP. Instead of increasing the congestion window exponentially every RTT,It does this every two RTT. The Figure 2.3 shows the behavior of the congestion window of TCP Vegas, which is quite different from the normal sawtooth of TCP.

In TCP Westwood [15], the sender side measures the bandwidth of the connection by looking at the rate of the incoming acknowledgments. Based on the information it calculates the Fair Share Estimate (FSE), which is then used to appropriately set the values of the cwnd and ssthresh variables after 3 duplicate acknowledgments or a time out. In this way, instead of halving the cwnd, TCP Westwood changes the cwnd and ssthresh to values that are more consistent with the effective bandwidth used by the connection when congestion occurred. This modification seems appropriate since TCP Westwood should not waste as much bandwidth as regular TCP versions during congestion. However, TCP Westwood doesn't modify the Slow Start and Congestion Avoidance mechanisms, using TCP Reno as the underlying protocol.

2.3 New TCP versions for HBDP network

The above widely used TCP protocols tolerate packet delay and packet losses rather gracefully and its underlying congestion control mechanism is the responsible of the stability of the Internet [9]. However, as the channel speeds are increased from the kilobits per second range in the early eighties to the gigabits or even terabits per second range we have today, the window-based mechanism of current TCP implementations starts becoming a problem. It can be easily seen that the widely adopted TCP versions described above substantially underutilize the network bandwidth over high bandwidth delay product channels. For example, In order for TCP to increase its window to achieve full utilization of a 10Gbps channel with 1500-byte packets, it requires over 83,333 RTTs. With a 100ms RTT, it takes TCP approximately 1.5 hours to achieve full channel utilization. Also, after a simple packet loss, TCP reduces its congestion window by half, responding too drastically and again underutilizing the bandwidth of the channel. Using the well known formula that models the throughput of a TCP connection in steady state [16], it was pointed out that TCP needs to have one loss event per 5,000,000,000 packets transmitted to achieve the necessary transfer rate to fill one of these high bandwidth channels. This packet loss rate is less than the theoretical limit of current networks' bit error rates [3]. After recognizing these limitations, several promising new protocols have been put forward. This is the case of Scalable TCP (STCP) [4], High Speed TCP (HSTCP) [3], eXplicit Control Protocol [7], and FAST TCP [6]. These protocols adaptively adjust their increase/decrease rates based on the current window size. So the larger the congestion window is, the faster it grows, and when the congestion occurs, the slashed window size is more accurate to obtain a short recovery time.

HighSpeed TCP was introduced in [3] to achieve high throughput in high bandwidth-delay product links without requiring unrealistically low packet loss rates. In addition, HighSpeed TCP is TCP-friendly when competing with regular TCP versions. HighSpeed TCP modifies the TCP response function making the parameters and dependent of the current congestion window and utilizes a pre-computed lookup able for finding the values of $a(w)$ and $b(w)$. HighSpeed TCP needs another solution proposed in [17] to limit the number of segments by which a congestion window can be increased during the Slow Start phase. During the Slow Start phase, TCP's congestion

window variable can reach a very large number and a very large number of packets can get dropped if the congestion window is doubled just right before the channel is about to be full of packets.

The eXplicit Control Protocol (XCP) [7] works with the help of two controllers embedded in the routers that calculate the feedback that sources will use to adjust their behavior and make the system to converge to optimal efficiency and minmax fairness. XCP modifies the packet headers to include information about the connection so that routers can make these calculations and send explicit feedback to the sources. Although XCP was shown to perform very well, it modifies the packet structure and needs router support. In addition, further analysis are needed to investigate XCP's performance when the connection is initiated. In [7], the authors claim that XCP reaches the desired rate after one RTT but they never show it.

Scalable TCP (STCP) [4] is a sender-based modification that builds on HighSpeed TCP meant to utilize high bandwidth delay product links in a simple and robust manner. Scalable TCP utilizes constants for the parameters a and b of TCP(a,b) to achieve similar goals than HighSpeed TCP. As such, it is still based on the same underlying AIMD strategy used by TCP. Further evaluations are needed to make sure Scalable TCP doesn't inherits the same disadvantages of TCP, such as the long times required to achieve full utilization after Slow Start and Congestion Avoidance procedures. Scalable TCP also relies on packet loss to change the congestion window.

The congestion control algorithm of STCP works as follows. For each acknowledgment received in a round trip time in which congestion has not been detected, it increase the *cwnd* by a fixed amount equal to 0.01. Upon detection of congestion in a given round trip time, STCP reduces the *cwnd* by a fixed amount equal to 0.125 the current window. The congestion control algorithm of STCP is then based on the general window update algorithm of TCP as follows:

$$\begin{aligned} cwnd &\leftarrow cwnd - \lfloor b * cwnd \rfloor \\ cwnd &\leftarrow cwnd - \lfloor 0.125 * cwnd \rfloor \\ cwnd &\leftarrow cwnd + a \\ cwnd &\leftarrow cwnd + 0.01 \end{aligned} \tag{2.5}$$

The values of $a = 0.01$ and $b = 0.125$ were found considering STCP's impact on legacy traffic, bandwidth allocation properties, flow rate variance, convergence properties, and control theoretic

stability [4]. However, as it is shown in the following chapter, the performance of STCP is not what it is supposed to be.

FAST TCP [6] uses packet loss and queueing delay to assess congestion and to solve the problem of very large packet loss interarrival times. FAST reacts to packet losses as TCP Reno but queueing delay information is used to change the congestion window when no losses occur. FAST borrows some ideas from TCP Vegas to estimate the queuing delay and increase the congestion window. In [6] it is said that the implementation should be less aggressive than Slow Start and less drastic than rate-halving.

In this thesis, only HSTCP and Scalable TCP are considered because the detailed description of FAST TCP is not yet available. XCP is not considered either because it needs router support, making its deployment much more difficult.

CHAPTER 3

PERFORMANCE EVALUATION OF TCP OVER HIGH BANDWIDTH DELAY PRODUCT CHANNELS

3.1 Simulation topology and parameter

In this Chapter the description of the simulation topology and parameters used to evaluate the performance of TCP Tahoe, Reno, New Reno, SACK, Vegas, Westwood, HighSpeed TCP and Scalable TCP is included. The network topology utilized consists of one TCP source, one TCP sink node (destination), and two routers connected by a bottleneck link as shown in Figure 3.1. The Network Simulator 2 (NS-2) [14] is the simulation tool utilized to run all the experiments.

To carry out the experiments, in the simulations, the maximum values of the congestion window for the TCP versions are set such that the connections can achieve full link utilization. The bottleneck link two way propagation delay is fixed and set to 25 msec while the link's bandwidth is varied to show the performance of these protocols as the bandwidth-delay product increases. The buffer size at the bottleneck link is set to 200 packets to absorb part of the sudden congestion. We perform experiments to show the link utilization and congestion window behavior of each protocol as well as the packet loss rate during the Slow Start Phase and the time to reach full link speed after a packet drop event.

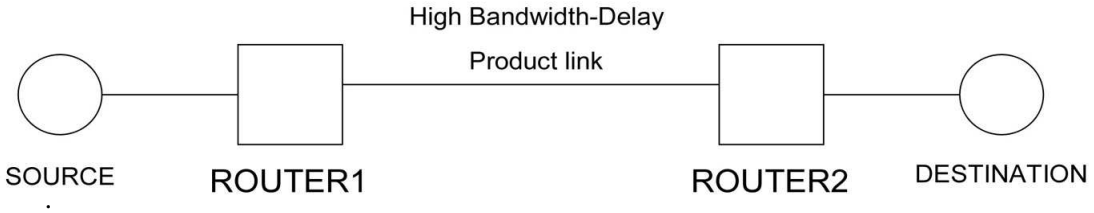

Figure 3.1 The network topology

3.2 Performance evaluation

Figures 3.2 and 3.3 plot the normalized throughput achieved by the protocols under consideration as a function of the link bandwidth, and the TCP sequence numbers when the link bandwidth is set to 1 Gbps. As the figures show, there are considerable differences among these protocols. As a general trend, it can be seen that the performance of most protocols degrade as the bandwidth increases showing clear scalability problems. This is the expected behavior and confirms what other researchers have found. Only TCP Vegas, Westwood 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 before in a HBDP scenario. The TCP versions also perform as expected with Tahoe presenting the worst performance followed by Reno, New Reno 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. This is the same behavior widely known of these versions in lower normal speed channel, therefore, is not investigated here any further. Instead, this thesis focus on the proposed new versions. Finally, TCP Westwood improves over the regular TCP versions but still below HighSpeed TCP and Vegas, the throughput performance of the protocols in HBDPC can be explained looking at the behavior of the congestion window variable.

The behavior of the congestion window of the different protocols is analyzed next. In Figures 3.4, 3.5, and 3.6 the cwnd of the protocols over time in the case where the bottleneck bandwidth is set to 1 Gbps is plotted. With the exception of Vegas, the figure shows the expected sawtooth pattern of TCP. There, 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 the current value. Reno presents deeper and longer reactions while New Reno and SACK are very similar. Interesting behaviors are the ones experienced by TCP Vegas, Westwood and HighSpeed TCP. TCP Westwood achieves better throughput because its cwnd doesn't drop as deep as the regular TCP versions, guided by the Fair Share Estimate (FSE). It will be shown later that TCP Westwood goes through a rather long Congestion Avoidance phase. The cwnd of HighSpeed TCP takes values similar to TCP Westwood but it achieves better throughput because it manages to transmit more packets, in particular at the beginning of the connection. HighSpeed TCP presents the oscillatory behavior also experienced in the simulation results in [18]

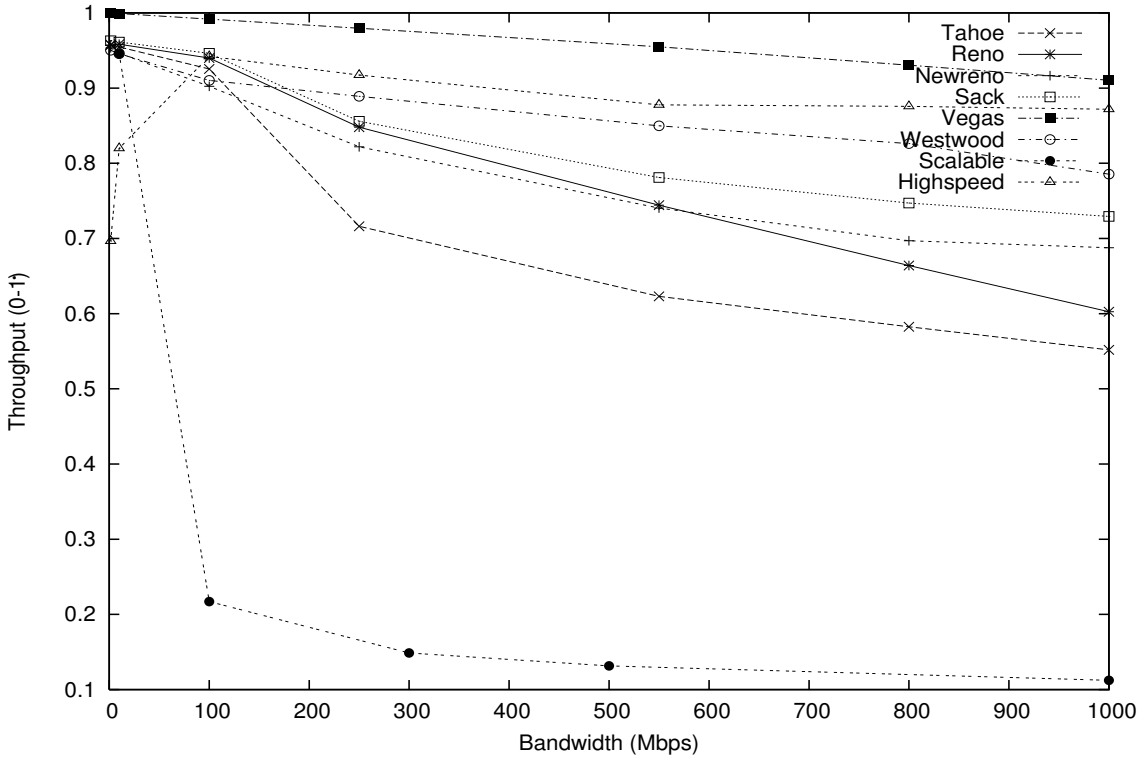

Figure 3.2 Channel utilization of TCP as a function of the bottleneck link bandwidth

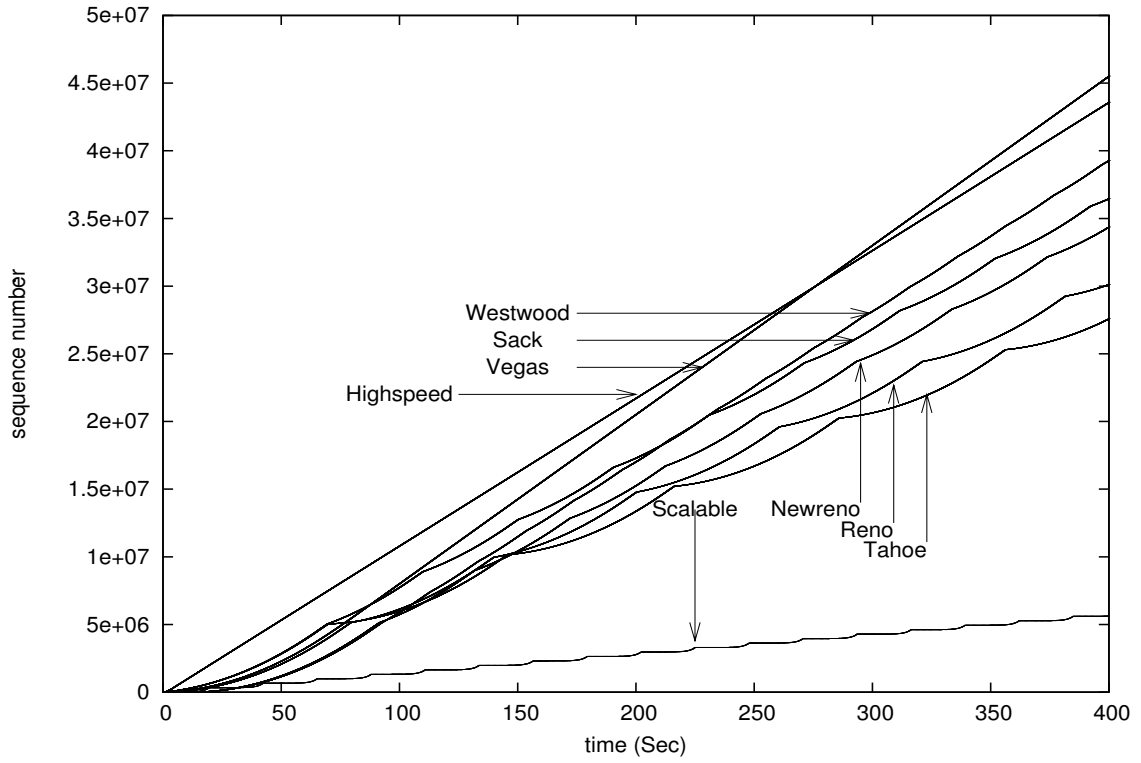

Figure 3.3 Sequence number of TCP with the bottleneck link bandwidth set a 1 Gbps

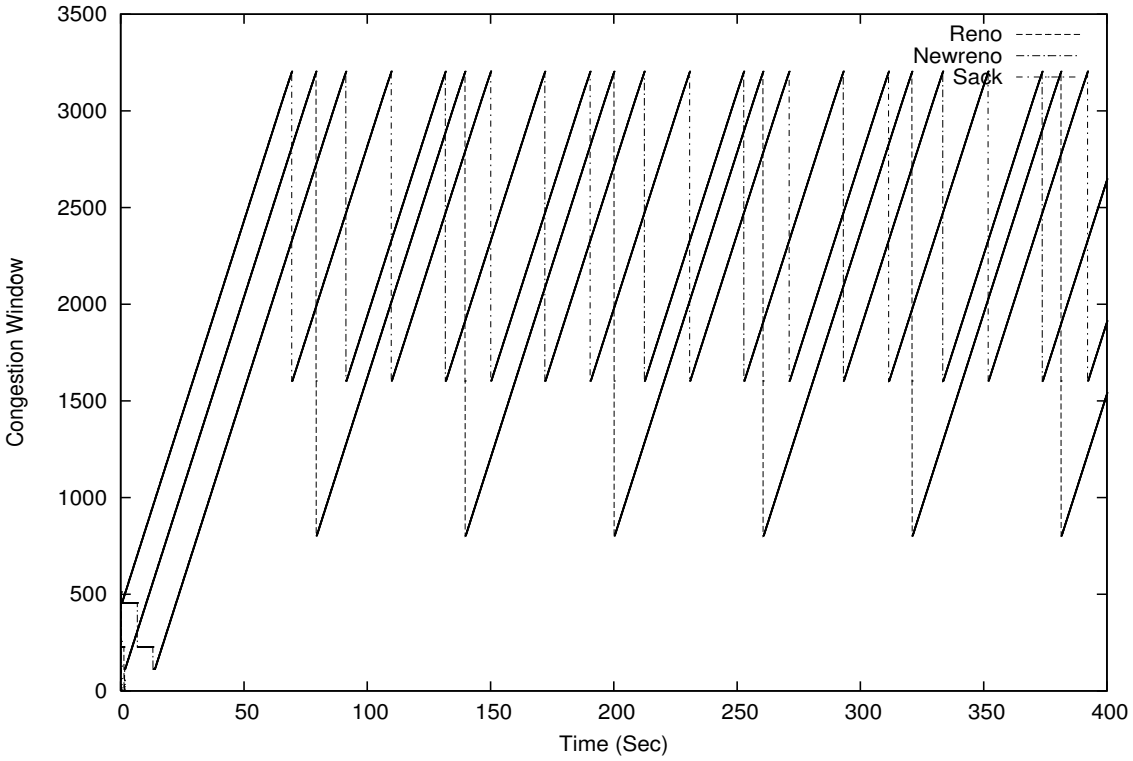

Figure 3.4 Congestion Window of TCP Reno, New Reno and SACK when the link bandwidth is set to 1 Gbps

when only one source is in the system. TCP Vegas's behavior is even better as it can be seen that 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. In order to achieve full link utilization, the mechanism used by Vegas could be a good way to go. If we use the case where the bottleneck bandwidth is set to 1 Gbps as an example, we know that the theoretical value of the congestion window needed to achieve full link utilization is about 3325 packets, given by the bandwidth-delay product of the network plus the buffer size. It can be observed from Figures 3.4, 3.5 and 3.6 that this is in fact the maximum value achieved by all protocols and that TCP Vegas' congestion window is very much steady and at a very close value after the Slow Start phase, indicating that TCP Vegas does a very good job estimating the available

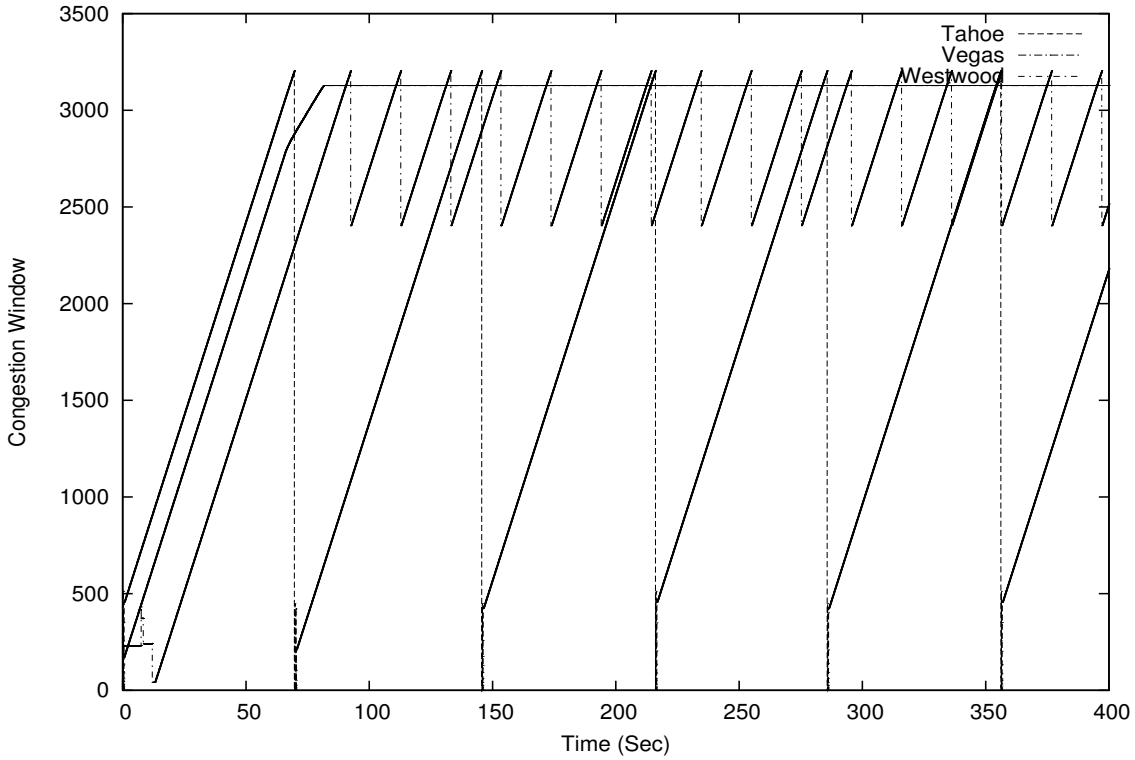

Figure 3.5 Congestion Window of TCP Tahoe, Vegas and Westwood

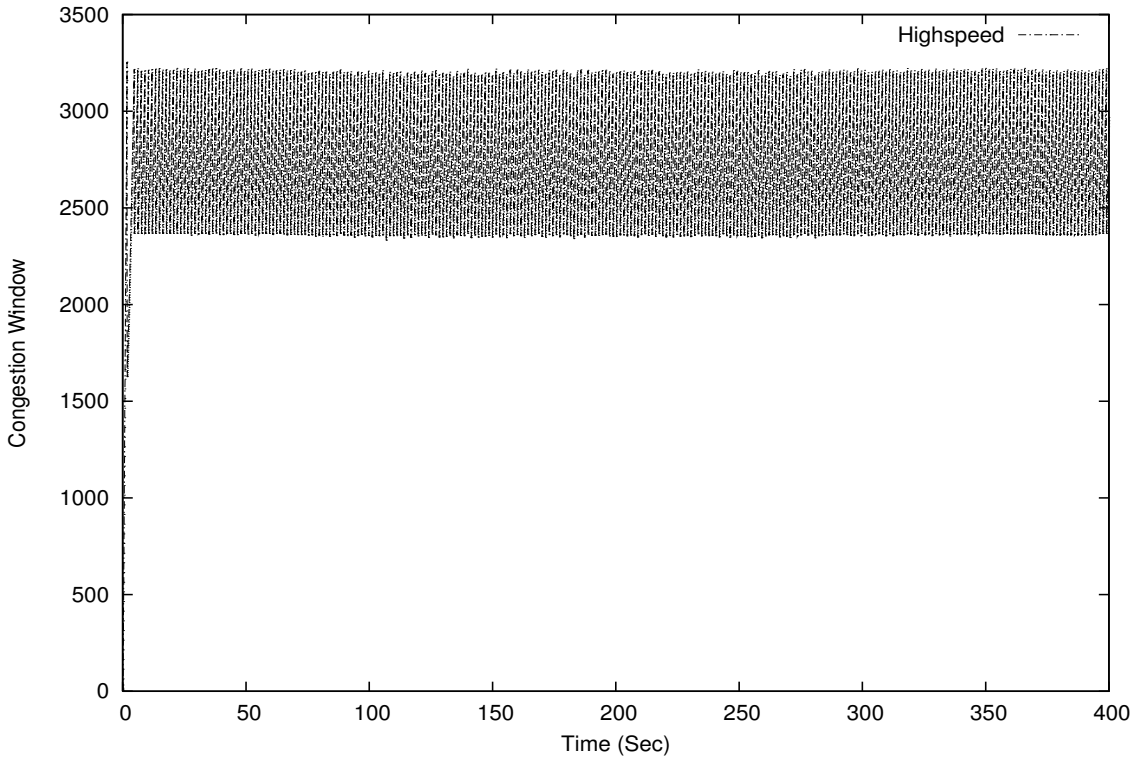

Figure 3.6 Congestion Window of TCP HighSpeed TCP

bandwidth. The main problem of Vegas relies in the first Congestion Avoidance phase; it takes Vegas a rather large amount of time to reach the 3325 value for the first time.

Next, the performance of the protocols during the Slow Start phase is evaluated. Here, the interest is in the Slow Start time and the Packet Loss Rate (PLR) during that period of time. The Slow Start time is important because the longer it takes the more the wasted capacity. The PLR is an indication of how efficient the Slow Start mechanism is. Obviously, the higher the PLR the worse. The PLR is measured as the number of packets lost divided by the total packets sent during the Slow Start phase. From Figure 3.7 it can be seen that all protocols have a similar and very short Slow Start duration. This is expected as they all use the same exponential mechanism. Vegas has a slightly longer duration because it increases the cwnd exponentially but every other RTT; however, 0.4 seconds is still a very short Slow Start time. The second observation is that most TCP versions have 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 no matter 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* achieved during Slow Start and consequently, a 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 our 1 Gbps case) when in reality the system can only absorb around 6250 packets. However, if the buffer size is set to more realistic values, as in this case, the *cwnd* will grow to modest values and not too many packets will be dropped. For instance, in this case the PLR was in the order of 6%. An interesting point to mention here is the fact that TCP Vegas and HighSpeed TCP were the only protocols with almost zero PLR. While TCP Vegas' Slow Start phase takes a little bit 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 explained in [19].

The performance of the protocols during the Congestion Avoidance phase is also considered. Here, the focus is on the Congestion Avoidance phase time. We call this time the *recovery time*, or the time that it normally takes the congestion window to reach its maximum value after a drastic reduction because of a packet drop. Figure 3.9 shows this time in seconds for the different protocols as the bandwidth of the channel is increased. As 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

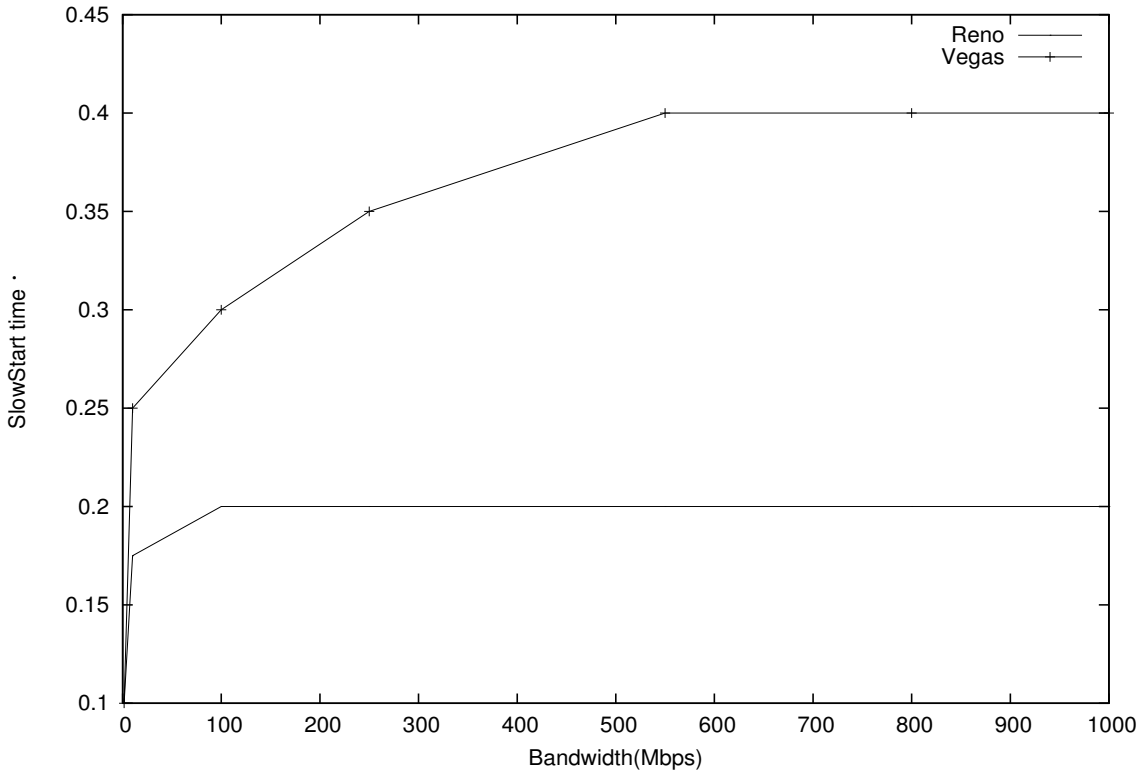

Figure 3.7 Duration of the Slow Start phase when the bottleneck bandwidth is set to 1Gbps

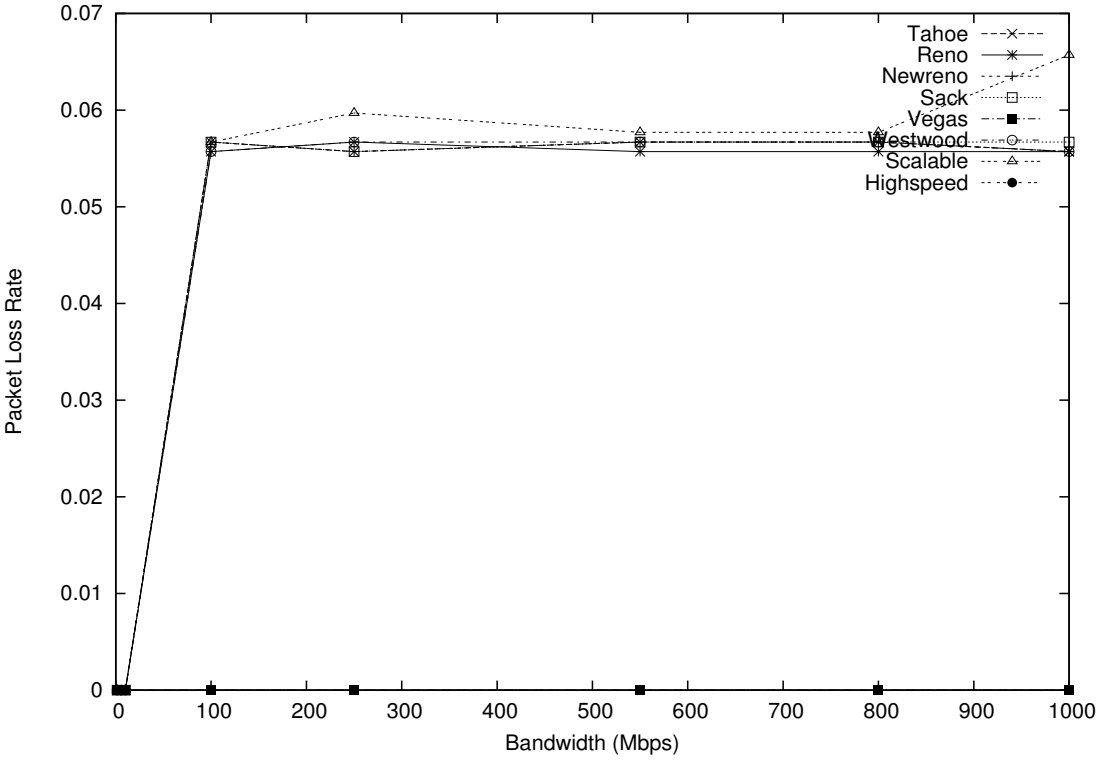

Figure 3.8 PLR during the Slow Start phase when the bottleneck bandwidth is set to 1Gbps

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 that this protocol presents as observed in Figures 3.4, 3.5, and 3.6. For this experiment, only data from the first Congestion Avoidance phase was utilized. Another interesting point is related to TCP Westwood. 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 3.10 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 during the initial phase of the connection and how the Congestion Avoidance phase starts with a very small *cwnd* and *ssthres* values. As a result, it 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 until 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 is a high value because the network is empty. However, the actual bandwidth decreases substantially, since the exponentially increase of the *cwnd* fills the buffers quite fast. At this time, Vegas losses some packets, reduces its *cwnd* and then it enters 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* stays 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. We can solve this problem modifying the Slow Start phase procedures or the algorithms that drive the Congestion Avoidance phase. In this thesis, we try the latter approach.

3.3 Modification of TCP Vegas

As shown in the last section, TCP Vegas has several good properties that make it a good candidate for high bandwidth-delay product links. For example, TCP Vegas experiences minimal packet losses during the Slow Start phase and no drastic reductions in the cwnd, and its bandwidth esti-

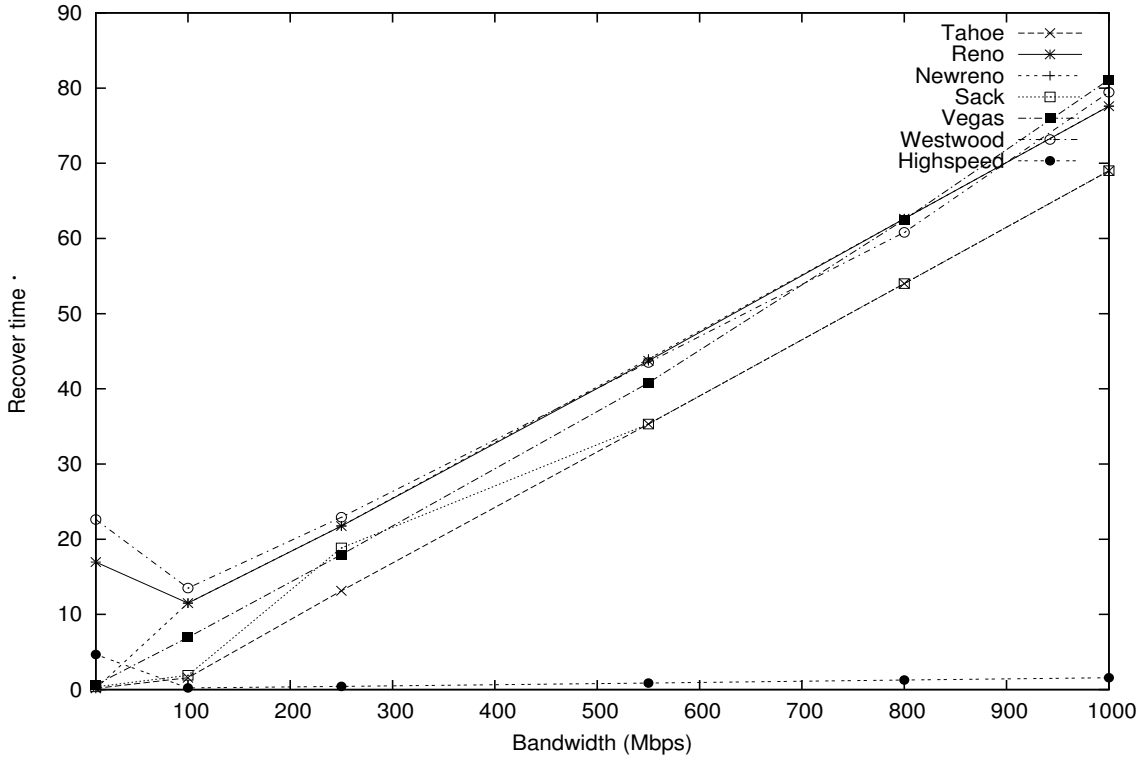

Figure 3.9 Recovery time of the protocols as a function of the bandwidth of the bottleneck link

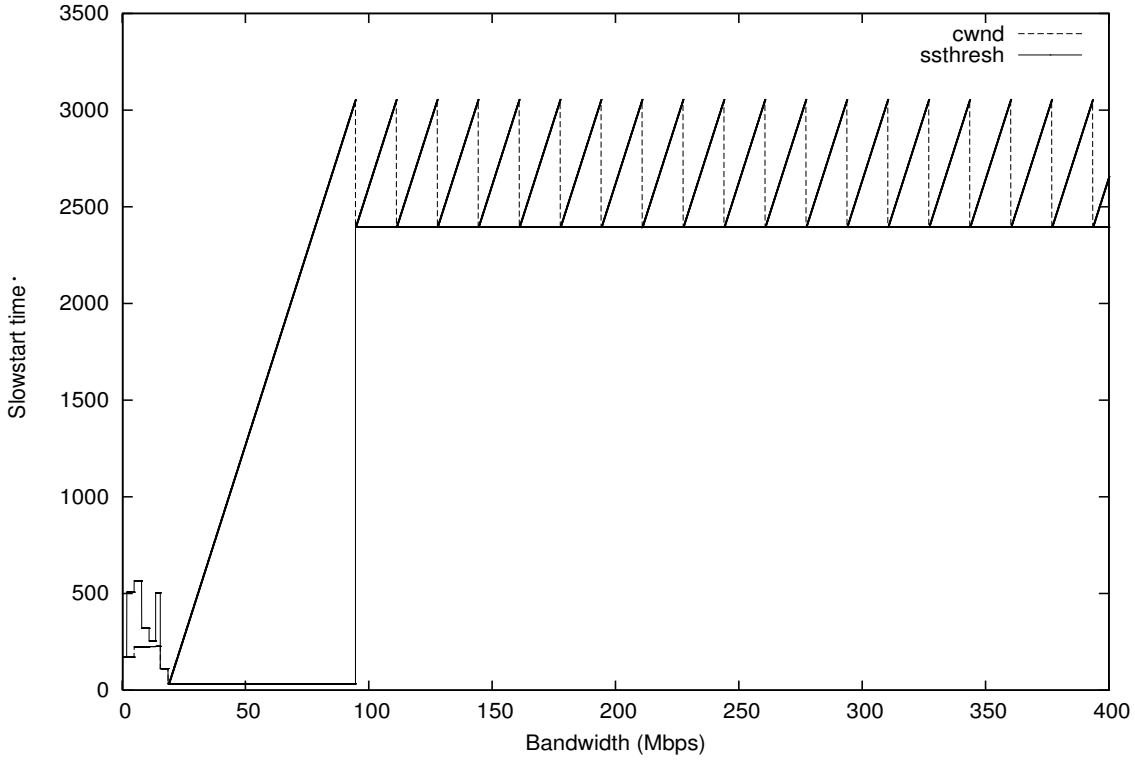

Figure 3.10 *cwnd* and *ssthres* of TCP Westwood over time

mation technique allows it to maintain its cwnd steady and at a very close value to the maximum possible.

Although TCP Vegas give the best performance in the simulations, it is not a perfect solution for the congestion control in high bandwidth-delay product networks. First,TCP Vegas's Slow-Start is too slow.It increases the window size every other RTT, which increases the time to fulfill the capacity of the link. The second aspect is about the friendliness. Consider about the following situation: TCP Vegas competes with TCP Reno for the limited bandwidth of bottleneck link. TCP Reno's Slow Start and Congestion Avoidance schemes are both aggressive compared to Vegas in the sense that they leave little room in the buffer for other connections, while TCP Vegas is conservative and tries to occupy little buffer space.

When a TCP Vegas connection shares a link with TCP Reno connection, the TCP Reno connection uses most of the buffer space and the TCP Vegas connection backs off, interpreting this as a sign of network congestion. So it is necessary to achieve fairness between Vegas and Reno connections for deployment of TCP Vegas into the operating network. Following, we propose two very simple modifications to TCP Vegas that will allow it to achieve higher throughput in high bandwidth-delay product links and mitigate the friendliness problem.

The first proposed modification changes the slow start phase of Vegas so that it increases its cwnd exponentially every RTT as regular TCP versions do. Although this modification in fact reduces Vegas's Slow Start time, it increases TCP Vegas' efficiency by an almost unnoticed margin. This is because the Slow Start phase is very short. In addition, it is worth mentioning that this modification interferes with the bandwidth estimation procedures of Vegas, and therefore it is not recommended.

The second modification is meant to reduce the time Vegas spends in the Congestion Avoidance phase or Vegas' *recovery time*. Instead of increasing and decreasing the *cwnd* by one every RTT following Equation 2.4, the proposed modification increases the *cwnd* by a factor γ and reduces the *cwnd* by a factor θ. As a result, the formula that drives the behavior of Vegas during the Congestion Avoidance phase is given by Equation 3.1 as follows:

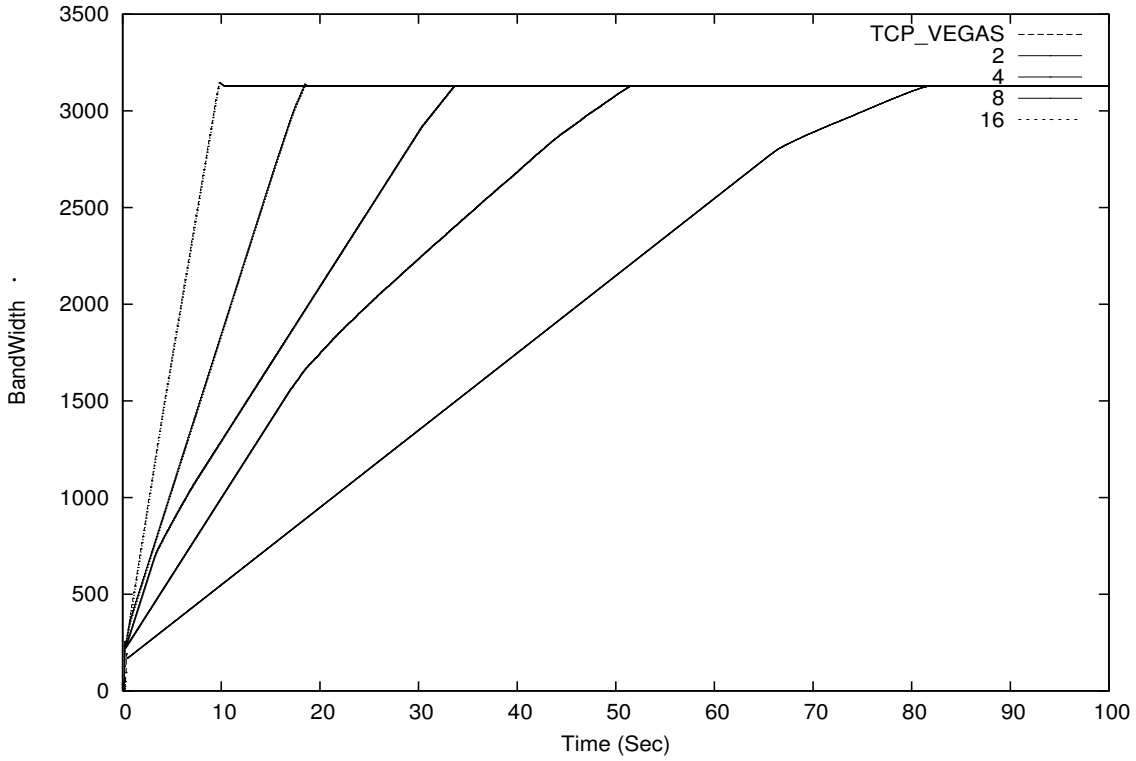

Figure 3.11 Congestion Window of Vegas and the modified Vegas for different values of γ when the bottleneck link bandwidth is set to 1Gbps

$$cwnd = \begin{cases} cwnd + \gamma & if\ Diff < \alpha \\ cwnd - \theta & if\ Diff > \beta \\ cwnd & Otherwise \end{cases} \tag{3.1}$$

Equation 3.1 allow us to play with the aggressiveness and responsiveness of Vegas just using different values of γ and θ. The results presented here utilize a configuration that makes the Vegas version more aggressive setting the value of γ equal to 2, 4, 8, and 16 and θ equal to 1. Figure 3.11 compares the cwnd of Vegas as a function time in the case where the bottleneck link is set to 1 Gbps with and without the proposed modifications. As it can be seen, the *recovery time* reduces the as the value of γ is increased from a value close to 80 seconds to a value of only 10 seconds. In addition, it can also be seen from Figure 3.12 that this change also translates into better throughput.

The modified Vegas with equal values of θ and γ set to 2, 4, 8, and 16 was also tried. In this case, it is equally aggressive and responsive. Results not shown here demonstrate that these versions achieve similar performance than in the case shown in Figures 3.11 and 3.12 where only

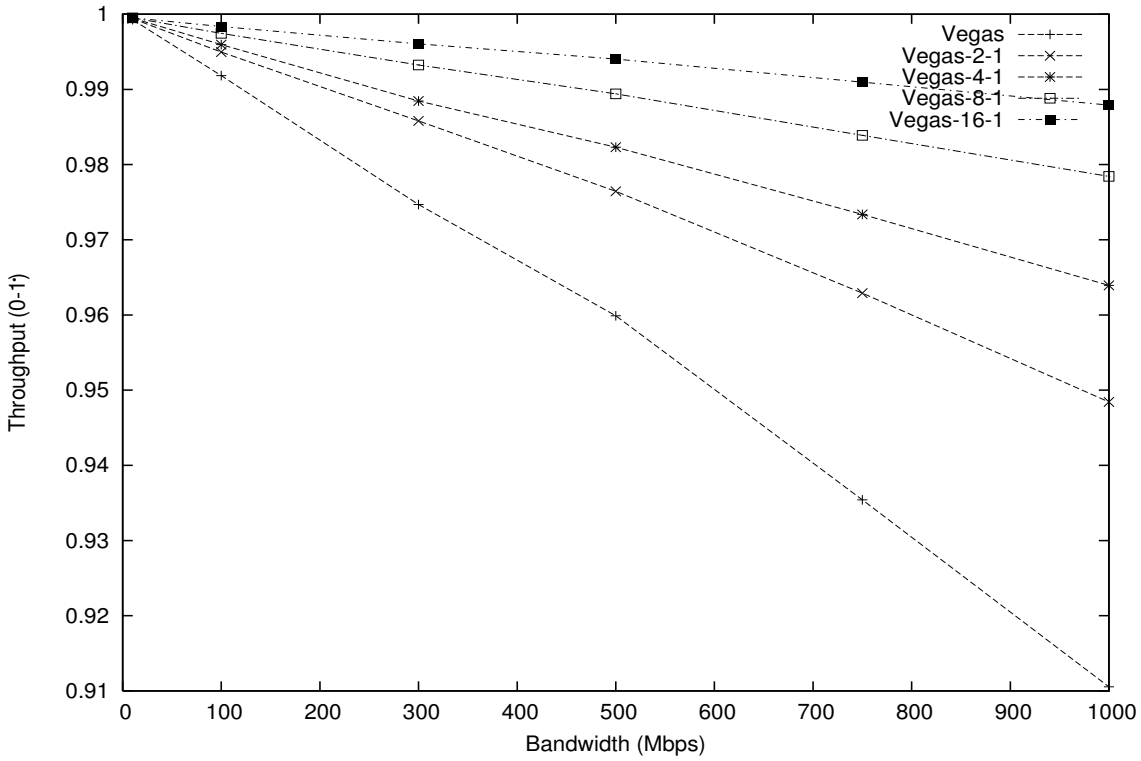

Figure 3.12 Channel Utilization of Vegas and the modified Vegas for different values of γ when the bottleneck link bandwidth is set to 1Gbps

the γ variable is changed except in the case where γ and θ are set to 16. In this particular case, the throughput of the connection reduced to half. This effect and the variation of these two variables needs further investigation.

3.4 Modification to Scalable TCP

Although Scalable TCP was announced to have negligible impact on existing network traffic while improving bulk transfer performance in highspeed wide area networks [4], this advantage is not seen here. Actually, using a limited buffer size, its performance is worse than expected. Figure 3.13 shows that using realistic buffer sizes, e.g. 200 packets of buffer space in the output queue of the bottleneck link, the STCP protocol performs poorly in HBDPC. Analyzing the simulation results of STCP, it can be inferred that the use of the fixed value of $a = 0.01$ as the increment of the congestion window is where the problem is. The congestion window size is increased with that fixed value for each acknowledgment received in a round trip time. As the congestion window size increases, the sum of the increment also increases. In HBDPC, in order to achieve high bandwidth

utilization, the congestion window value needs to grow to really large values and, if a fixed value is used as the increment, the increment of the cwnd will become correspondingly large for every RTT.

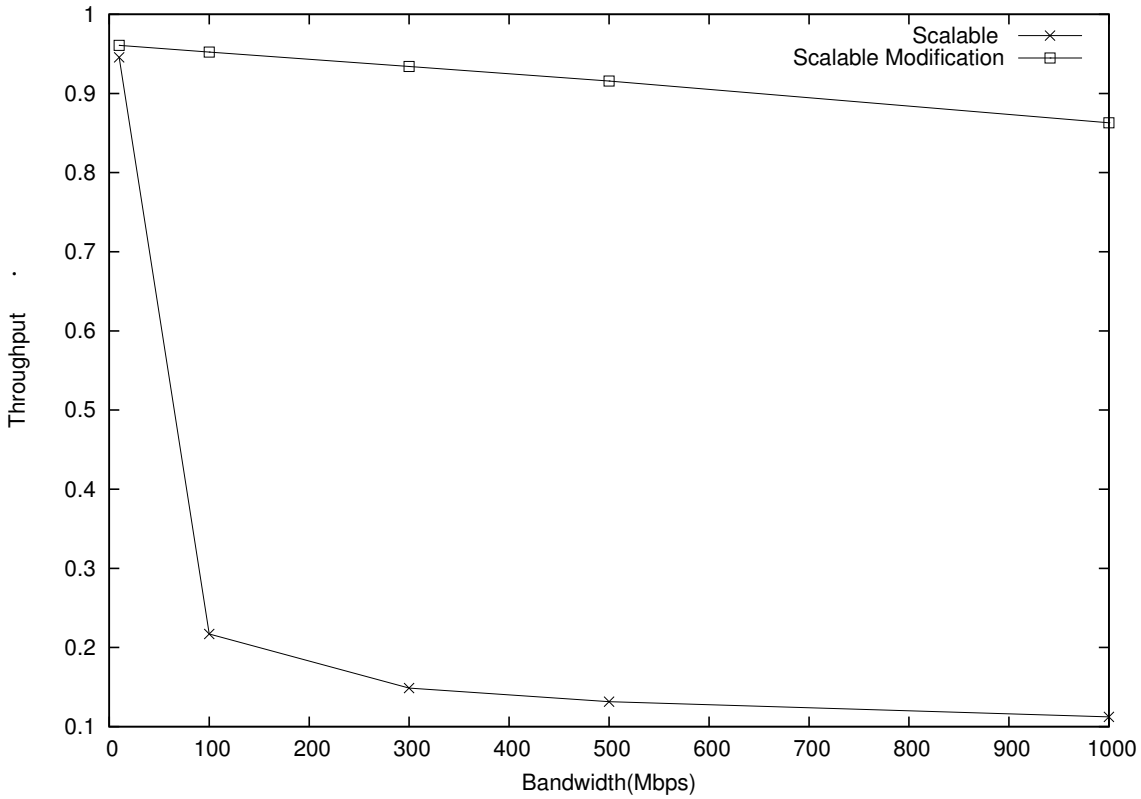

Figure 3.13 Throughput of STCP and the modified STCP for different values of a

In Figure 3.13, it can be seen that it takes 19.25 seconds for STCP's congestion window to reach the peak size (*cwnd*=3172). After that, congestion occurs, and STCP reduces the *cwnd* to $7/8$ its current size. The connection stays in that size and after finding that it can not resolve the congestion in the networks it cuts $1/8$ of the congestion window again. Since congestion remains, a timeout occurs and the connection restarts its Slow-Start phase with a window size of one. Because it takes a long time for the network to "recover" from the congestion and most of the bandwidth dissipates in that "recovery" period, to avoid this situation, a more adaptive mechanism which will be more cautious in the way its congestion window is increased, is needed. As a result, the increase factor a of Scalable TCP is changed back to the same value of the normal TCP versions as follows:

$$cwnd \leftarrow cwnd + 1/\lfloor cwnd \rfloor \tag{3.2}$$

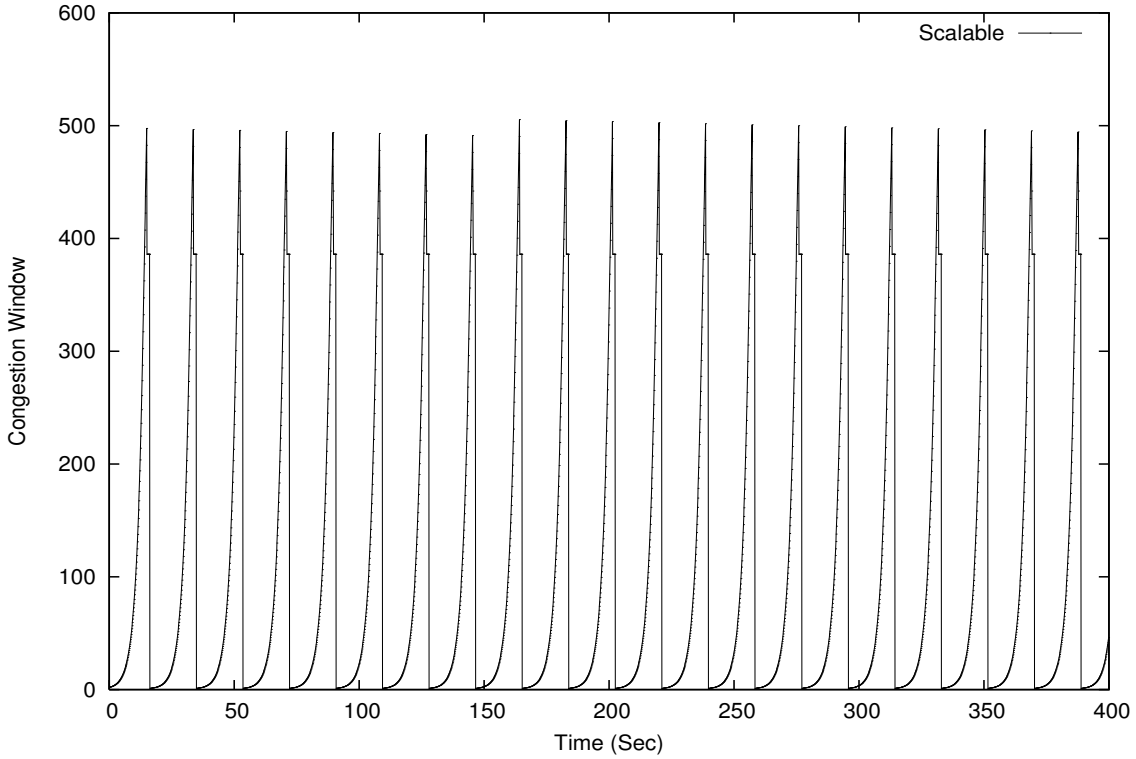

Figure 3.14 Congestion Window of STCP when the bottleneck link bandwidth is set to 1Gbps

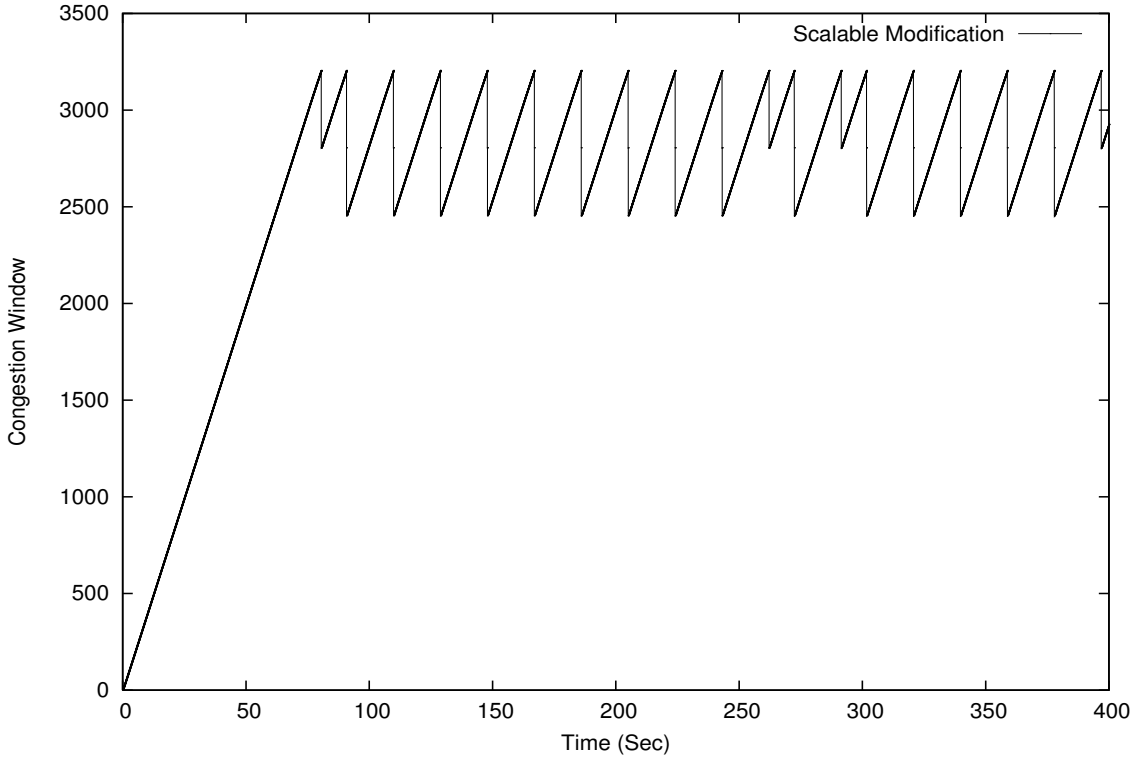

Figure 3.15 Congestion Window of modified STCP in the 1Gbps case

This modification actually makes STCP behave as the standard AIMD algorithm, and because it slashes the window size by 1/8 instead of the one half, it shows better performance. This is shown in Figure 3.13. Comparing Figures 3.14 and 3.15, which show the congestion window of the original STCP and modified STCP, it can be seen that the time needed by the modified STCP to reach the first peak size of the congestion window (80.525s) is longer than the original STCP. However, since its increment is considerably less aggressive than the original version, the congestion that the modified version causes is also considerably smaller, and therefore, the connection needs only 0.025s to resolve the problem and step into the congestion avoidance phase. On the other hand, the original STCP needs 23s to resolve the congestion and get back to that level. That is the reason why the modified STCP performs better than the original STCP.

CHAPTER 4

PERFORMANCE EVALUATION OVER WIRELESS NETWORKS AND FAIRNESS ANALYSIS

New TCP proposals for HBDPC have started to emerge and new ones are expected soon. So far, these versions have proved one way or another than they do a better job than current TCP versions in this specific environment. However, nobody has analyze how these new versions perform regarding old or current technologies and where they stand in terms of some important issues that have been analyzed for old version, such as fairness. These are the two main aspects analyzed in this chapter.

4.1 Simulation topology

It is interesting to note that all these new TCP proposals for high bandwidth delay product channels do in fact improve the performance of TCP in this environment. However, do all these new versions also perform well in common wireless environments such as the well known wired cum wireless scenario shown in Figure 4.1? This is a fair question since all these scenarios are going to coexist for quite some time and we all want to have the best performing TCP version in most of them. The upcoming analysis will help users decide which TCP version to use according to their needs.

Figure 4.1 shows the well-known two nodes wired-wireless topology that has been used in many other studies [20] [21]. In this topology, a sender connected to a wired infrastructure communicates with a receiver in a wireless local area network with a wireless access point connecting these two different technologies. The sender is connected to the base station by means of a 10Mbps capacity channel with 20ms of propagation delay. The base station implements the widely known IEEE 802.11 standard working at 2 Mbps with a wireless channel characterized by negligible propagation delay. A TCP agent runs end-to-end from the sender to the receiver. An infinite FTP source at the sender generates packets of length 1000 bytes.

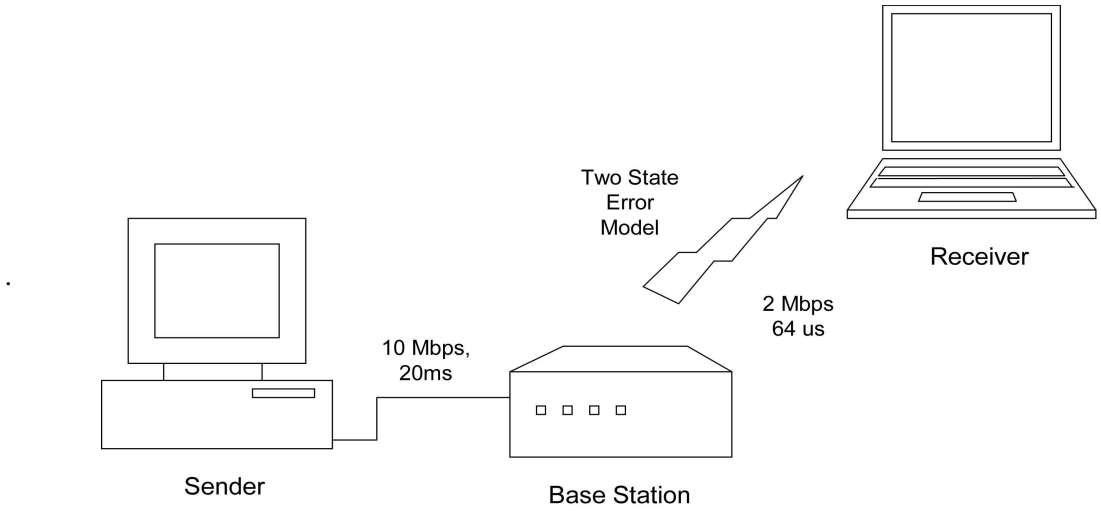

Figure 4.1 Simulation topology

As suggested in [22, 23, 24], a two state Markov model generates errors in the wireless channel to simulate the effects of the wireless media. The two state Markov model consists of a good state and a bad state. It remains in each of the two states for a mean duration of time after which it moves on to the next state. A packet is transmitted when the system is in the good state and dropped otherwise. A series of packets in the good state are thus transmitted whereas the next series of packets in the bad state are dropped. This is an appropriate model for the wireless channel where correlated errors are common. Figure 4.2 shows the two state Markov chain with the steady state transition probabilities of λ and μ.

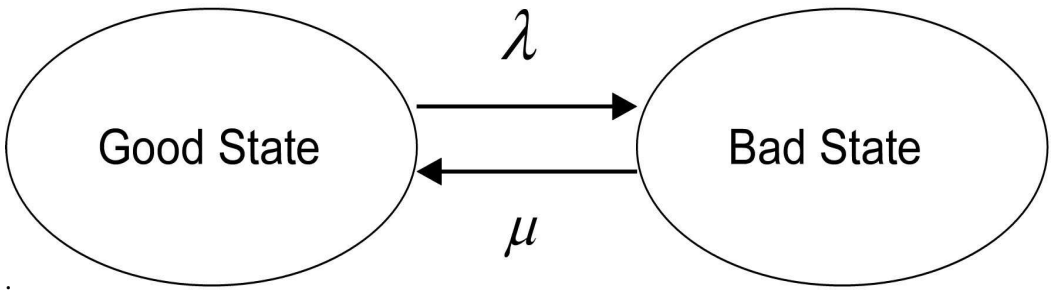

Figure 4.2 Two state Markov chain to model errors in wireless channels

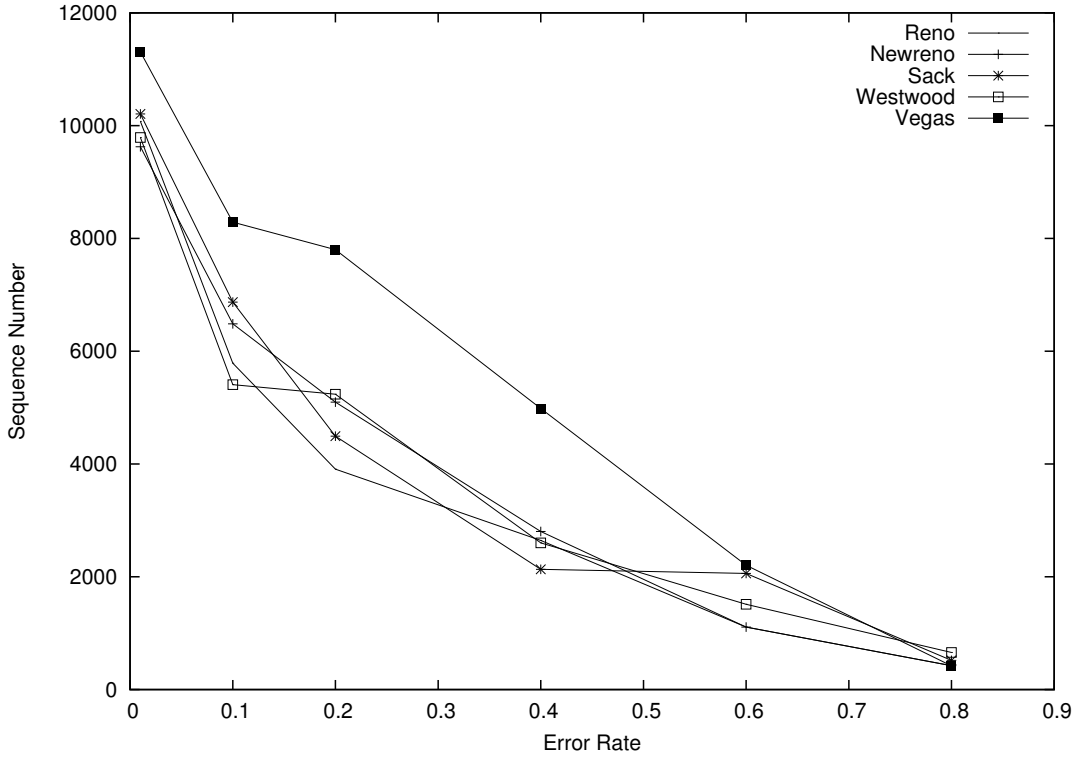

Figure 4.3 Throughput of the current TCP versions under consideration as a function of the channel errors

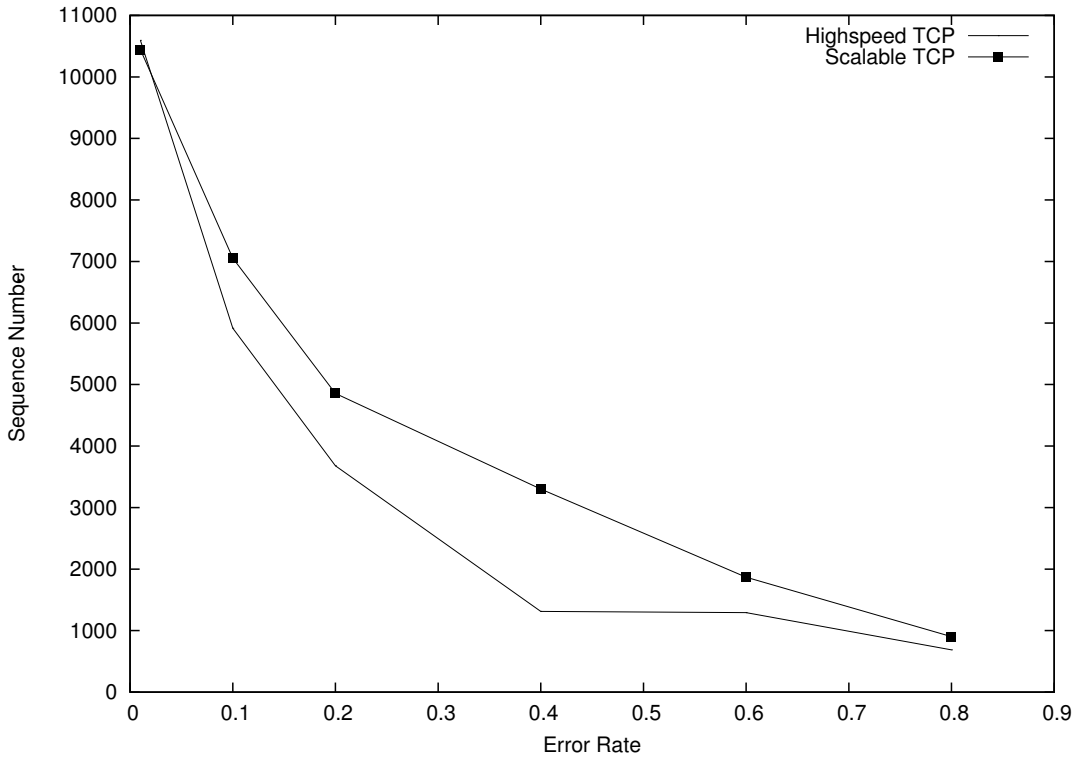

Figure 4.4 Throughput of the TCP versions for HBDPC under consideration as a function of the channel errors

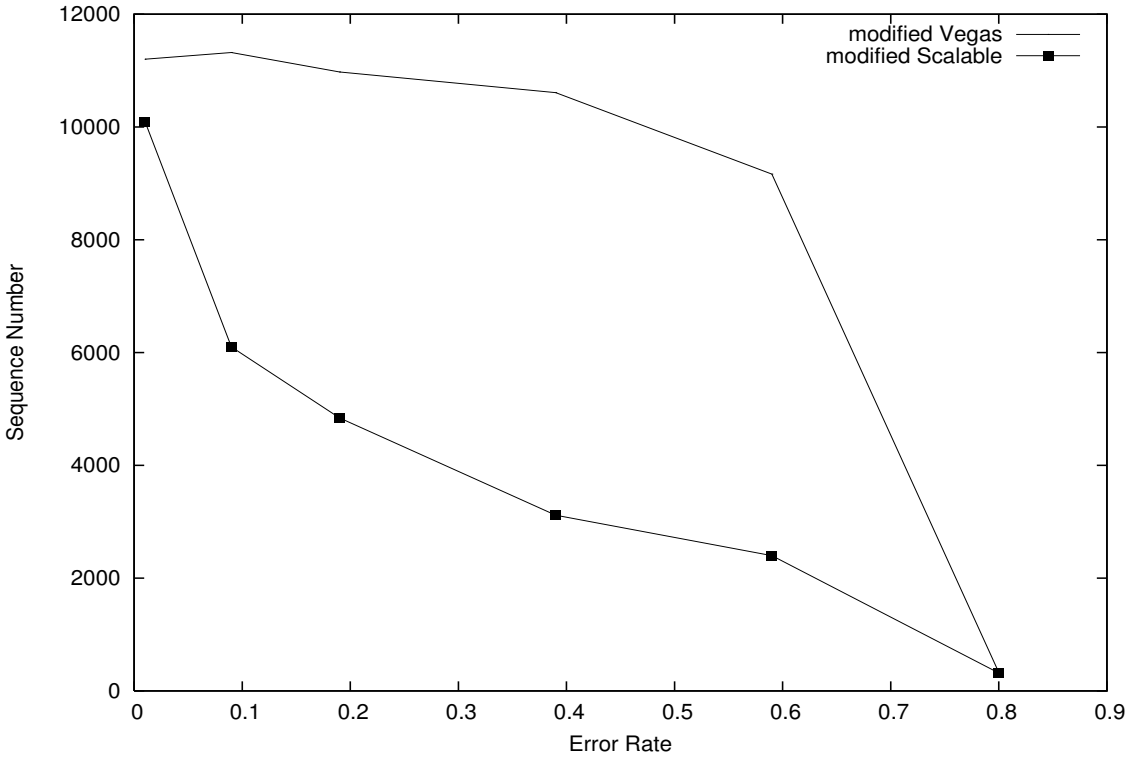

Figure 4.5 Throughput of the modified TCP versions under consideration as a function of the channel errors

4.2 Performance evaluation

Figures 4.3 and 4.4 show that the throughput achieved by all the current TCP and HBDPC TCP versions as a function of the errors introduced in the channel. In the graphs, the X axis represents the percentage of time the Markov chain is in the bad state. First, it can be seen that the behavior of all TCP versions is totally expected. As the channel errors increase, the throughput decreases. Also, if the channel is very error prone, most TCP versions perform similarly. However, as the channel conditions improve the congestion control mechanisms of the TCP versions differ and some versions provide better performance than others. Again, Vegas is the best performing version because it detects losses faster, reduces its congestion window fewer number of times and avoids more timeouts [13]. Comparing the two graphs, it can be seen that HSTCP and STCP also perform worse than TCP Vegas in this environment. However, the new TCP proposals perform differently over wired than wireless networks. Now, STCP performs better than HSTCP but both perform worse than Vegas. The behavior of HSTCP is expected as this version performs like TCP SACK under small congestion window values. In Figure 4.5, which presents the throughput of the modified versions

of Vegas and Scalable TCP, it can be seen that the modified Vegas beats STCP and all other TCP versions especially in the case of high channel error rates.

4.3 Fairness analysis

Another interesting aspect not analyzed so far is the fairness characteristics of these new TCP versions. In particular, it is important to know (1) if these new TCP versions are fair to themselves, and (2) if they present the same unfairness problems that ”old” TCP versions present when similar connections have different RTTs.

Figures 4.6 and 4.7 present the fairness simulation results to answer the two main questions presented above. Figure 4.6 presents the results for the new TCP versions. The graph plots in the y-axis the throughput achieved by two connections of the same class sharing a 10Mbps bottleneck link channel. Experiments are repeated keeping the RTT of one connection constant and varying the RTT of the second connection. This is represented in the x-axis where the numbers 2, 3, 4, 5 and 6, mean that the second connection has an RTT that is two, three, four, five and six times longer than the first one. As it can be seen, the new proposals are fair to themselves when the connections have the same RTTs (when the fairness factor is equal to one). However, they present severe fairness problems otherwise with the shorter connection obtaining most of the bandwidth at the expense of the longer one. It can be seen that HSTCP and STCP behave fairly similar presenting severe unfairness against the longest connections. According to Figure 4.7, TCP Westwood, Newreno and Vegas also present fairness problems although less pronounced. In fact, Vegas is shown to be the less unfair of all. The fairness behavior of current TCP versions such as Tahoe and Reno is not plotted here because the results are well known.

In summary, it can be concluded that newer TCP versions present a more severe unfairness problem than old TCP versions when connections with different RTTs share the same bottleneck link. These new versions increase their window sizes in a more aggressive manner in order to capture the huge amount of bandwidth that is supposed to be available in high bandwidth delay product networks. Therefore, the same new algorithms that were designed to deal with the scalability problems of TCP in these networks are the responsible for the more severe unfairness in current environments.

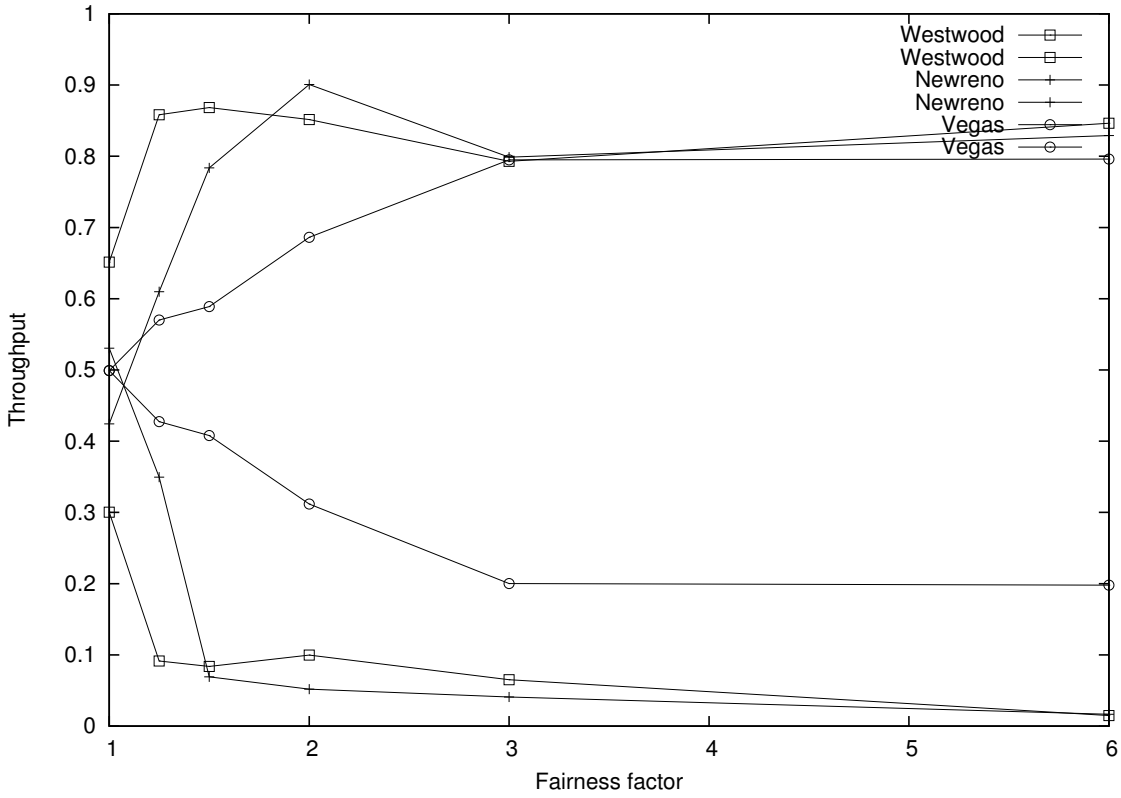

Figure 4.6 Fairness of HSTCP and STCP

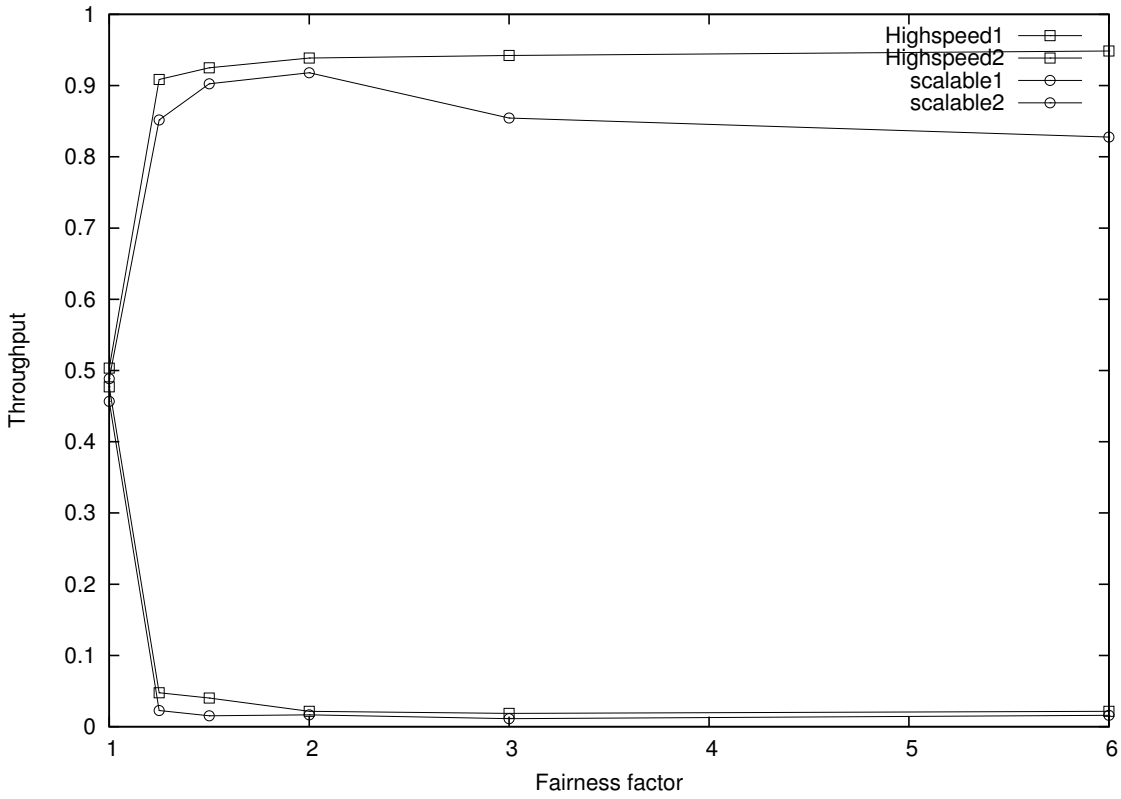

Figure 4.7 Fairness of the TCP Newreno, Westwood and Vegas

CHAPTER 5

CONCLUSIONS

This thesis evaluates the performance of existing congestion control algorithms in heterogeneous environments. Using simulations, it is shown that the window-based congestion control mechanism of current TCP versions doesn't scale well as the bandwidth-delay product of the connection increases. TCP reduces its congestion window very fast and drastically in the presence of congestion while it increases it very slowly during the congestion avoidance phase. On the other hand, for this new environment, bandwidth estimation techniques such as the ones utilized by TCP Vegas and Westwood can be very beneficial as they avoid packet drops considerably and can make full utilization of the available bandwidth during congestion avoidance. The thesis also shows that HighSpeed TCP and Scalable TCP perform better than normal TCP versions but still worse than TCP vegas. The fundamental reason for this is that they all continue to increase and decrease the congestion window variable in kind of a blindly manner. After analyzing the TCP's congestion control mechanism, simple modifications are introduced in TCP Vegas and STCP to improve their performance. The fairness of the new TCP proposals is also evaluated. It is shown that these new versions, as the old ones, present the same unfairness problem when connections with different RTTs share the same bottleneck link but even worse. The same algorithms that were included in these versions to make them better in HBDPC are the responsible for the worse fairness performance. The new proposals and the modified TCP versions are also evaluated in wireless networks. It is found that there is no compelling reason to switch to neither of these newer versions and that we should use TCP Vegas instead, which is the best performing version in all the scenarios considered.

REFERENCES

[1] "First International Workshop on Protocols for Fast Long-Distance Networks, PFLDnet 2003," *http://datatag.web.cern.ch/datatag/pfldnet2003/index.html*, February 2003.

[2] J. Bunn, J. Doyle, S. Low, H. Newman, and S. Yip, *Ultrascale Network Protocols for Computing and Science in the 21st Century. White paper to DoE's Ultrascale Simulation for Science. Available at http://netlab.caltech.edu/FAST*, 2002.

[3] S. Floyd, "HighSpeed TCP for Large Congestion Windows," *IETF RFC 3649, Experimental*, December 2003.

[4] T. Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," in *Proceedings of First International Workshop on Protocols for Fast Long-Distance Networks*, February 2003. [Online]. Available: http://datatag.web.cern.ch/datatag/pfldnet2003/program.html.

[5] A. Kamra, V. Misra, and D. Towsley, "Achieving High Throughput in Low Multiplexed, High Bandwidth, High Delay Environments," in *Proceedings of First International Workshop on Protocols for Fast Long-Distance Networks*, February 2003. [Online]. Available: http://datatag.web.cern.ch/datatag/pfldnet2003/program.html.

[6] C. Jin, D. Wei, and S. Low, "FAST TCP for High-Speed Long-Distance Networks," *IETF Internet draft (draft-jwl-tcp-fast-01.txt)*, June 2003.

[7] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *Proceedings of ACM SIGCOMM*, August 2002, pp. 89–102.

[8] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control for Fast, Long Distance Networks," in *To appear in Proceedings of Infocom 2004*. [Online]. Available: Available at http://www.csc.ncsu.edu/faculty/rhee/p.html.

[9] V. Jacobson, "Berkeley TCP evolution from 4.3-tahoe to 4.3-reno," in *Proceedings of the 18th Internet Engineering Task Force*, August 1990.

[10] W. Stevens, *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*. IETF RFC 2001, 1997.

[11] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 2582," *http://www.ietf.org/rfc/rfc2582.txt*, April 1999.

[12] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *Computer Communication Review*, vol. 26, pp. 5–21, 1996.

[13] L. Brakmo and L. Peterson, "TCP Vegas: End to End Congestion Avoidance in a Global Internet," *IEEE Journal on Selected Areas in Communications*, vol. Vol.13, No.8, pp. 1465–1480, 1995.

[14] N. S. . (ns2), *http://www.isi.edu/nsnam/ns/*.

[15] C. Casetti, M. Gerla, S. Mascolo, M. Sansadidi, and R. Wang, "TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks," *Wireless Networks*, vol. 8, pp. 467–479, 2002.

[16] M. Handley, J. Pahdye, S. Floyd, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," in *Internet Draft (draft-ietf-tsvwg-tfrc-03.txt)*, May 2001.

[17] E. Souza and D. Agarwal, "A HighSpeed TCP Study: Characteristics and Deployment Issues," *LBNL Technical Report Number LBNL-53215. Available at: http://www-itg.lbl.gov/ evandro/hstcp/*.

[18] S. Floyd, "Limited Slow Start for TCP with Large Congestion Windows," *IETF Internet draft (draft-floyd-tcp-slowstart-01.txt*, August 2002.

[19] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," IETF RFC2581 ," APR 1999.

[20] S. Vangala and M. A. Labrador, "Performance of TCP over Wireless Networks with the Snoop Protocol," in *Proceedings of IEEE LCN*, November 2002, pp. 600–601.

[21] I. F. Akyildiz, G. Morabito, and S. Palazzo, "TCP-Peach: A New Congestion Control Scheme for Satellite IP Networks," *IEEE/ACM Transactions on Networking*, vol. Vol. 9, No. 3, pp. 307–321, 2001.

[22] M. Zorzi, A. Chockalingam, and R. Rao, "Throughput Analysis of Channels with Memory," *IEEE Journal on Selected Areas in Communications*, vol. Vol. 18, pp. 1289–1300, 2000.

[23] R. R. R. M. Zorzi and L. B. Milstein, "On the accuracy of a first-order Markov Model for data transmission on Fading Channels ," in *IEEE ICUPC'95*, New Jersey, USA., November 1995, pp. 211–215.

[24] A. Abouzeid, S. Roy, and M. Azizoglu, "Stochastic Modelling of TCP over Lossy Links," 2000, pp. 1724–1733.