A Comparison Between BECN-Capable TCP New-Reno and TCP Vegas

By

Ali Dabir, B.A.Sc.

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment
of the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical and Computer Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario
Canada, K1S 5B6

January 29, 2004

© 2004, Ali Dabir

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-89830-X

Our file Notre référence

ISBN: 0-612-89830-X

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this dissertation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de ce manuscrit.

While these forms may be included in the document page count, their removal does not represent any loss of content from the dissertation.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Today, the most important transport protocol in the Internet is the Transmission Control Protocol (TCP). Therefore, different implementations of it have been introduced. New-Reno and Vegas are two variants of TCP. Besides improving TCP, it is important to improve the performance of routers as well. Therefore, efforts have also been made to provide routers with active queue management and even enable them to inform the TCP senders about developing congestion (i.e. explicit congestion notification). One of the active queue management techniques is Random Early Detection (RED). Also, routers can use Backward Explicit Congestion Notification (BECN) to notify TCP senders of developing congestion.

In this research, the BECN option is enabled for TCP New-Reno flows and then the performance of this combination is compared with that of TCP Vegas. In all the cases, the bottleneck router is RED-capable. The comparisons are made for connections with equal and unequal Round Trip Times (RTTs), in environments that can be homogeneous or heterogeneous. Also, two different types of traffic, File Transport Protocol (FTP) and web are analyzed in the experiments. Finally, a modification is introduced to improve the fairness of New-Reno + BECN when long and short connections coexist.

Acknowledgements

I would like to thank my supervisor, Professor I.E. Lambadaris, for his consistent guidance and encouragement during the course of my thesis.

Also, I would like to express my gratitude to my friend, Rupinder Makkar, for his valuable suggestions throughout my research.

To My Parents

Table of Contents

Acceptance Sheet.....	ii
Abstract.....	iii
Acknowledgements	iv
Table of contents	vi
List of Tables	ix
List of Figures.....	x
List of Acronyms	xiii
Chapter 1: Introduction	1
1.1 Motivation.....	1
1.2 Thesis Contributions	2
1.3 Thesis Organization	3
Chapter 2: Background.....	4
2.1 TCP Operation.....	4
2.2 TCP Tahoe	10
2.2.1 Slow Start and Congestion Avoidance.....	10
2.2.2 Fast Retransmit	12
2.3 TCP Reno.....	13
2.4 TCP Vegas	15
2.4.1 Modified Congestion Avoidance Mechanism.....	16
2.4.2 Modified Slow-Start Mechanism	19
2.4.3 Modified Retransmission Mechanism.....	20
2.5 Previous Comparisons of TCP Reno with TCP Vegas.....	22
2.6 TCP New-Reno.....	24

2.7 Random Early Detection (RED)	24
2.8 Explicit Congestion Notification (ECN).....	27
2.9 Backward Explicit Congestion Notification (BECN).....	30
Chapter 3: Simulation Guidelines	33
3.1 Topology.....	33
3.2 Simulation Randomization and Test Cases	35
Chapter 4: Simulations and Analysis for Equal-RTT FTP Connections	38
4.1 FTP Traffic with Light Web Traffic in a Homogeneous Environment	38
4.2 FTP Traffic without Web Traffic in a Heterogeneous Environment.....	49
4.3 FTP Traffic with Heavy Web Traffic in a Homogeneous/Heterogeneous Environment.....	52
Chapter 5: Simulations and Analysis for unequal-RTT FTP Connections.....	63
5.1 Before Improving the Fairness of New-Reno + BECN.....	65
5.2 After Improving the Fairness of New-Reno + BECN	71
5.3 Analyzing the Improvement of New-Reno + BECN Fairness	73
Chapter 6: Conclusions / Future Research.....	75
6.1 A Set of Equal-RTT FTP Connections in a Homogeneous Environment	75
6.2 A Set of Equal-RTT FTP Connections in a Heterogeneous Environment	76
6.3 A Set of Equal-RTT FTP Connections in a Homogeneous/Heterogeneous Environment with Heavy Web Traffic	77
6.4 A Set of Unequal-RTT FTP Connections in a Homogeneous Environment.....	78
6.5 Summary of Conclusions.....	78
6.6 Future Research	79

References	81
-------------------------	-----------

List of Tables

Table 1: Parameter Definitions in RED	26
Table 2: Light Web Traffic Parameters	39
Table 3: Heavy Web Traffic Parameters	53

List of Figures

Figure 1: Evolution of <i>cwnd</i> for TCP Reno	15
Figure 2: Window Control of TCP Vegas during Congestion Avoidance	18
Figure 3: Dropping Probability in RED.....	27
Figure 4: Explicit Congestion Notification (ECN)	28
Figure 5: The Simulation Topology for Equal-RTT FTP Connections.....	34
Figure 6: Average Queue Size for 10 Equal-RTT Connections (Homogeneous Environment: New-Reno + BECN).....	41
Figure 7: Average Queue Size for 15 Equal-RTT Connections (Homogeneous Environment: Vegas).....	41
Figure 8: Average Queue Size for 20 Equal-RTT Connections (Homogeneous Environment: New-Reno + BECN).....	42
Figure 9: Average Queue Size for 20 Equal-RTT Connections (Homogeneous Environment: Vegas).....	42
Figure 10: Bottleneck Link Output for FTP with Light Web Traffic (Homogeneous Environment: Equal-RTT).....	45
Figure 11: Fairness Index for FTP with Light Web Traffic (Homogeneous Environment: Equal-RTT).....	46
Figure 12: Drop Rate for FTP with Light Web Traffic (Homogeneous Environment: Equal-RTT).....	46
Figure 13: Delay for FTP with Light Web Traffic (Homogeneous Environment: Equal-RTT).....	47
Figure 14: Bottleneck Link Utilization for FTP without Web Traffic (Heterogeneous Environment)	50
Figure 15: Drop Rate for FTP without Web Traffic (Heterogeneous Environment)	50

Figure 16: Average Queue Size for 5 FTP Connections with Heavy Web Traffic (Vegas FTP with Vegas Web)	54
Figure 17: Average Queue Size for 5 FTP Connections with Heavy Web Traffic (Vegas FTP with New-Reno + BECN Web)	55
Figure 18: Bottleneck Link Utilization for FTP with Heavy Web Traffic (Homogeneous Environment)	56
Figure 19: Bottleneck Link Utilization for FTP with Heavy Web Traffic (Heterogeneous Environment)	56
Figure 20: Drop Rate for FTP with Heavy Web Traffic (Homogeneous Environment)	57
Figure 21: Drop Rate for FTP with Heavy Web Traffic (Heterogeneous Environment)	57
Figure 22: Number of Dropped HTTP Packets for Heavy Web Traffic with FTP Traffic (Homogeneous Environment)	58
Figure 23: Number of Dropped HTTP Packets for Heavy Web Traffic with FTP Traffic (Heterogeneous Environment)	58
Figure 24: Average Object Transfer Delay for Heavy Web Traffic with FTP Traffic (Homogeneous Environment)	59
Figure 25: Average Object Transfer Delay for Heavy Web Traffic with FTP Traffic (Heterogeneous Environment)	59
Figure 26: The Simulation Topology for Unequal-RTT FTP Connections	64
Figure 27: Bottleneck Link Utilization for FTP with Light Web Traffic (Homogeneous Environment: Unequal-RTT)	65
Figure 28: Fairness Index for FTP with Light Web Traffic Before Improvement (Homogeneous Environment: Unequal-RTT)	66
Figure 29: Drop Rate for FTP with Light Web Traffic (Homogeneous Environment: Unequal-RTT)	66
Figure 30: Delay for FTP with Light Web Traffic (Homogeneous Environment: Unequal-RTT)	67

Figure 31: Throughputs of a Pair of Long and Short Connections (New-Reno + BECN, Before the Improvement)	69
Figure 32: Throughputs of a Pair of Long and Short Connections (New-Reno).....	70
Figure 33: Throughputs of a Pair of Long and Short Connections (Vegas).....	70
Figure 34: Fairness Index for FTP with Light Web Traffic After Improvement (Homogeneous Environment: Unequal-RTT)	72
Figure 35: Throughputs of a Pair of Long and Short Connections (New-Reno + BECN, After the Improvement).....	74

List of Acronyms

ACK	Acknowledgement
AQM	Active Queue Management
BECN	Backward Explicit Congestion Notification
CE	Congestion Experienced
cwnd	Congestion window
CWR	Congestion Window Reduced
DRR	Deficit Round-Robin
ECN	Explicit Congestion Notification
ECT	ECN-Capable Transport
FAACK	Forward Acknowledgement
FTP	File Transport Protocol
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message protocol
IP	Internet Protocol
ISQ	ICMP Source Quench
IW	Initial Window
LW	Loss Window
MECN	Multiple-level BECN
MTU	Maximum Transfer Unit
MSS	Maximum Segment Size
ns	Network simulator

RED	Random Early Detection
RMSS	Receiver Maximum Segment Size
RTT	Round Trip Time
RW	Restart Window
rwnd	Receiver window
SACK	Selective Acknowledgement
SMSS	Sender Maximum Segment Size
ssthresh	Slow start threshold
TCP	Transmission Control Protocol
TOS	Type of Service
UDP	User Datagram Protocol

Chapter 1: Introduction

1.1 Motivation

The Internet relies heavily on TCP to transfer data reliably and cope with congestion. Therefore, any improvement of this protocol is of considerable importance. The introduction of various improvements to TCP has resulted in the creation of different variants of this protocol. The use of each variant carries certain advantages and drawbacks.

Reno and Vegas are two types of TCP that take two different approaches in controlling congestion. While TCP Reno relies on packet drops as a sign of congestion, TCP Vegas uses round trip times to detect the congestion. This fundamental difference between these two approaches causes each variant to behave differently under similar circumstances.

A number of previous studies have compared Reno with Vegas. But, the comparison can be extended by considering a multitude of new test cases. Also, in addition to looking at a large number of test cases, we can make these test cases more sophisticated by using active queue management and explicit congestion notification techniques.

Finally, we can use an enhanced version of Reno (i.e. New-Reno) to make the comparison more valuable. While TCP New-Reno preserves all the characteristics of TCP Reno, it actually presents a superior performance to TCP Reno due to a minor modification in its algorithm. Our study, thus, attempts to take this comparison one step further by addressing these issues.

1.2 Thesis Contributions

This thesis offers three main contributions:

- i) TCP Reno has been compared with TCP Vegas in a limited form in a number of previous works [6] [7] [11]. Here, we extend this study by using TCP New-Reno instead of TCP Reno, examining a variety of new test cases, and using a RED-capable bottleneck router. Also, we consider the effect of both long-lived FTP-traffic sources as well as short-lived web-traffic sources on our comparison. The test cases comprise connections that can have equal or unequal RTTs, and environments that can be homogeneous or heterogeneous.
- ii) In addition to comparing TCP New-Reno with TCP Vegas, we also compare BECN-capable TCP New-Reno with TCP Vegas. In [17], the advantages of using BECN together with TCP New-Reno have been demonstrated. Therefore, we concluded that it would be interesting to compare this improved performance with the performance of TCP Vegas.

iii) The simulations here show that the fairness of BECN-capable TCP New-Reno is not that high when it comes to environments where long-RTT and short-RTT connections compete with one another. So, we propose a minor modification in the behavior of BECN-capable TCP New-Reno sources that can improve the fairness.

1.3 Thesis Organization

This thesis is organized as follows. Chapter 2 introduces TCP, RED, ECN and BECN. Chapter 3 explains the basic simulation topology and parameters, as well as the techniques used to create randomization in the simulations. Chapter 4 comprises test cases where equal-RTT connections exist in homogeneous/heterogeneous environments. In such environments, besides FTP traffic, web traffic might have light presence, heavy presence or no presence at all. Chapter 5 looks at test cases where unequal-RTT connections exist in homogeneous environments. There, light web traffic coexists with FTP traffic. Finally, Chapter 6 gives conclusions and recommendations for future research.

Chapter 2: Background

In packet-switched computer communication networks, TCP acts as a reliable host-to-host protocol. TCP belongs to layer 4 of the OSI stack and operates above the Internet Protocol (IP). More specifically, as a connection-oriented and end-to-end reliable protocol, TCP can create reliable communication between pairs of processes in host computers that are attached to interconnected computer networks. Regarding the lower level protocols, TCP assumes it can obtain a simple and potentially unreliable datagram service from them [1].

2.1 TCP Operation

Generally, the operation of TCP can be categorized into 6 groups: Basic Data Transfer, Reliability, Flow Control, Multiplexing, Connections and Precedence and Security. Each of these groups touches a certain aspect of the operation of TCP [1]:

Basic Data Transfer

TCP packages octets into segments in order to transfer a continuous stream of octets in each direction between its users across the network. TCP also decides when to send or block data. Full-duplex operation is one of the important features of TCP.

Reliability

TCP assigns a sequence number to and demands a positive acknowledgement (ACK) for each transmitted segment. This is how TCP recovers from duplicated or lost data as well as data that is delivered out of order.

Data is retransmitted if an ACK is not received within a timeout interval. Also, the receiver uses sequence numbers to eliminate duplicates and to order segments that are received out of order. Furthermore, TCP assigns a checksum to each transmitted segment. By checking the checksum, receiver looks for damage and discards damaged segments.

Flow Control

The receiver returns a window with every sent ACK to tell the sender how much data it can send. This window indicates a range of acceptable sequence numbers beyond the last successfully received segment. This is how the receiver controls the amount of data the sender can send.

Multiplexing

Within each host, TCP provides a set of addresses (or ports). This allows simultaneous

use of TCP communication facilities by multiple processes running on a single host. Concatenating these addresses with network and host addresses then forms a socket. Thus, a pair of sockets is enough to uniquely identify each connection. In fact, multiple connections can use a socket simultaneously. It is a good practice to attach frequently used processes to fixed sockets and make them known to the public.

Connections

In order for two processes to communicate, a connection should first be established between their respective TCPs. Namely, certain status information must be initialized on each host. This status information contains sockets, sequence numbers and window sizes. After the completion of communication, the connection is terminated to free the resources.

In order to avoid the erroneous initialization of connections due to unreliable hosts and unreliable Internet communication system, a handshake mechanism with clock-based sequence numbers is used.

Precedence and Security

TCP provides the users with the ability to indicate the precedence and security of their communication. If these features are not demanded, default values are utilized.

In short, TCP compensates for IP's deficiencies by adding considerable functionality to it. From these functionalities, the most important ones are: Streams, Reliable Delivery, Network Adaptation and Flow Control.

Reliable delivery and flow control were just explained above. Streams refers to the fact that just like a file, TCP data is organized as a stream of bytes. This hides the datagram nature of the network. Network Adaptation refers to TCP's ability in maximizing throughput without overloading the network. TCP achieves this by dynamically learning about the delay characteristics of the network and adjusting its operation accordingly [2].

The following basic terms are associated with TCP [3].

SEGMENT

SEGMENT refers to any TCP/IP data or an acknowledgment packet (or both of them).

SENDER MAXIMUM SEGMENT SIZE (SMSS)

SMSS refers to the size of the largest segment that the sender can transmit. TCP/IP headers and options are not included in this size. *SMSS* can be based on a number of factors such as: the maximum transmission unit of the network, the path Maximum Transfer Unit (MTU) discovery algorithm and the *RMSS* (see below).

RECEIVER MAXIMUM SEGMENT SIZE (*RMSS*)

RMSS refers to the size of the largest segment that the receiver can accept. During connection startup, receiver specifies this value in the Maximum Segment Size (MSS) option it sends. Again, TCP/IP headers and options are not included in this size.

FULL-SIZED SEGMENT

This refers to a segment containing the maximum number of data bytes allowed. More specifically, this segment contains *SMSS* bytes of data.

RECEIVER WINDOW (*rwnd*)

The most recently advertised receiver window is entitled *rwnd*.

CONGESTION WINDOW (*cwnd*)

The amount of data TCP can send is limited by this TCP state variable. The minimum of *cwnd* and *rwnd* indicates how much data TCP can send on the next step.

INITIAL WINDOW (IW)

After the three-way handshake is completed, the size of the sender's congestion window

is referred to as the Initial Window.

LOSS WINDOW (LW)

The TCP sender uses the retransmission timer to detect loss. In essence, TCP sets a retransmission timer for every segment it transmits. The value of this timer is equal to the timeout interval for that segment. If the ACK to that segment is not received within this interval, TCP assumes that the segment has been lost and retransmits it. After the detection of loss, the size of the congestion window of the TCP sender is denoted by the Loss Window.

RESTART WINDOW (RW)

When a TCP restarts transmission after an idle period, the size of its congestion window is denoted by the Restart Window.

FLIGHT SIZE

This refers to the amount of data that has been transmitted, but not yet acknowledged.

Since TCP has been around for a long time, different implementations of TCP have been introduced; main examples of these implementations include TCP Tahoe, TCP Reno, TCP New-Reno, TCP Selective Acknowledgement (SACK), TCP Forward

Acknowledgement (FACK) and TCP Vegas. This thesis focuses on TCP New-Reno and TCP Vegas.

2.2 TCP Tahoe

Among the different implementations of TCP mentioned above, the most basic one is TCP Tahoe. This implementation is characterized by three algorithms: Slow Start, Congestion Avoidance and Fast Retransmit. In comparison with early implementations of TCP that used a go-back-n model, the three algorithms of TCP Tahoe result in higher throughput, while better controlling the network congestion [4]. Other implementations either add additional algorithms to these three algorithms (e.g. TCP Reno) or modify these three algorithms (e.g. TCP Vegas).

2.2.1 Slow Start and Congestion Avoidance

To control the amount of traffic being sent into the network, the slow start algorithm is used together with the congestion avoidance algorithm. In order to determine which one of these two algorithms must be used at a given time, a state variable called slow start threshold (*ssthresh*) is needed. Even though the initial value of *ssthresh* can be high, congestion can cause this value to be reduced. If $cwnd < ssthresh$, slow start is used. Otherwise, if $cwnd > ssthresh$, congestion avoidance must be utilized. In the cases where $cwnd$ is equal to *ssthresh*, the sender can use either slow start or congestion avoidance.

When TCP faces a network with unknown conditions, it has to somehow probe the network for bandwidth. If this probing happens at a slow rate, it will take a long time to arrive at an estimate of the available bandwidth. On the other hand, if the speed of probing is unreasonably fast, it can lead to a large number of packet drops. The slow start algorithm has thus been generally designed to take the middle line, although its exponential growth can still be too fast for some networks. Also, the IW should be less than or equal to $2 * SMSS$ bytes and should not be greater than 2 segments [3].

Throughout slow start, in every RTT the sender increases its *cwnd* by at most *SMSS* bytes per each received ACK (that acknowledges new data). This effectively means that *cwnd* is doubled every round trip time (i.e. exponential growth of *cwnd* over time). Slow start is exited when either the value of *cwnd* exceeds *ssthresh* or congestion is encountered.

In congestion avoidance, the sender increases its *cwnd* by 1 full-sized segment every RTT until congestion is reached (linear growth of *cwnd* over time). This is because during congestion avoidance, every received ACK in an RTT causes the *cwnd* to grow according to the following formula:

$$cwnd += SMSS * SMSS / cwnd$$

This formula provides an approximation for increasing *cwnd* by one full-sized segment per RTT, and is invoked upon the receipt of a non-duplicate ACK every RTT.

In the case of packet loss, the retransmission timer expires. This leads to timeout and the retransmission of the dropped segment. Upon the occurrence of timeout, the sender sets its *cwnd* to no more than LW (which is equal to one full-sized segment) regardless of the value of IW. Also, it sets the *ssthresh* to $\max(\text{FlightSize} / 2, 2 * SMSS)$. Then, over the next RTTs, the sender uses slow start to increase its *cwnd* from one full-sized segment to the new value of *ssthresh*. When *cwnd* reaches this value, slow start ends and congestion avoidance begins.

2.2.2 Fast Retransmit

When a TCP receiver receives a segment with a specified sequence number, it puts the sequence number of the next expected segment in the ACK it sends. Thus, while awaiting a segment with a specified sequence number, the receipt of segments with sequence numbers above that sequence number triggers the generation of duplicate ACKs. Practically, duplicate ACKs are generated when segments are dropped, re-ordered or replicated by the network [3].

When a loss occurs, the TCP sender must resort to fast retransmit. As a convention, the receipt of 3 duplicate ACKs is a sign of segment loss. So, after receiving 3 duplicate ACKs, the sender retransmits the missing segment without waiting for the retransmission timer to expire; this is called fast retransmit. Upon receiving the ACK to this retransmitted packet, the sender starts in the slow start phase.

2.3 TCP Reno

One of the shortcomings of TCP Tahoe is that after fast retransmit, the sender starts in slow start. This means that the pipe goes empty upon detecting a loss. If this is prevented, however, the throughput increases. To achieve this, an algorithm entitled fast recovery should be added to the three algorithms of TCP Tahoe. These four algorithms then constitute TCP Reno [4].

Whenever the TCP receiver generates an ACK, whether duplicate or non-duplicate, it is due to a received segment. In other words, when the TCP sender receives an ACK, it means that one of its sent segments has left the pipe. This is regardless of whether or not that segment was the one the receiver was expecting. This fact can be used to construct the fast recovery phase, which removes the need of going back to slow start upon detecting a loss.

The joint implementation of fast retransmit and fast recovery algorithms is as follows [3]:

#1) Upon the receipt of the third duplicate ACK, set *ssthresh* to no more than

$\max(\text{FlightSize} / 2, 2 * \text{SMSS})$.

#2) Retransmit the segment that has been lost. Then, in order to artificially inflate the *cwnd*, set *cwnd* to *ssthresh* plus $3 * \text{SMSS}$. This shows the number of segments (3) that have left the pipe (i.e. have been buffered by the receiver).

#3) Continue the artificial inflation of *cwnd* for every additional duplicate ACK received.

Each of these duplicate ACKs represent a segment that has left the pipe.

#4) If permitted by the receiver's advertised window and the new value of *cwnd*, transmit a segment.

#5) One RTT later, deflate the window upon the arrival of the ACK that acknowledges new data. To deflate the window, *cwnd* should be set to *ssthresh*. This ACK is in response to the retransmission done on the previous RTT. Also, this ACK should acknowledge all the segments transmitted between the time of sending of the lost segment and the time of receipt of the third duplicate ACK, given that none of these segments have been lost.

In short, during fast recovery, the incoming duplicate ACKs are used to clock the outgoing new data.

According to [4], TCP Reno can recover from the case where one packet is dropped from a window of data. It can also inefficiently recover from 2 (and rarely 3) packet drops by reducing its window more than once in succession. But, 4 or more packet drops from a window of data leads to timeout.

Figure 1, which is based on a diagram from [5], illustrates the evolution of *cwnd* for TCP Reno:

Figure 1: Evolution of *cwnd* for TCP Reno

2.4 TCP Vegas

Vegas is another variant of TCP that has its own unique slow start, congestion avoidance and retransmission mechanisms. While TCP Tahoe and TCP Reno use packet drops as a sign of congestion, the algorithms of TCP Vegas use the round trip times of packets to detect the congestion. This bandwidth estimation technique of Vegas results in better performance when it comes to throughput, drop rate and fairness [6] [7] [11].

2.4.1 Modified Congestion Avoidance Mechanism

The specialty of TCP Vegas is its advanced bandwidth estimation scheme. In Vegas, the difference between the expected and the actual throughputs is used to get an estimate of the available bandwidth in the network. If the network is not congested, the actual flow rate is either equal to the expected flow rate or slightly less than that. As the network starts to get congested, however, the amount of this difference increases. In Vegas, this difference is used in estimating the network's congestion level. Then, the congestion window is updated accordingly. Now, let's take a look at the following equation [6]:

$$Diff = (Expected - Actual) BaseRTT$$

The above equation basically converts the difference in throughputs into a difference between the window size and the number of acknowledged packets during the round trip time. Here, *Expected*, *Actual* and *BaseRTT* refer to the expected rate, the actual rate and the minimum round trip time (usually the RTT of the first segment sent by the connection [7]), respectively.

The algorithm on the next page explains in more detail the congestion avoidance mechanism in Vegas [6]:

- #1) The source finds the expected throughput as $Expected = cwnd / BaseRTT$, where $cwnd$ is the current window size and $BaseRTT$ is the minimum round trip time.
- #2) The source estimates the actual throughput as $Actual = cwnd / RTT$, where $cwnd$ is the current window size and RTT is the actual round trip time of a packet.
- #3) After finding the expected and the actual throughputs, the source estimates the backlog in the queue as $Diff = (Expected - Actual) BaseRTT$.
- #4) Upon finding $Diff$, the source updates its window size as follows:
- a) If $Diff < \alpha$, $cwnd$ is increased by 1.
 - b) If $Diff > \beta$, $cwnd$ is decreased by 1.
 - c) Otherwise, $cwnd$ remains unchanged.

The figure on the next page shows the congestion avoidance stage of TCP Vegas in action [6]. Vegas always tries to keep the backlog in the queue between α and β .

Figure 2: Window Control of TCP Vegas during Congestion Avoidance

To see the behavior of TCP Vegas from the above figure, let's contemplate a network with one connection and one link with capacity C . Also, let's represent the minimum round trip delay by $BaseRTT$. When $window < C * BaseRTT$, the throughput of this connection is $window / BaseRTT$. In the above figure, W represents the window size at which $window = C * BaseRTT$. If $window > W$, it results in having

$(Expected - Actual) > 0$ and the buildup of a queue. If $window < W + \alpha$, TCP Vegas increases the window size by one during the next round trip time. If $window > W + \beta$, TCP Vegas decreases the window size by one during the next round trip time. Otherwise, the window size is left unchanged. In the figure, the user i 's backlogged queue size in the link is represented by $Diff$.

TCP Vegas tries to detect and use the extra bandwidth whenever it is available without congesting the network. Thus, Vegas attempts to keep at least α but no more than β packets in the queue. In fact, in the case of having only one connection, the window size of TCP Vegas converges to a point that lies between $W + \alpha$ and $W + \beta$ [6].

Typical values for α and β are one and three, respectively [7].

It is important to know that prior to Vegas, a number of other approaches have also been proposed that use RTT to realize if the network is approaching congestion [8] [9] [10].

2.4.2 Modified Slow-Start Mechanism

During the slow-start period, the congestion window grows exponentially. Upon reaching the threshold window, slow-start ends and congestion avoidance begins. After the occurrence of a retransmit timeout, the Reno sender sets the threshold window to one half of the congestion window. Therefore, when the sender restarts sending data, it begins in slow-start phase, reaches the threshold window and then enters the congestion avoidance phase. This helps in preventing losses because the exponential growth turns into linear growth at half the previous value.

At the beginning of a connection, however, it is not obvious what the proper value for the threshold window is. If the initial value of the threshold window is too low, it becomes difficult to reach an optimal congestion window size since the exponential growth ends too quickly. On the other hand, if this value is too high, losses can be considerable since

the rapid growth of the congestion window can cause it to exceed the available bandwidth significantly.

In order to face this challenge, Vegas constitutes its slow-start by combining exponential growth with an aspect of its congestion avoidance scheme. Therefore, Vegas takes two actions during the slow-start phase. First of all, it allows the exponential growth only every other RTT. Secondly, in the RTTs where the window is fixed, it compares the expected rate with the actual rate. If the actual rate is below the expected rate by an equivalent of one router buffer, the Vegas sender leaves slow-start and enters congestion avoidance.

The purpose of growing the window every other RTT is that in the RTTs where the window is fixed, the actual rate represents the bandwidth that the connection allows. This is because in those RTTs, the amount of data that the sender sends is in accordance with the amount of data that is being acknowledged [7].

2.4.3 Modified Retransmission Mechanism

One of the important characteristics of TCP Vegas is that it keeps a fine grain retransmission timer for every segment it transmits. In fact, for every transmitted segment, Vegas records a millisecond resolution timestamp. Thus, the Vegas sender can easily obtain the segment's RTT in millisecond after receiving the ACK for that segment and retrieving the corresponding timestamp. Then, Vegas sets the fine resolution timestamp interval as the average RTT plus four times its variance [11].

In Reno, upon the expiration of the coarse-grain timer or the receipt of three duplicate ACKs, the sender retransmits the oldest unacknowledged segment. In Vegas, however, the receipt of a single duplicate ACK can trigger retransmission if the corresponding segment's fine grain timer has expired.

The fine grain timestamps can be additionally helpful in reacting to multiple packet drops from a window of data. To achieve this, after receiving the first two ACKs that acknowledge previously unacknowledged data following a retransmission, Vegas checks the fine grain retransmission timers of the unacknowledged segments. Then, it simply retransmits the segment corresponding to the oldest of the expired timers.

One of the important disadvantages of TCP Reno is its shortcoming in dealing with multiple packet drops from a window of data. This is because the receipt of a partial ACK takes Reno out of fast recovery. So, while the first dropped segment causes Reno to use fast retransmission and reduce its congestion window, the second dropped segment invokes fast retransmission again and causes Reno to reduce its congestion window a second time. That is why the Reno sender must wait for a retransmit timeout if 4 or more segments are dropped from a window of data.

Vegas, however, avoids successive reduction of its congestion window in the case of multiple segment drops. This is achieved by comparing the original transmission timestamp of the retransmitted segment with the timestamp of the last window decrease.

So, if the window was decreased after this segment had been originally sent, it will not be decreased again.

If Vegas decides to reduce its congestion window, the window decreases only by a quarter of its size. Reno, on the other hand, uses a multiplicative factor of $\frac{1}{2}$ to reduce its window. This is the only aspect where Vegas appears to be more aggressive than Reno. But, Vegas defaults to Reno's factor if it has retransmitted this segment recently [11].

2.5 Previous Comparisons of TCP Reno with TCP

Vegas

In [7], Brakmo et al. present the key algorithms of TCP Vegas and implement this protocol under the University of Arizona's x-kernel framework [12]. Then, they compare the performance of TCP Vegas with that of TCP Reno, which is implemented by retrofitting the BSD implementation into the x-kernel. This comparison is made both across the Internet as well as by using a simulator. The simulator itself is built based on the x-kernel. In both types of experiments, Vegas achieves 37-71% better throughput than Reno, while retransmitting one-fifth to one-half as many bytes as retransmitted by Reno. In Internet experiments, the improvements introduced by Vegas become more pronounced as the transfer size decreases. Also, Vegas does not adversely affect latency, it does not suffer from stability problems and it is as fair as Reno [7].

In [11], Ahn et al. try to verify the results mentioned above. Their comparison is again made by sending actual data through the Internet as well as by doing simulations. Here, Vegas 0.8 is actually ported to SunOS 4.1.3. Also, the native Reno transmitters and receivers run the Net-BSD-1.0 operating system. Furthermore, to do the simulations, a WAN emulator is used.

The results confirm that Vegas transfers bytes more efficiently than Reno, and achieves 3-8% higher network throughput than Reno. In Vegas, there is a reduction in the number of retransmitted packets by a factor of 2 to 5. In addition, RTT averages and variances are shorter for Vegas, since it keeps less data in the network. In comparison to Tahoe and Reno, Vegas is more resilient to various acknowledgment algorithms. In cases where the destination uses Tahoe rather than Reno acknowledgment algorithm, Vegas presents a considerably better performance. In contrast to Brakmo's study, however, this improvement becomes modest when the receiver employs Reno acknowledgment algorithm. In head-to-head transfers, it is apparent that Reno steals bandwidth from Vegas. But, when the congestion is really heavy, the throughput of Vegas becomes higher than that of Reno. Finally, an improvement is noticed in Vegas speedup with hop count and path buffer capacity [11].

In [6], Mo et al. use both analysis and simulation to compare TCP Reno with TCP Vegas. The results show that TCP Vegas leads to a fair allocation of bandwidth. Thus, unlike TCP Reno, it does not suffer from delay bias. Since TCP Vegas does not use packet drops to estimate the available bandwidth, it achieves a better performance than TCP

Reno. However, when Reno and Vegas flows compete with each other, the aggressive nature of Reno takes bandwidth away from Vegas. But, as the size of the queue at the bottleneck link becomes smaller, the amount of this stolen bandwidth goes down [6].

2.6 TCP New-Reno

In each round trip time, the Reno sender can retransmit at most one dropped packet. When multiple packets are dropped from a window of data, the sender might have to wait for a retransmit timeout. The reason for this is that TCP Reno exits fast recovery upon the receipt of a partial ACK that acknowledges some but not all the packets outstanding at the beginning of fast recovery. If the sender remains in fast recovery until all these outstanding packets are acknowledged, it can considerably avoid the need to wait for a timeout. This idea is the cornerstone of TCP New-Reno. More precisely, the receipt of a partial ACK signifies that the packet immediately following the acknowledged packet is missing. So, while staying in fast recovery, New-Reno retransmits one lost packet per round trip time until all the missing packets prior to entering fast recovery are retransmitted [4].

2.7 Random Early Detection (RED)

When drop-tail queues are used, the sources have no way to determine and react accordingly against a congestion buildup. In other words, this type of queues does not

give the sources the ability to take action before congestion is created. So, some forms of Active Queue Management (AQM) can be of help. As a form of AQM, RED operates by randomly dropping packets from arbitrary flows in order to avoid congestion collapse [13].

RED is designed to work with TCP. The router implementing RED starts dropping packets sooner than necessary. The reason for this is that if the sender continues sending a large number of packets, the router's buffer soon reaches its maximum capacity and the router will have to drop a large number of packets. Therefore, by early dropping of a small number of packets, the router notifies the sender that it must decrease its congestion window earlier than it would normally do. Note that the "Early" part of the RED acronym also points to this dropping. Of course, if RED is not implemented, the sender only reduces its *cwnd* if it becomes aware of congestion in the network through experiencing timeout or other ways of detecting packet loss such as the receipt of duplicate ACKs.

In the RED implementation, an average queue length is calculated using a low-pass filter with an exponential weighted moving average. This captures the long-term network congestion. The instantaneous queue length, on the other hand, shows the temporary system loading. To write the formulas for calculating the average queue length, let's first define the relevant parameters [13]:

Parameter	Definition
w_q	queue weight
s	typical transmission time
q	current queue size
$time$	current time
q_{time}	start of the queue idle time
avg	average queue length
p_a	packet dropping probability
min_{th}	Minimum threshold for the queue
max_{th}	Maximum threshold for the queue
max_p	Probability associated with max_{th}

Table 1: Parameter Definitions in RED

Upon the arrival of a packet, if the queue is nonempty,

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

Otherwise,

$$avg = (1 - w_q)^{(time - q_{time})/s} avg$$

Upon the arrival of a new packet, RED calculates the new average queue size avg and compares it with the two RED queue length thresholds, min_{th} and max_{th} . If $avg > max_{th}$,

the packet is dropped with probability one. If $avg < min_{th}$, there is no need to take any action. Finally, If $min_{th} \leq avg \leq max_{th}$, the arrived packet is dropped with some probability (p_a), which depends on the average queue size.

The following figure shows the operation of RED:

Figure 3: Dropping Probability in RED

2.8 Explicit Congestion Notification (ECN)

In RED, dropping packets is used to inform the senders that they should reduce their congestion windows. However, instead of dropping packets, it is much better to mark them. The reason for this is that since packets carry data, dropping them means loss of data. In addition, if a packet is dropped, it should be retransmitted. Therefore, the information in that packet can reach the destination with long delay. These issues give rise to the idea of Explicit Congestion Control (ECN), which can be deployed on top of

RED. Namely, when the average queue size is between the two thresholds, the router marks the packets with the appropriate probability (instead of dropping them with that probability). Then, later on, the senders will notice this marking in the ACKs they receive for those marked packets. Thus, they know they should reduce their congestion windows. In cases where heavy congestion pushes the average queue size over max_{th} , however, packets will be dropped with probability one. Let's take a look at the following figure showing some packets being marked using ECN [14]:

Figure 4: Explicit Congestion Notification (ECN)

To activate ECN, certain settings are needed in IP and TCP headers [15]. In the IP header, a field with two bits should be allocated for ECN: the ECN-Capable Transport (ECT) bit and the Congestion Experienced (CE) bit. The sender of data uses the ECT bit to indicate that the connection end-points are ECN-capable. The router uses the CE bit in order to indicate congestion to the connection end nodes. In the IP header, bits 6 and 7 in

the Ipv4 Type of Service (TOS) octet constitute the ECN field. Bit 6 is actually the ECT bit, whereas bit 7 is the CE bit. In IPv6, the Traffic Class octet corresponds to the TOS octet in IPv4.

In the TCP header, two flags should be introduced: CWR (Congestion Window Reduced) and ECN-Echo. The ECN-Echo flag is used by the receiver to inform the sender of the receipt of CE packet. The CWR flag is used by the sender to inform the receiver about the reduction of the congestion window. Bits 8 and 9 in the Reserved field of the TCP header become CWR and ECN-Echo flags, respectively.

If the TCP sender sets the ECT bit of the packet it is sending out into the network, it has indicated that ECN is supported in the transport of this packet. This implies that, if necessary, the routers on the path can set the CE bit to mark this packet. When this packet is received at the receiver, by seeing bits ECT and CE set, the receiver understands that it should inform the sender to reduce its congestion window. Otherwise, the network can become congested. So, the receiver sets the ECN-Echo flag in the ACKs it is sending out. The receiver will continue to do this until it receives a packet from the sender with the CWR flag set. This would mean that the sender has just reduced its congestion window [15].

Note that if the sender does not set the ECT bit, it means that ECN does not apply to that packet. So, the CE bit is ignored. Also, there is currently no traffic reduction mechanism

for the receivers to react to congestion notification. This is why pure acknowledgement packets are sent with the ECT bit off [15].

2.9 Backward Explicit Congestion Notification (BECN)

The use of Backward Explicit Congestion Notification in an IP network refers to the generation of Internet Control Message Protocol (ICMP) [16] Source Quench (ISQ) messages by the router to inform the TCP source of developing congestion. One of the advantages of BECN is that it uses the existing network layer signaling for congestion notification. So, since it is independent of the transport layer, it can be used with both TCP and User Datagram Protocol (UDP). Also, while BECN and ECN both promote the idea of marking packets instead of dropping them, BECN provides faster congestion notification and better performance than ECN [17]. But, there have always been a number of concerns associated with using BECN. Among them, the most important ones have been the lack of a set of well-defined conditions under which the router should generate an ISQ message and the lack of a standard algorithm for the response of the TCP source to an ISQ message.

Recently, Akujobi et al. [17] have proposed a behavior for the BECN-capable router as well as for the BECN-capable TCP sender. They have also suggested New-Reno as the type of TCP with which the BECN option is enabled. Their work is an enhancement of the algorithm suggested in [18]. Also, it is the ns implementation of BECN by Frank

Akujobi that is used in the simulations of our research. The following algorithm indicates the behavior of the BECN-capable router [17]:

```
If (the arriving packet pushes the average queue size above
    the maximum RED threshold)
{
    check the IP header of the packet and drop the packet just like an ECN packet;
    if (the ECT bit is marked in the IP header)
    {
        send an ISQ message due to a dropped packet back to the source;
    }
} else if (the arriving packet pushes the average queue size between the minimum and
    the maximum RED thresholds){
    if (the RED rules select this packet and the ECT bit is set) and
    if (the packet is not already marked)
    {
        mark the packet (CE bit) and send an ISQ message due to a marked packet back
to the
        source;
    }
    else if (the RED rules select this packet and the ECT bit is not set)
        drop the packet;
}
```

The ISQ message due to a dropped packet is distinguished from the one due to a marked packet by using a single bit. Also, the BECN-capable router assumes that the TCP sender is BECN-capable if the ECT bit is marked in the IP header of the packet.

Since the BECN scheme does not involve the TCP receiver, no initial end-to-end negotiations are needed to establish the BECN capability. Upon receiving an ISQ message due to a marked packet, the TCP sender sets its congestion window and *ssthresh* to one-half of the current congestion window and waits one full RTT before increasing its window. If the ISQ message is due to a dropped packet, however, the TCP sender sets its congestion window and *ssthresh* to one-half of the current congestion window and follows the TCP congestion control algorithm. In every RTT, the TCP sender does not react to ISQ messages more than once [17].

Chapter 3: Simulation Guidelines

The software that is used to do the simulations is called the Network Simulator - ns-2 [19]. In this research, the performance of TCP Vegas is compared with that of TCP New-Reno as well as BECN-capable TCP New-Reno. The latter refers to TCP New-Reno flows for which the BECN option has been enabled. Generally, the comparisons are made by looking at the bottleneck link. In some cases, the queue at R1 (see Figure 5) is also monitored since this queue acts as the entrance to the bottleneck link.

In addition, homogeneous environments refer to those where only one type of TCP is present (i.e. Vegas, New-Reno or New-Reno + BECN), whereas heterogeneous environments are those where these TCP variants coexist.

3.1 Topology

Unless stated otherwise, the simulations are based on the topology shown on the next page:

In the section where we investigate the combination of long and short connections, however, we modify the delay of these links accordingly. The queue at R1, which acts as the entrance to the bottleneck link, has RED capability. This is the case regardless of the type of TCP variant present. The maximum size of the queue at R1 is set to 90 packets. Also, the minimum and maximum thresholds that the RED queue at R1 should use for marking / dropping packets are 15 and 45 packets, respectively. In addition, $max_p = 0.1$ and $w_q = 0.002$. These parameters have been selected similar to [17], and are based on the guidelines first indicated in [20].

The size of each packet is the default ns value of 1000 bytes, and the size of each ACK packet is 40 bytes. The maximum size of the congestion window any sender can reach is set to 100 packets.

3.2 Simulation Randomization and Test Cases

All the simulations are run for 600 seconds, unless stated otherwise. In each experiment, we take the necessary actions to randomize the simulations. This randomization is needed because all the graphs in this thesis are drawn with 95% confidence intervals. More specifically, since we run 30 replications for each experiment, changing the seed of randomization in each replication gives a different set of results. Then, from the results of these 30 independent replications, we form 6 groups each of which consists of the results of 5 replications. After this, we simply use the Student's t-distribution to draw the confidence interval.

The reason we use the Student's t-distribution instead of the Gaussian distribution is that the latter distribution demands having the results of 30 or more replications per group. In each group, the average of the results of 5 replications is computed. Therefore, these 6 groups yield 6 independent random variables indicated as $x_1, x_2, \dots, x_6$. Let's represent the mean and the variance estimates of these random variables as m and s^2 , respectively. In order to find m , we simply calculate the average of $x_1, x_2, \dots, x_6$. To find s^2 , we use the expression on the next page:

$$s^2 = \frac{1}{n-1} \sum_i^n (x_i - m)^2$$

$$= \frac{1}{n-1} (\sum_i^n x_i^2 - nm^2)$$

The standard deviation of the random variables, s , is simply the square root of s^2 .

In this case, the value of n is 6. Also, $n - 1$ is known as the degrees of freedom. Now, we define the quantity t as follows:

$$t = \frac{m - \mu}{\left(\frac{s}{\sqrt{n}}\right)}$$

The quantity t is distributed according to the Student's t-distribution. While this distribution is bell-shaped similar to the Gaussian distribution, it has fatter tails. In the definition of t , μ represents the true mean of the random variables. In order to find the 95% confidence interval, the value of k in the following expression should be such that [21]:

$$\text{prob} \left(-k < \frac{m - \mu}{\left(\frac{s}{\sqrt{n}}\right)} < k \right) = 0.95$$

According to the table, the value of k is 2.78. Our 95% confidence interval, thus, becomes:

$$m - k \frac{s}{\sqrt{n}} < \mu < m + k \frac{s}{\sqrt{n}}$$

The actual actions taken to randomize the simulations differ from one test case to the other, and are described for each test case in its own section.

Generally, we categorize all the simulation test cases into one of the two main groups: equal-RTT and unequal-RTT connections.

Chapter 4: Simulations and Analysis for

Equal –RTT FTP Connections

All the test cases in this section can be further categorized into three sub-sections:

FTP traffic with light web traffic in a homogeneous environment, FTP traffic without web traffic in a heterogeneous environment and FTP traffic with heavy web traffic in a homogeneous/heterogeneous environment.

Homogeneous environments are the ones where only one type of TCP is present (i.e. Vegas, New-Reno or New-Reno + BECN). Heterogeneous environments, on the other hand, are those where these TCP variants coexist.

4.1 FTP Traffic with Light Web Traffic in a

Homogeneous Environment

In this case, in a homogeneous environment, we send FTP traffic over either New-Reno + BECN, New-Reno or Vegas TCP connections. Here, the connections all have equal RTTs. To randomize each replication, the FTP flows end randomly between 500s and 600s. The starting time of all the FTP flows, however, is the same (0s). In Figure 5, it is assumed that FTP servers consistently transfer FTP data to FTP clients (S11 to D11, S12 to D12 and so on).

In this case, the focus is on the statistics related to FTP. So, while web traffic is needed in the background to provide more randomization, we make sure that the amount of web traffic is light. In Figure 5, web traffic interaction happens between web servers S1,..., S10 and web clients D1,..., D10. The parameters for light web traffic are indicated in Table 2. These parameters are a modification of the web traffic parameters in [17]. In this case, the first session is launched at 20s. Also, the web traffic is present through the entire 600 seconds of the simulation run. In web traffic, each session consists of a number of pages. Each page then contains a number of objects, and each object contains a number of packets. In every experiment, the type of TCP that carries the web traffic is the same as the one that carries the FTP traffic.

Parameter Value	Distribution	Mean	Shape
Intersession Time	Exponential	50s	-
Session Size	Constant	10 pages	-
Inter-Page Time	ParetoII	10s	2
Page Size	ParetoII	3 objects	1.5
Inter-Object Time	ParetoII	0.1s	1.5
Object Size	ParetoII	12 packets	1.2
Number of Sessions	Constant	12	-

Table 2: Light Web Traffic Parameters

In order to congest the network in different levels, we need to have different number of FTP connections. As Figures 6 and 8 indicate, 10 and 20 FTP connections carried over TCP New-Reno + BECN can cause mid-level and heavy congestion, respectively.

We also separately studied the levels of congestion in TCP New-Reno with respect to different numbers of FTP connections and found out that its pattern is the same as New-Reno + BECN. In the case of Vegas, Figures 7 and 9 illustrate that mid-level and heavy congestion are formed by 15 and 20 FTP connections, respectively.

As indicated in section 3.1, the minimum and maximum thresholds for the RED queue at R1 are 15 and 45. Thus, since in Figures 6 and 7 the average queue sizes fluctuate around 30, we conclude that in those cases the congestion is mid-level. In Figures 8 and 9, however, the average queue sizes fluctuate around 45 (the upper threshold). So, those cases represent heavy congestion.

Figure 6: Average Queue Size for 10 Equal-RTT Connections
(Homogeneous Environment: New-Reno + BECN)

Figure 7: Average Queue Size for 15 Equal-RTT Connections
(Homogeneous Environment: Vegas)

Figure 8: Average Queue Size for 20 Equal-RTT Connections
(Homogeneous Environment: New-Reno + BECN)

Figure 9: Average Queue Size for 20 Equal-RTT Connections
(Homogeneous Environment: Vegas)

To do the comparisons, the best area is that of mid-level congestion. This is because when having light congestion, there are too few packets to really show the performance differences of different TCP variants. On the other hand, during heavy congestion, the network is so much under pressure that it becomes difficult to distinguish between the behaviors of various TCP variants. Thus, when comparing TCP New-Reno (+ BECN) with Vegas in this section, it is best to look at 10 and 15 connections. When there are 10 connections, TCP New-Reno (+ BECN) is in mid-level congestion. But, TCP Vegas is still in light congestion, even though the congestion shows signs of getting heavier.

When there are 15 connections, TCP New-Reno has left mid-level congestion. But, it is not yet heavily congested. Vegas, however, is under mid-level congestion for 15 flows.

Here, for 4 different numbers of FTP connections, the bottleneck link output, drop rate, fairness index and delay are computed per TCP variant. Note that these statistics are computed for the FTP traffic. The web traffic only serves to provide more randomization. The results for the 3 TCP variants are then shown together for comparison.

In order to find the bottleneck link output in the case of 5 connections, for instance, first the total amount of traffic that has passed throughout the bottleneck link over the course of 600 seconds is measured. Then, this number is divided by 600 seconds in order to have the units of bits/sec.

In a real situation, there is no guarantee that the traffic flowing in the bottleneck link will reach its destination. In the simulation environment, however, the link error rate is set to zero. In addition, the bandwidth of the link going into R2 (i.e. the bottleneck link) is much lower than the bandwidth of all the links leaving R2. So, no packets are dropped at R2. Therefore, the traffic flowing in the bottleneck link reaches its destination without any problem.

Furthermore, only one copy of a packet ever passes through the bottleneck link. This is because ns is a packet-based simulation tool. Therefore, at R1, a packet is either dropped or passed to the bottleneck link. In the latter case, the packet will definitely reach its destination.

The Fairness Index shows how fair the available bandwidth has been distributed among the several present connections. Here, we use the Jain's Fairness Index [7]:

$$f(x_1, x_2, \dots, x_n) = \frac{\left(\sum_{i=1}^n x_i \right)^2}{n \sum_{i=1}^n x_i^2}$$

The x_i refers to the cumulative throughput obtained by connection i at the end of 600 seconds. The total number of connections is represented by n .

The drop rate refers to ratio of the dropped data at R1 to the total data received at R1 until the 600s.

Finally, the delay refers to the queuing delay that a packet across all connections has on average incurred.

Figure 10: Bottleneck Link Output for FTP with Light Web Traffic
(Homogeneous Environment: Equal-RTT)

Figure 11: Fairness Index for FTP with Light Web Traffic
(Homogeneous Environment: Equal-RTT)

Figure 12: Drop Rate for FTP with Light Web Traffic
(Homogeneous Environment: Equal-RTT)

Figure 13: Delay for FTP with Light Web Traffic
(Homogeneous Environment: Equal-RTT)

According to Figure 10, in a homogeneous environment, none of the three TCP variants has a superior performance compared to the others. This is because in any TCP variant, even five flows are enough to keep the bottleneck link busy most of the time.

Figure 11 shows that for 10 and 15 flows, Vegas is less fair than both New-Reno and New-Reno + BECN. This result is in agreement with the test case in [7] where the fairness of 2 and 4 equal-RTT connections is tested. But, a more appropriate test case for studying fairness is the one in which the connections do not have similar RTTs. In such a case, the inherent differences between Vegas and New-Reno in treating unequal-RTT connections become clear. This will be studied in chapter 5.

In Figure 12, the drop rates are demonstrated. Apparently, the New-Reno + BECN combination achieves the lowest drop rate. This is in conformance with the results in [17], and shows that BECN is very effective in preventing drops. Also, before reaching heavy congestion (5, 10 and 15 flows), it is visible that Vegas has a lower drop rate than New-Reno. This is in conformance with the findings in [7] and [11], and is chiefly due to the congestion avoidance mechanism of Vegas. This mechanism causes each connection to keep the number of its packets in the queue between the two thresholds.

Finally, Figure 13 shows the delays. Before reaching heavy congestion, Vegas has a lower delay than the others. This is again mainly due to the congestion avoidance mechanism of TCP Vegas, which tries not to overpopulate the queue. But, New-Reno (+ BECN) frequently overburdens the queue with packets since it relies on packet drops as a sign of congestion. Another observation in Figure 13 is that for 10 and 15 flows, New-Reno + BECN presents a higher delay than New-Reno. This is because New-Reno + BECN generally pushes more data into the queue since BECN prevents drops considerably.

4.2 FTP Traffic without Web Traffic in a Heterogeneous Environment

In this subsection, the three TCP variants are present in a heterogeneous environment simultaneously. Then, 3, 9, 15 and 21 FTP connections are used to congest the network in different levels. In order to create mid-level congestion, 9 connections should be used. The use of 21 connections then creates heavy congestion. Note that these numbers are all divisible by 3. This is because we want to have an equal number of connections from each TCP variant in the network. In this case, to create randomization, it is just enough to end the TCP flows randomly between 500 and 600 seconds. So, web traffic is not needed.

The figures on the next page show the bottleneck link utilization and the drop rate for different TCP variants. The FTP bottleneck link utilization is found through dividing the bottleneck link output by the bandwidth of the bottleneck link (1.5 Mbps).

Figure 14: Bottleneck Link Utilization for FTP without Web Traffic (Heterogeneous Environment)

Figure 15: Drop Rate for FTP without Web Traffic (Heterogeneous Environment)

In Figure 14, we observe that in the area of mild to mid-level congestion, Vegas has a lower throughput than the other two TCP variants. But, as congestion becomes heavier, Vegas consumes more and more of the bottleneck link's bandwidth. At 21 flows, Vegas gets more bandwidth than the other two variants.

In [6], the incompatibility of TCP Vegas with TCP Reno is shown both analytically and through simulation. In essence, the argument is that the aggressive nature of TCP Reno injects a large number of packets into the network and forces TCP Vegas to reduce its congestion window. Thus, in a heterogeneous environment, TCP Reno steals bandwidth away from TCP Vegas. This fact explains the behavior of TCP Vegas for 3 and 9 connections, which represent mild and mid-level congestion, respectively.

According to [11], in a heterogeneous environment, Vegas gets more bandwidth than Reno in the area of heavy congestion. This result can be generalized to New-Reno (with or without BECN). The reason for this phenomenon is that based on its nature, the number of multiple packet drops for Vegas is not as high as New-Reno. So, timeouts happen less for Vegas compared to New-Reno. In addition, upon a packet drop, Vegas reduces its congestion window by a quarter, whereas New-Reno reduces it by half. So, in heavy congestion, New-Reno backs off its congestion window more frequently and more radically than TCP Vegas. The radical reduction of New-Reno packets in the network also creates some extra bandwidth. Upon detecting this extra bandwidth, Vegas can increase its congestion window to take advantage of the situation. That is why the

throughput of Vegas is more than New-Reno + BECN and New-Reno during heavy congestion.

In Figure 15, New-Reno + BECN has the lowest drop rate as expected. But, the drop rates of New-Reno and Vegas are not distinguishable. This is because unlike the homogeneous environment, here Vegas shares the queue with New-Reno and New-Reno + BECN. So, the bandwidth estimation techniques of Vegas cannot completely control the backlog in the queue. In addition, the aggressive nature of New-Reno (+ BECN) frequently overpopulates the queue, forcing more Vegas packets to be dropped than in the homogeneous case. That is why we do not see the improvement in drop rate that Vegas achieves over New-Reno in homogeneous environments.

4.3 FTP Traffic with Heavy Web Traffic in a Homogeneous/Heterogeneous Environment

In this subsection, FTP traffic is sent over either New-Reno + BECN or Vegas TCP. But, together with FTP traffic, heavy web traffic is also sent over either New-Reno + BECN or Vegas TCP. So, in terms of the type of traffic and the type of TCP over which it travels, we can have four different combinations: New-Reno + BECN FTP with New-Reno + BECN web, Vegas FTP with Vegas web, New-Reno + BECN FTP with Vegas web and Vegas FTP with New-Reno + BECN web. Here, again nodes S1, ..., S10 and D1, ..., D10 are used for web traffic interaction. Also, all connections have equal RTTs.

The parameters of web traffic are designed such that in every simulation, it occupies about 50% of the bottleneck link's bandwidth. The heavy web traffic is accompanied by 5, 10, 15 or 20 FTP connections. Here, since heavy web traffic creates considerable burstiness, there is no need to end FTP flows randomly between 500 and 600 seconds. Therefore, both FTP and web traffic flow in the network until the simulation ends at 500 seconds. In Table 3, the parameters for generating heavy web traffic are represented. The first session is launched at 50s. In web traffic, each session consists of a number of pages. Each page then contains a number of objects, and each object contains a number of packets.

Parameter Value	Distribution	Mean	Shape
Inter-session Time	Exponential	2.75s	-
Session Size	Constant	10 pages	-
Inter-Page Time	ParetoII	10s	2
Page Size	ParetoII	3 objects	1.5
Inter-Object Time	ParetoII	0.1s	1.5
Object Size	ParetoII	12 packets	1.2
Number of Sessions	Constant	200	-

Table 3: Heavy Web Traffic Parameters

The generated web traffic using these parameters is quite overwhelming. In fact, even with 5 FTP connections running in the background, the network can become heavily

congested. In order to show this, we represent the average queue size for two cases when 5 background FTP connections are present: the case where Vegas FTP is accompanied by Vegas web traffic (Figure 16) and the case where Vegas FTP is accompanied by New-Reno + BECN web traffic (Figure 17). But, while the level of congestion is high even for 5 FTP connections, increasing the number of FTP connections may boost the FTP link utilization in some cases.

Figure 16: Average Queue Size for 5 FTP Connections with Heavy Web Traffic (Vegas FTP with Vegas Web)

Figure 17: Average Queue Size for 5 FTP Connections with Heavy Web Traffic
(Vegas FTP with New-Reno + BECN Web)

In this subsection, we specifically monitor four metrics: FTP bottleneck link utilization, FTP drop rate, number of dropped Hypertext Transfer Protocol (HTTP) packets and average web object transfer delay. The last metric is the average of the transfer delays of all the web objects sent during a simulation. According to [17], web object transfer delay is the interval between the time a web client makes an initial request (GET message) for a web object and the time the server receives the ACK to the last packet of that web object. The following figures show these metrics for different configurations. Note that we separate the results related to homogeneous environments from those related to heterogeneous environments for easier analysis.

Figure 18: Bottleneck Link Utilization for FTP with Heavy Web Traffic (Homogeneous Environment)

Figure 19: Bottleneck Link Utilization for FTP with Heavy Web Traffic (Heterogeneous Environment)

Figure 20: Drop Rate for FTP with Heavy Web Traffic (Homogeneous Environment)

Figure 21: Drop Rate for FTP with Heavy Web Traffic (Heterogeneous Environment)

Figure 22: Number of Dropped HTTP Packets for Heavy Web Traffic with FTP Traffic (Homogeneous Environment)

Figure 23: Number of Dropped HTTP Packets for Heavy Web Traffic with FTP Traffic (Heterogeneous Environment)

Figure 24: Average Object Transfer Delay for Heavy Web Traffic with FTP Traffic (Homogeneous Environment)

Figure 25: Average Object Transfer Delay for Heavy Web Traffic with FTP Traffic (Heterogeneous Environment)

The results shown on Figures 20, 21, 22 and 23 are as expected. This is because we know that BECN significantly brings down the number of dropped packets. So, when it is used with TCP New-Reno to carry either FTP or web traffic, it lowers the amount of drops considerably.

In Figures 18 and 19, we observe that whenever web traffic is running over TCP Vegas, it lowers the bottleneck link utilization for its corresponding FTP traffic. Supposedly, as the number of FTP connections increases, the share of FTP traffic from the link's bandwidth should go up.

But, in Figure 18, when 15 and 20 FTP connections are present, the web traffic running over Vegas prevents the FTP traffic running over Vegas to consume more bandwidth. On the contrary, when web traffic runs over New-Reno + BECN, the FTP traffic running over New-Reno + BECN increases its share of bandwidth.

In Figure 19, across all numbers of FTP connections, the web traffic running over Vegas does not let the FTP traffic running over New-Reno + BECN to occupy more bandwidth. These observations point out that Vegas does not decrease its rate when it carries heavy web traffic.

For further investigation, let's take a look at Figures 24 and 25. In Figure 24, we observe that for 15 and 20 connections, the average web object transfer delay for New-Reno +

BECN is lower than that of Vegas. This means that the queuing delay is higher in the case of Vegas.

In Figure 25, when there are 5 connections, it is obvious that TCP Vegas carrying heavy web traffic results in higher queuing delay. But, the average web object transfer delays for the two configurations converge for higher connections. This is because Vegas FTP accompanying New-Reno + BECN web also injects a relatively high amount of data during heavy congestion, as noted in section 4.2.

In short, the main new result concluded from the analysis of this section is that when carrying heavy web traffic, TCP Vegas does not slow down as it should. The principal cause of this phenomenon is an inherent problem in TCP Vegas as pointed out in [22], and summarized in [6]. This problem is entitled persistent congestion.

In essence, due to the central role played by *BaseRTT* in the algorithm of TCP Vegas, an overestimation of this parameter adversely affects the performance of Vegas. According to Table 3, heavy web traffic starts from 50s and flows until the end of the simulation.

This traffic consists of 200 sessions, each of which consists of a number of pages. Each page then contains a number of objects, and each object contains a number of packets.

In order to exchange data reliably, each object opens its own TCP connection.

When an object opens a TCP connection in the middle of the simulation, for instance, the network is congested. Therefore, the value of *BaseRTT* it obtains is set too high. This

value is found only once at the beginning of the connection. Theoretically, during heavy congestion, the difference between expected and actual flow rates becomes high. This, in turn, causes the sender to reduce its congestion window.

But, an overestimation of *BaseRTT* causes this difference to remain small, even if the congestion is heavy. So, contrary to what is desirable, the TCP sender starts to increase its congestion window under the assumption that it is keeping the backlog in the queue between the two thresholds. This situation happens for every web object during the simulation, and aggravates the congestion considerably.

Chapter 5: Simulations and Analysis for Unequal –RTT FTP Connections

In this case, in a homogeneous environment, we send FTP traffic over either New-Reno + BECN, New-Reno or Vegas TCP connections. To randomize each replication, the FTP flows end randomly between 500s and 600s. In addition, to create more randomization, light web traffic with parameters shown in Table 2 also flows through the network. In each experiment, FTP and its accompanying web traffic both run on the same type of TCP. Here, the connections do not have equal RTTs. Also, they have been selected such that their RTTs resemble real RTTs on the Internet. According to [23], 400 ms is a threshold RTT above which the connection can generally be considered long. So, Figure 26 depicts a realistic set of long and short connections.

In this figure, the minimum RTT of half of the connections is greater than 400 ms (484 ms). The other half have minimum RTTs lower than 400 ms. This group is further divided equally into two groups with different minimum RTTs: 184 ms and 164 ms

Figure 26: The Simulation Topology for Unequal-RTT FTP Connections

In order to congest the network at different levels, we use 8, 16, 24 and 32 connections.

We have verified through looking at the average queues that 16 and 32 connections create mid-level and heavy congestion for TCP New-Reno (+ BECN), respectively. In the case of Vegas, 24 connections cause mid-level congestion, while heavy congestion is reached if 32 connections are present. At 24 connections, New-Reno (+ BECN) has left mid-level congestion, but has not yet reached heavy congestion. At 16 connections, Vegas is still facing light congestion, while the congestion shows tendencies of becoming heavier.

The four metrics analyzed here are the bottleneck link utilization, fairness index, drop rate and queuing delay. In this section, we also introduce an improvement to increase the fairness of TCP New-Reno + BECN when it comes to unequal-RTT connections.

Then, we analyze this improvement.

5.1 Before Improving the Fairness of New-Reno + BECN

The following figures show the four metrics for different TCP variants.

Figure 27: Bottleneck Link Utilization for FTP with Light Web Traffic (Homogeneous Environment: Unequal-RTT)

Figure 28: Fairness Index for FTP with Light Web Traffic Before Improvement
(Homogeneous Environment: Unequal-RTT)

Figure 29: Drop Rate for FTP with Light Web Traffic
(Homogeneous Environment: Unequal-RTT)

Figure 30: Delay for FTP with Light Web Traffic
(Homogeneous Environment: Unequal-RTT)

The observations in Figures 27, 29 and 30 are consistent with those in section 4.1. So, the explanations provided there are applicable to this case as well.

In Figure 28, after a careful analysis, it becomes clear that the confidence intervals do not overlap at 16 connections. So, except in heavy congestion, we see that Vegas acts more fairly toward long and short connections in comparison with New-Reno + BECN and New-Reno. In [6], there is a mathematical argument that supports this finding. In essence, the argument is centered on studying the behavior of one long and one short connections simultaneously flowing in a simple closed fluid model.

It is then shown that under quasi-static assumption, the flow rate of each connection is proportional to its queue size. So, in Vegas, the flow rate does not depend on the propagation delay of the connection. But, each connection should try to keep its queue size between α and β . The values of α and β are usually set to 1 and 3. Thus, given these values, the ratio of the two throughputs is at worst equal to 3. But, if the window sizes of the two connections are such that the two queue sizes are exactly equal, the ratio of the two throughputs would then be 1 (absolute fairness). Therefore, the ratio is always maintained between 1 and 3.

In order to see what happens in practice, a number of simulations are then conducted. In each simulation, a different value is used for the difference between the delays of the two connections. Then, the ratio of the two throughputs can be observed. The results illustrate that some ratios are closer to one (higher fairness), while others are closer to 3 (lower fairness). In short, the mathematical argument and its corresponding simulations show that Vegas is not completely fair. But, the amount of its unfairness is limited.

This property, however, does not exist in New-Reno. In Figure 28, with or without BECN, the fairness of New-Reno is lower than that of Vegas. This is due to the nature of New-Reno. New-Reno relies solely on packet drops as a sign of congestion, and never looks at the RTT of the packets as a means for determining the state of congestion. So, the New-Reno sender simply sends a set of segments, waits for their ACKs to return and then increases its congestion window in the next RTT. The rate of the increase, of course, differs in the case of slow start from the case of congestion avoidance.

Since the ACKs to a set of sent segments come fast in the case of a short-RTT flow, the sender of this flow increases its congestion window quickly. But, it takes a longer time for the sender of a long-RTT connection to receive the ACKs to its sent segments. Therefore, this sender increases its congestion window relatively slowly. This is why short-RTT connections consume most of the available bandwidth in the bottleneck link when they are together with long-RTT connections. This unequal use of bandwidth by various connections brings down the value of fairness index for New-Reno (+ BECN).

In order to further investigate the results shown on Figures 28, it is useful to do a careful analysis on the performance of long and short connections for different TCP variants. The following figures depict the throughput across time of a pair of long and short connections when there are a total of 16 connections present.

Figure 31: Throughputs of a Pair of Long and Short Connections (New-Reno + BECN, Before the Improvement)

Figure 32: Throughputs of a Pair of Long and Short Connections (New-Reno)

Figure 33: Throughputs of a Pair of Long and Short Connections (Vegas)

In Figure 33, the bottleneck link utilization values of long-RTT and short-RTT connections are closer together than in Figures 31 and 32. This shows practically why Vegas is fairer than New-Reno + BECN and New-Reno.

5.2 After Improving the Fairness of New-Reno + BECN

In Figure 28, the fairness of Vegas is higher than that of the other two variants. But, between New-Reno + BECN and New-Reno, neither is fairer than the other. It is, however, possible to introduce an improvement to the algorithm of BECN that can considerably boost the fairness of New-Reno + BECN.

As indicated in section 2.9, upon receiving an ISQ message due to a marked packet, the TCP New-Reno sender sets its congestion window and *ssthresh* to one-half of the current congestion window and waits one full RTT before increasing its window. As an improvement, then, the senders in connections with RTTs greater than 400 ms are freed from having to wait one RTT before increasing their congestion windows. Figure 34 shows the fairness of New-Reno + BECN after this improvement. The fairness of the other two variants, of course, is similar to Figure 28.

Figure 34: Fairness Index for FTP with Light Web Traffic After Improvement
(Homogeneous Environment: Unequal-RTT)

The above figure presents some changes in comparison with Figure 28. First of all, for 8 and 16 connections, the New-Reno + BECN shows better fairness than New-Reno.

In both New-Reno and New-Reno + BECN, 8 and 16 connections cause light and mid-level congestion, respectively. Secondly, unlike Figure 28, Vegas is not fairer than New-Reno + BECN for 8 connections. Thirdly, for 16 connections, New-Reno + BECN is fairer than Vegas.

5.3 Analyzing the Improvement of New-Reno + BECN

Fairness

Inherently, the improvement is more pronounced when two correlated conditions are met. First of all, the minimum RTTs of short connections should be considerably less than 400 ms. This is because after adding the queuing delay, if the RTTs of short connections go above 400 ms, the sharing of bottleneck bandwidth will again shift in favor of short-RTT connections. In such a case, not only short-RTT connections naturally inject more packets into the network than long-RTT connections, but they even send additional packets by benefiting from the improvement every time their RTTs surpass 400 ms.

Secondly, the level of congestion for New-Reno + BECN should be light or mid-level. According to Figure 30, at 8 and 16 connections, the values of average queuing delay per packet are 110 ms and 160 ms, respectively. These values become 210 ms and 230 ms for 24 and 32 connections, respectively. At 32 connections, for instance, the RTT of a short connection with minimum RTT of 184 ms reaches 414 ms. This is why in Figure 34, no improvement is noticeable for 24 and 32 connections.

In order to practically see the impact of the improvement, let's do a careful analysis on the throughput across time of a pair of long and short connections, when there are a total of 16 connections present (mid-level congestion for New-Reno + BECN).

Figure 35: Throughputs of a Pair of Long and Short Connections
(New-Reno + BECN, After the Improvement)

A comparison of the above figure with Figure 31 indicates that the link utilization of the short-RTT connection is decreasing, while that of the long-RTT connection is increasing. This is the direct result of the improvement.

Chapter 6: Conclusions / Future Research

From the variety of simulation test cases studied, we can draw a number of conclusions.

6.1 A Set of Equal-RTT FTP Connections in a Homogeneous Environment

For this part, the conclusions are:

#1) There is no noticeable difference in the throughputs of the three TCP variants.

#2) For the number of connections that cause mid-level congestion in New-Reno (+ BECN), New-Reno (+ BECN) is fairer than Vegas. Also, for the number of connections that cause mid-level congestion in Vegas, again New-Reno (+ BECN) is fairer than Vegas. However, it is not possible to distinguish between the fairness of New-Reno + BECN and New-Reno.

#3) New-Reno + BECN has the smallest drop rate across all levels of congestion.

Vegas then has a lower drop rate than New-Reno for light and mid-level congestion.

In heavy congestion, however, the drop rate of Vegas is higher than that of New-Reno.

#4) For the number of connections that cause light and mid-level congestion in New-Reno (+ BECN), Vegas has a lower delay than New-Reno (+ BECN). But, as the level of congestion becomes heavier, the delay of Vegas increases. Also, in mid-level congestion, New-Reno + BECN has a higher delay than New-Reno.

6.2 A Set of Equal-RTT FTP Connections in a Heterogeneous Environment

For this part, the conclusions are:

- #1) Under light congestion, Vegas has the lowest link utilization. As the level of congestion gets heavier, the link utilization of Vegas increases. At heavy congestion, the link utilization of Vegas is higher than that of the other two variants. Between New-Reno + BECN and New-Reno, neither generally surpasses the other in utilizing the bottleneck link.
- #2) The drop rate of New-Reno + BECN is the lowest. However, it is not possible to distinguish between the drop rates of New-Reno and Vegas.

6.3 A Set of Equal-RTT FTP Connections in a Homogeneous/Heterogeneous Environment with Heavy Web Traffic

For this part, the conclusions are:

- #1) The amount of web traffic is so high that even with 5 FTP connections in the background, the network frequently experiences heavy congestion.
- #2) Whether it is used with New-Reno for FTP or web traffic, BECN significantly lowers the amount of drops.
- #3) When heavy web traffic runs over TCP Vegas, it does not let the accompanying FTP traffic to increase its share of bandwidth even as the number of FTP connections increases. Also, the queuing delay is high when Vegas carries heavy web traffic.

6.4 A Set of Unequal-RTT FTP Connections in a Homogeneous Environment

For this part, the conclusions are:

- #1) The points 1, 3 and 4 explained in section 6.1 are applicable here as well.
- #2) Except under heavy congestion, the fairness of Vegas is higher than that of New-Reno (+ BECN).
- #3) Under mid-level congestion, after the improvement, the fairness of New-Reno + BECN becomes higher than that of Vegas. Also, the improvement takes away the higher fairness of Vegas over New-Reno + BECN under light congestion. Due to the improvement, New-Reno + BECN is fairer than New-Reno under both light and mid-level congestion. But, as congestion gets heavier, the effect of improvement disappears.

6.5 Summary of Conclusions

An overall observation of all the drawn conclusions indicates a number of key results. First of all, when alone, Vegas presents a noticeably better performance than New-Reno in terms of drop rate and delay. But, when New-Reno is accompanied with BECN, its

drop rate goes down significantly. Furthermore, for unequal-RTT connections, Vegas is fairer than New-Reno (+ BECN). But, a slight modification of the BECN algorithm can boost the fairness of New-Reno + BECN considerably.

In the cases where Vegas competes with New-Reno (+ BECN), it can suffer due to the aggressive nature of New-Reno (+ BECN). However, in heavy congestion, Vegas may achieve a better throughput than New-Reno (+ BECN). Finally, we observe that persistent congestion is an important problem associated with TCP Vegas. This problem shows itself, for instance, when heavy web traffic is present.

6.6 Future Research

In order to deploy TCP Vegas, the adoption of a gateway control scheme such as Deficit Round-Robin (DRR) is a research topic. Especially, DRR gateways create certain isolation among the users in the network and draw the users towards cooperating more with the network. This can be a good motivation for TCP Reno users to switch to TCP Vegas [6].

Furthermore, two inherent problems in TCP Vegas are indicated in [21]: re-routing and persistent congestion. While corresponding solutions are suggested in the same work, there is still room for better solutions.

Besides New-Reno, already several other modifications of Reno have been introduced, such as TCP SACK and TCP FACK. But, it is still possible to come up with better algorithms that can handle congestion in a more efficient manner.

In addition, there are two ways of enhancing BECN. The first way is to use multiple-level BECN (MECN), which is one step further from BECN. The second way is through combining ECN and BECN to provide a more robust scheme for controlling congestion [17]. Thus, after implementing these enhancements, it would be interesting to see the impact of their use on the test cases studied in this research. Also, more generally, BECN can be used with other transport protocols such as UDP to provide performance improvements. Therefore, doing more investigations into the potential benefits of using BECN with those protocols can be a research topic.

Finally, while we have carried out extensive simulations in our research, it would be quite beneficial to compare the performance of New-Reno + BECN with Vegas over the Internet. In fact, such kind of study would provide more realistic results about the characteristics of these two schemes.

References

- [1] Jon Postel, "Transmission Control Protocol", Request For Comments 793, The Internet Engineering Task Force (IETF), September 1981.

- [2] "TCP Protocol Overview", Connected: An Internet Encyclopedia, <http://www.freesoft.org/CIE/Topics/83.htm> (January 29, 2004)

- [3] M. Allman et al., "TCP Congestion Control", Request For Comments 2581, The Internet Engineering Task Force (IETF), April 1999.

- [4] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", ACM Computer Communication Review, Vol. 26, No. 3, pp. 5-21, July 1996.

- [5] Harkirat Singh, "Analysis of Energy Consumption of TCP Reno, Newreno, and SACK in Multi-Hop Wireless Network", www.cs.pdx.edu/~harkirat/thesis/thesis.ppt, (January 29, 2004)

- [6] Jeonghoon Mo, Richard J. La, Venkat Anantharam and Jean Walrand, "Analysis and Comparison of TCP Reno and Vegas", IEEE INFOCOM'99, March 1999.

- [7] L.S. Brakmo and L.L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", IEEE Journal on Selected Areas in Communications, 13(8): 1465-80, October 1995.
- [8] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks", ACM Computer Commun. Review, vol. 19, no 5, pp 56-71, Oct. 1989.
- [9] Z. Wang and J. Crowcroft, "A new congestion control scheme: Slow start and search (Tri-S)" ACM Computer Commun. Review, vol. 21, no. 1, pp. 32-43, Jan. 1991.
- [10] Z. Wang and J. Crowcroft, "Eliminating periodic packet losses in 4.3-Tahoe BSD TCP congestion control algorithm", ACM Computer Commun. Review, vol. 22, no. 2, pp. 9-16, Apr. 1992.
- [11] J.S. Ahn, P.B. Danzig, Z. Liu and L. Tan, "Evaluation of TCP Vegas: Emulation and Experiment", IEEE Transactions on Communications, 25(4): 185-95, Oct. 1995.
- [12] N. C. Hutchinson and L. L. Peterson, "The x-kernel: An Architecture for implementing network protocols", IEEE Trans. Software Eng., vol. 17, no. 1, pp. 64 -76, Jan 1991.

[13] "Random Early Detection",

<http://www.ee.ust.hk/~heixj/publication/thesis/node36.html>, (January 29, 2004)

[14] Minseok Kwon and Sonia Fahmy, "TCP Increase/Decrease Behavior with Explicit Congestion Notification (ECN)",

www.cs.purdue.edu/homes/fahmy/talks/ecn-icc02.ppt, (January 29, 2004)

[15] K. Ramakrishnan and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP", Request For Comments 2481, The Internet Engineering Task Force (IETF), January 1999.

[16] J. Postel, "Internet Control Message Protocol", RFC 792, September 1981.

[17] F. Akujobi, et al., "BECN for Congestion Control in TCP/IP Networks: Study and Comparative Evaluation", In Proceedings of GLOBECOM 2002, vol. 3, 2588-2593, Nov. 2002.

[18] J. Hadi Salim, B. Nandy, N. Seddigh, "A proposal for Backward ECN for the Internet Protocol (IPv4/IPv6)", Internet Draft <draft-salim-jhsbnns-ecn00.txt> July 1998. Available at <http://www.sce.carleton.ca/~nseddigh/ietf.html>, (January 29, 2004)

- [19] UCB/LBNL/VINT Network Simulator (NS). Available at
<http://www.isi.edu/nsnam/ns>, (January 29, 2004)
- [20] S. Floyd, V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, V.1, N.4, August 1993.
- [21] "Estimates and Confidence Intervals for the Mean",
<http://www.vims.edu/~david/MS505/estimate.pdf>, (January 29, 2004)
- [22] Richard J. La, Jean Walrand, and Venkat Anantharam, "Issues in TCP Vegas",
available at
<http://www.eecs.berkeley.edu/~ananth/1999-2001/Richard/IssuesInTCPVegas.pdf>
(January 29, 2004)
- [23] "PC Pitstop", <http://www.pctuneup.com/internet/pinger.asp>, (January 29, 2004)