國 立 交 通 大 學

資 訊 工 程 學 系

博 士 論 文

TCP 壅塞控制技術之研究與設計

Study and Design of TCP Congestion Control Techniques

研 究 生：詹益禎

指導教授：陳耀宗 博士

中 華 民 國 九 十 三 年 六 月

TCP 壅塞控制技術之研究與設計

Study and Design of TCP Congestion Control Techniques

研 究 生：詹益禎　　Student: Yi-Cheng Chan

指導教授：陳耀宗 教授　　Advisor: Yaw-Chung Chen

國 立 交 通 大 學
資 訊 工 程 學 系
博 士 論 文

A Dissertation
Submitted to Institute of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
In Partial Fulfillment of the requirements
For the Degree of
Doctor of Philosophy
in
Computer Science and Information Engineering

June 2004
Hsinchu, Taiwan, Republic of China

中 華 民 國　九十三　年　六　月

Study and Design of TCP Congestion Control Techniques

Student: Yi-Cheng Chan

Advisor: Dr. Yaw-Chung Chen

A Dissertation Submitted to

the Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Chiao-Tung University

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science and Information Engineering

Hsinchu, Taiwan, Republic of China

June 2004

TCP 壅塞控制技術之研究與設計

學生：詹益禎　　　　　　　　　　　指導教授：陳耀宗博士

國立交通大學資訊工程學系

摘　　要

隨著網際網路訊務流量的快速成長，如何以有效率的方式使用網路資源是一個成功的壅塞控制機制所要面對的根本問題。TCP 身為網際網路上一個被廣為使用的端對端傳輸層協定，它被創造出許多不同的版本，用來改進網路的使用效能。在目前的 TCP 版本中有兩個特別值得留意的方案，一個是現今網際網路上廣為使用的 Reno，另一個則是宣稱相較於 Reno 可增進百分之三十七到七十一傳輸效能的 Vegas。

TCP Vegas 能夠偵測出初期的網路壅塞而且可以成功的避免在 TCP Reno 上時常發生的週期性封包遺失的現象，很多研究報告已經指出，Vegas 在很多方面都要優於 Reno，例如整體網路的使用率、穩定性、公平性、以及傳輸速率等。很可惜的是 Vegas 並不完美，仍然有一些缺點存在於它的壅塞控制機制中，這些問題或問題的起因包含了重新繞路、永久壅塞、競爭連線間的公平性、非對稱網路傳輸、高頻寬-延遲乘積網路、以及無線傳輸下的非壅塞封包遺失等。在這份論文中，我們提出了四個改進 Vegas 的機制，為 Vegas 移除邁向成功的障礙。這些新提出的機制有些是單純的端對端方法，有些則利用了路由器所提供的資訊以改善連線的傳輸效能。

第一個提出的方案 RoVegas 是一個使用路由器訊息回饋的改進方案，藉由封包路徑上的路由器執行特定的機制，RoVegas 可以解決重新繞路時所引起的問題，可以解決永久壅塞問題，也可以增進競爭連線間的公平性，以及改善在非對稱網路傳輸時，TCP 可能的效能損失。

Enhanced Vegas 是一個純粹端點對端點的改進機制，不用路由器的協助，它可以量測出發生在前送路徑和返回路徑上的網路壅塞程度，因此它能精準而合宜的調整封包的傳送速率，有效提高當壅塞發生在返回路徑時的連線效能。

由於在擁有大壅塞窗口時的反應速度過於緩慢，TCP 在高頻寬-延遲乘積網路中顯得效能不彰，因此第三個改進機制 Quick Vegas 被提出來。Quick Vegas 利用連

線壅塞窗口的調整紀錄以及連線估測堆積在佇列中的封包數量為依據，對 TCP 壅塞控制演算法做出調整，這個改變使得一個連線的送端在調整壅塞窗口大小時採取更有效和積極的態度，因此讓連線在高頻寬-延遲乘積網路中能有較好的表現。

TCP 壅塞控制的一個眾所週知的問題是它沒有辦法分辨出封包遺失的原因，傳統的 TCP 把所有封包遺失的原因都歸咎於網路的壅塞，這種推測在異質性日益顯著的網際網路中並不合宜。錯把傳輸失誤所造成的封包遺失當成網路壅塞的訊號將導致 TCP 不必要的效能損失。最後一個提出的改進方案 RedVegas 利用 TCP Vegas 原有的特性以及路由器在封包上的壅塞標記，可以準確的判斷出傳輸失誤所造成的封包遺失，透過封包遺失原因的分類，RedVegas 可以適切的對不同原因的封包遺失做出不同的反應，因此改善了在異質網路傳輸中的 TCP 效能。

Study and Design of TCP Congestion Control Techniques

Student: Yi-Cheng Chan Advisor: Dr. Yaw-Chung Chen

Institute of Computer Science and Information Engineering

National Chiao Tung University

ABSTRACT

With the fast growth of Internet traffic, how to efficiently utilize network resources is essential to a successful congestion control. Transmission Control Protocol (TCP) is a widely used end-to-end transport protocol in the Internet, it has several implementation versions (i.e., Tahoe, Reno, Vegas...) which intend to improve network utilization. Among these TCP variants, there are two notable approaches. One is Reno which has been widely deployed on the Internet; the other is Vegas with a claim of 37 to 71 percent throughput improvement over Reno was achieved.

TCP Vegas detects network congestion in the early stage and successfully prevents periodic packet loss that usually occurs in TCP Reno. It has been demonstrated that TCP Vegas outperforms TCP Reno in the aspects of overall network utilization, stability, fairness, and throughput. However, TCP Vegas still suffers problems that inhere in its congestion control algorithm, these include issues of rerouting, persistent congestion, fairness, network asymmetry, high bandwidth-delay product (BDP) networks, and internetworking of wired and wireless networks. In this dissertation, we propose four enhanced mechanisms to remove the obstacles of TCP Vegas for achieving a real success. These mechanisms not only adopt end-to-end approaches but also utilize the information that provided by routers to improve the performance of connections.

The first proposed mechanism, RoVegas, uses a router-assisted approach. By performing the proposed scheme in routers along the round-trip path, RoVegas can solve the problems of rerouting and persistent congestion, enhance the fairness among the competitive connections, and improve the throughput when congestion occurs on the backward path.

An end-to-end scheme, Enhanced Vegas, is also presented to improve the performance degradation of TCP Vegas in asymmetric networks. Through distinguishing whether congestion occurs in the forward path or not, Enhanced Vegas significantly advances the connection throughput when the backward path is congested.

TCP congestion control may function poorly in high BDP networks because of its slow response with large congestion window size. In the third mechanism, we propose an improved version of TCP Vegas called Quick Vegas, in which we present an efficient congestion window control algorithm for a TCP source. The modification allows TCP connections to react faster and better to high BDP networks and therefore improves the overall performance.

A well-known problem in providing TCP congestion control over wired and wireless networks is that it may encounter both congestion loss and random loss. Traditional TCP interprets every packet loss as caused by congestion which may not be the case in the current Internet. In the last proposed mechanism, RedVegas, we utilize the innate nature of TCP Vegas and congestion indications marked by routers to detect random packet losses precisely. Through the packet loss differentiation, RedVegas reacts appropriately to the losses, and therefore the throughput of connection over heterogeneous networks can be significantly improved.

Keywords: Asymmetric networks, heterogeneous networks, high bandwidth-delay product networks, Internet, TCP congestion control, TCP Vegas, transport protocol, wireless networks.

Acknowledgements

I come, I research, I conquer. Finally and fortunately, I acquire my Ph. D. degree.

Years ago, passing the entrance exam of NCTU, I got into the science research hall. Then I started leading a very different and unforgettable life from what I used to have. Especially the recent two years of my daily life was: studying, researching, experimenting, writing, submitting, waiting, and then repeating them all. This was not an easy life for me. Sometimes I would say it was "suffering" and "frustrating". So during this suffering and frustrating time, there must be some strength to help me to go through the dilemma. The strength is from my supervisor – Prof. Yaw-Chung Chen, my senior – Dr. Chia-Tai Chan, my wife, my two lovely kids, and my parents as well.

Here I will devote my gratitude to them that they always stayed with me and encouraged me to pass through the way overgrown with brambles. And let me find my broad road.

First I would say thanks to Prof. Yaw-Chung Chen. He is such a nice supervisor that never gives me too much pressure but will help me at the right time. No matter in life or in research, he is my best supervisor.

Second I would thank my senior, Dr. Chia-Tai Chan. In the preceding six years in NCTU, I was still having a job in Accton Corp. At that time, as getting no precise idea about my research, I was a little bit scared to throw myself into the unknown future. But Dr. Chia-Tai Chan who called me via the phone a couple of times and encouraged me "When you give up, you'll regret in your whole life". Yes, I agreed with him. Then I quit my job and concentrated on my research. And all the way, he stayed beside me as a best friend as well as a good teacher.

Then I appreciate my beloved family – my dear wife and lovely kids. Since I dedicated myself to finish my doctor degree, my wife – Jeannie, has become the backbone of my family. She needs to earn money to support the high tuition of two kids in kindergarten. She takes care of the two kids, most of household chores, ... etc. More importantly, she hardly complains. Moreover, I have to thank the two lovely kids: my daughter – Mia, and my son – Lucas. Being tired from my Lab. on the way home, I expect to watch the smiles from my two kids. They always make me relaxed and recharge my exhausting energy.

Finally, I would devote my thankfulness to my father and mother: without them, without me. They make me what I am today. Here I would dedicate this dissertation to both of them: my dear father – Mr. De-Zhen Chan, and my dear mother – Mrs. Shu Qien-Dai Chan.

Yi-Cheng Chan

National Chiao Tung University in Hsinchu, Taiwan.
June 2004

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Transmission Control Protocol (TCP) is the most popular transport protocol for the current Internet. It provides a reliable data transport between two end hosts of a connection as well as controls the connection's bandwidth usage to avoid network congestion. Many Internet applications use it as the underlying communication protocol. The behavior of TCP is therefore tightly coupled with the overall Internet performance.

The essential strategy of TCP is sending packets to the network without a reservation and then reacting to observable events occurred. Original TCP is officially defined in [1]. It has a simple sliding window flow control mechanism without any congestion control. After observing a series of congestion collapses in 1980's, Jacobson introduced several innovative congestion control mechanisms into TCP in 1988. This TCP version, called TCP Tahoe [2], includes the slow-start, additive increase and multiplicative decrease (AIMD), and fast retransmit algorithms. Two years later, the fast recovery algorithm was added to Tahoe to form a new TCP version called TCP Reno [3]. TCP Reno is currently the dominating TCP version deployed in the Internet.

TCP congestion control is an active research area. Over the years, considerable research regarding the knowledge about TCP has been done [4, 5, 6, 7, 8, 9, 10]. TCP Reno can be thought as a reactive congestion control scheme. It uses packet loss as an indicator for congestion. In order to probe the available bandwidth along the end-to-end path, TCP congestion window will be increased until a packet loss is detected, at which point the congestion window is halved and a linear increase algorithm will take over until further packet loss is experienced.

It is known that TCP Reno may periodically generate packet loss by itself and can not efficiently recover multiple packet losses from a window of data. Moreover, the AIMD strategy of TCP Reno leads to periodic oscillations in the aspects of congestion window size, round-trip delay, and queue length of the bottleneck node. Recent works have shown that the oscillation may induce chaotic behavior into the network thus adversely affects overall network performance [11, 12].

To alleviate the performance degradation problem of packet loss, many researchers attempted to refine the fast recovery algorithm which embedded in TCP Reno. New proposals includes TCP NewReno [13], SACK [14], FACK [15], Net Reno [16], and LT [17]. All these algorithms bring performance improvement for a connection after a packet loss is detected.

To combat the inherent oscillation problem of TCP Reno, many congestion avoidance mechanisms are proposed. These works include DUAL [18], CARD [19], Tri-S [20], Packet-Pair [21], TCP vegas [22, 23, 24, 25], and TCP Santa Cruz [26]. Among these creative mechanisms, there is a notable approach, TCP Vegas, with a claim of 37 to 71 percent throughput improvement over TCP Reno was achieved.

1.1 Motivation

With the fast growth of Internet traffic, how to efficiently utilize network resources is essential to a successful congestion control. TCP Vegas with a proactive congestion control strategy has the potential to provide a stable and efficient network environment.

To prevent the performance degradation caused by AIMD, TCP Vegas employs a fundamentally different congestion avoidance approach. It uses the difference between the expected and actual throughput to estimate the available bandwidth in the network. The idea is that when the network is not congested, the actual throughput will be close to the expected throughput. Otherwise the actual throughput will be smaller than the expected throughput. TCP Vegas uses the difference in throughput to gauge the congestion level in the network and update the congestion window size accordingly. As a result, TCP Vegas is able to detect network congestion in the early stage and successfully prevents periodic packet loss that usually occurs in TCP Reno. Many studies have demonstrated that TCP Vegas outperforms TCP Reno in the aspects of overall network utilization [23, 24], stability [9, 10], fairness [9, 10], and throughput [11, 23, 24].

Although TCP Vegas is superior to TCP Reno in the aforementioned aspects, however, TCP Vegas still suffers problems that inhere in its congestion control algorithm, these include issues of rerouting [9], persistent congestion [9], fairness [10, 27, 28], network asymmetry [29, 30, 31], high bandwidth-delay product (BDP) networks [32], internetworking of wired and wireless networks [33, 34], and incompatibility between TCP Reno and Vegas [9, 35, 36]. All these problems may prevent TCP Vegas from achieving a success.

1.2 Contributions

In this dissertation, we propose four enhanced mechanisms to remove the obstacles of TCP Vegas for achieving a success. These mechanisms not only adopt end-to-end approaches but also utilize the information that provided by routers to improve the performance of connections. We now briefly describe each proposed mechanism and its contributions as follows:

- **RoVegas:** The first proposed mechanism, RoVegas, is a router-assisted approach. In RoVegas, we define a new IP options named AQT (accumulate queuing time) to collect the queuing time experienced by a probing packet. By performing the proposed scheme in routers along the round-trip path, a RoVegas source may obtain the queuing time of both forward and backward directions as well as the fixed delay of the round-trip path. As a result, it can solve the problems of rerouting and persistent congestion, enhance the fairness among the competitive connections, and improve the throughput when congestion occurs along the backward path.

- **Enhanced Vegas:** An end-to-end scheme, Enhanced Vegas, is also presented to improve the performance degradation problem of TCP Vegas in asymmetric networks. The mechanism uses TCP timestamps option to estimate queueing delay on the forward and backward path separately without clock synchronization. Through distinguishing whether congestion occurs in the forward path or not, Enhanced Vegas significantly advances the connection throughput when the backward path is congested.

- **Quick Vegas:** TCP congestion control may function poorly in high BDP networks because of its slow response with large congestion window size. In the third mechanism, we propose an improved version of TCP Vegas called Quick Vegas, in which we present an efficient congestion window control algorithm for a TCP source. Our algorithm is based on the increment history and estimated amount of extra data to update the congestion window intelligently. The modification allows TCP connections to react faster and better to high BDP networks and therefore improves the overall performance.

- **RedVegas:** A well-known problem in providing TCP congestion control over wired and wireless networks is that it may encounter both congestion loss and random loss. Traditional TCP interprets every packet loss as caused by congestion which may not be the case in the current Internet. Misinterpretation of a random loss as an indication of network congestion results in TCP slowing down its sending rate unnecessarily. In the last proposed mechanism, RedVegas, we utilize the innate nature of TCP Vegas and congestion indications marked by routers to detect random packet losses precisely. Through the packet loss differentiation, RedVegas reacts appropriately to the losses, and therefore the throughput of connections over heterogeneous networks can be significantly improved.

1.3 Dissertation Outline

The rest of this dissertation is organized as follows. In Chapter 2, we review the design principles of the two notable TCP implementations, TCP Reno and TCP Vegas. Some variants of TCP Reno and several innovative congestion avoidance mechanisms are also discussed in this Chapter. The proposed RoVegas, Enhanced Vegas, Quick Vegas, and RedVegas are described and evaluated in Chapters 3, 4, 5, and 6 respectively. Finally, we conclude the main work of this dissertation and point out some future work in Chapter 7.

Chapter 2

Background

End hosts sharing a best-effort network need to respond to congestion by implementing congestion control mechanisms to ensure network stability. Otherwise, the network may be driven into congestion collapses. Over past decades, two main congestion control algorithms were proposed and tested in real networks. One is TCP Reno [2, 3], and the other is TCP Vegas [22, 23, 24, 25]. TCP Reno has been widely deployed on the current Internet. Several RFCs are documented for the implementation of TCP Reno. The basic functionality is recommended by [1, 37, 38, 39, 40] and extensions are exhibited in [14, 41, 42, 43]. Table 2.1 lists these RFCs. In the following sections, we summarize the congestion control mechanism embedded in TCP Reno and TCP Vegas. Besides, some variants of TCP Reno and several innovative congestion avoidance mechanisms are also discussed.

2.1 TCP Reno

TCP Reno is a window[1]-based congestion control mechanism. Its window-adjustment algorithm consists of three phases; slow-start, AIMD (additive increase/multiplicative decrease), and fast retransmit and recovery. A connection begins with the slow-start phase. The objective of slow-start is to enable a TCP connection to discover the

[1]TCP window refers to the amount of outstanding data that can be transmitted by the sender without acknowledgements.

Table 2.1: RFCs for the TCP implementation.

RFC number	**Topic**
793	Transmission Control Protocol
1122	Requirements for Internet Hosts - Communication Layers
1323	TCP Extensions for High Performance
2018	TCP Selective Acknowledgement Options
2581	TCP Congestion Control
2582	The NewReno Modification to TCP's Fast Recovery Algorithm
2914	Congestion Control Principles
3168	The Addition of Explicit Congestion Notification (ECN) to IP
3390	Increasing TCP's Initial Window

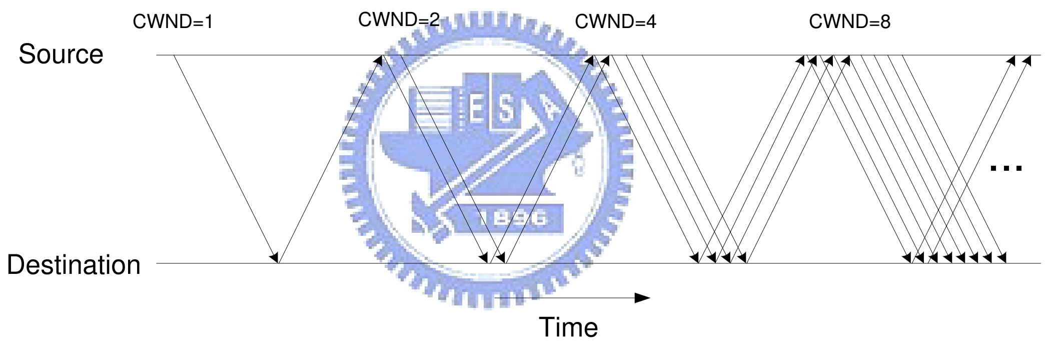


Figure 2.1: Packets in transit during slow-start.

available bandwidth by gradually increasing the amount of data injected into the network from the initial window size[2]. Upon receiving an acknowledgement packet (ACK), the congestion window size ($CWND$) is increased by one packet. With reference to Fig. 2.1, initially, the sender starts by transmitting one packet and waits for its ACK. When that ACK is received, the congestion window is incremented from one to two, and two packets can be sent. When both of these two packets are acknowledged, the congestion window is increased to four, and so on.

[2]RFC 2581 suggests an initial window size of two packets and RFC 3390 suggests a larger initial window can be used for reducing the duration of startup period, specifically for connections running in long propagation delay networks.

Since the *CWND* in the slow-start phase expands exponentially, the packets sent at this increasing rate would quickly lead to network congestion. To avoid this, the AIMD phase begins when *CWND* reaches the slow-start threshold (*SSTHRESH*). In AIMD phase, the *CWND* is added by $1/CWND$ packet every once receiving an ACK, this makes window size grow linearly. The process continues until a packet loss is detected and then the the *CWND* will be cut by half.

There are two ways for TCP Reno to detect packet loss. One is based on the reception of three duplicate ACKs, the other is based on retransmission timeout. When a source receives three duplicate ACKs, the fast retransmit and recovery algorithm is performed. It retransmits the lost packet immediately without waiting for a coarse-grained timer to expire. In the meantime, the *SSTHRESH* is set to half of *CWND*, which is then set to *SSTHRESH* plus the number of duplicate ACKs. The *CWND* is increased by one packet every once receiving a duplicate ACK. When the ACK of a retransmitted packet is received, the *CWND* is set to *SSTHRESH* and the source reenters the AIMD phase.

If a serious congestion occurs and there is no sufficient survived packets to trigger three duplicate ACKs, the congestion will be detected by a coarse-grained retransmission timeout. When the retransmission timer expires, the *SSTHRESH* is set to half of *CWND* and then the *CWND* is reset to one and finally the source restarts from slow-start phase.

A window evolution example including three window-adjustment phases of TCP Reno can be referred to Fig. 2.2. A connection starts from slow-start phase with an exponentially increasing rate. Since the connection has no idea about the available bandwidth of the network, the over expanded window size incurs a severe congestion quickly. After a retransmission timeout, the connection restarts from slow-start phase. When the *CWND* grows up to the *SSTHRESH*, the window size is increased linearly. After that, the pattern of periodically additive increasing and multiplicative decreasing of window size continues throughout the lifetime of the connection.

The fast retransmit and recovery algorithm of TCP Reno allows a connection to quickly recover from isolated packet losses. However, when multiple packets are

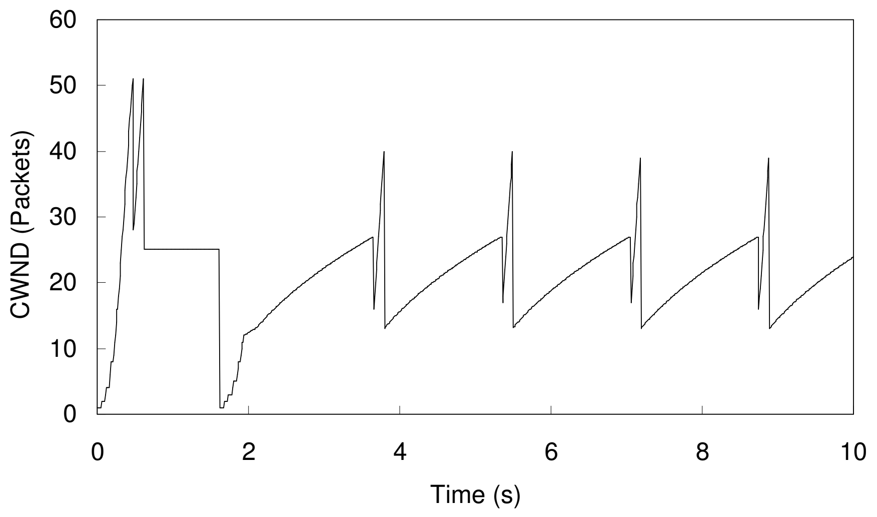


Figure 2.2: TCP Reno's window evolution.

dropped from a window of data, TCP Reno may suffer serious performance problems. Since it retransmits at most one dropped packet per round-trip time, and further the *CWND* may be decreased more than once due to multiple packet losses occurred during one round-trip time interval. In this situation, TCP Reno operates at a very low rate and loses a significant amount of throughput.

A number of enhanced loss recovery algorithms have been proposed to improve the above problem. In the following subsections, we briefly describe three noted remedies of TCP Reno, these include TCP NewReno [13], SACK [14], and FACK [15].

2.1.1 TCP NewReno

TCP NewReno makes a small change to a connection source, it may eliminate TCP Reno's waiting for a retransmission timeout when multiple packets are lost from a window. The change enhances the fast recovery algorithm of TCP Reno.

In TCP Reno, partial ACKs[3] bringing the connection out of fast recovery results

[3]Partial ACK is an acknowledgement that acknowledge some but not all of the outstanding packets at the start of that fast recovery phase.

in a retransmission timeout in case of multiple packet losses. In TCP NewReno, when a source receives a partial ACK, it won't get out of fast recovery [5, 42, 13]. Instead, it assumes that the packet immediately follows the most recently acknowledged packet has been lost, and hence retransmits the lost packet. Thus, in the situation of multiple packet losses, TCP NewReno will retransmit one lost packet per round-trip time until all of the lost packets from the same window have been recovered, and will not incur retransmission timeout. It remains in fast recovery phase until all of the outstanding packets at the start of that fast recovery phase have been acknowledged. Although this can avoid the unnecessary window reduction, the recovery time is still long. The implementation details of TCP NewReno has been specified in RFC 2582.

2.1.2 SACK

Another way to deal with multiple packet losses is to tell the source which packets have arrived at the destination. Selective Acknowledgments (SACK) does so exactly. TCP adapts accumulated acknowledgement strategy to acknowledge the successfully transmitted packets, this improves the robustness of acknowledgement when the path back to the source features high loss rate. However, the drawback of accumulated acknowledgement is that after a packet loss the source is unable to find out which packets are successfully transmitted. Therefore, it is unable to recover more than one lost packet in each round-trip time.

SACK option [14] field contains a number of SACK blocks, where each SACK blocks reports a non-contiguous set of data that has been received and buffered. The destination uses ACK with SACK option to inform the source one contiguous block of data that has been received out of order at the destination.

When SACK blocks are received by the source, they are used to maintain an image of the receiver queue, i.e., which packets are missing and which have been received at the destination. Scoreboard is set up to track those transmitted and received packets according to the previous information of the SACK option. For each transmitted packet, scoreboard records its sequence number and a flag bit that indicates whether the packet has been "SACKed". A packet with the SACKed bit turned on does not require to retransmit, but packets with the SACKed bit off and sequence number less than the highest SACKed packet are eligible for retransmission. Whether a SACKed packet is on or off, it is removed from the retransmission buffer only when it has been cumulatively acknowledged.

SACK TCP implementation still uses the same congestion control algorithms as TCP Reno. The main difference between SACK TCP and TCP Reno is the behavior in the event of multiple packet losses. SACK TCP refines the fast retransmit and fast recovery strategy of TCP Reno so that multiple lost packets in a single window can be recovered within one round-trip time.

2.1.3 FACK

Forward Acknowledgments (FACK) [15] was developed to decouple the congestion control algorithms from the data recovery algorithms. It uses the additional information provided by SACK option to keep an explicit measure of the total amount of outstanding data in the network. The goal of the FACK algorithm is to perform precise congestion control during recovery. By accurately controlling the outstanding data in the network, FACK can improve the connection throughput during the data recovery phase.

2.2 TCP Vegas

TCP Vegas is a rate-based congestion control mechanism. It can detect network congestion in the early stage and successfully prevents periodic packet loss that usually occurs in TCP Reno. TCP Vegas features three improvements as compared with TCP Reno: (1) a new retransmission mechanism, (2) an improved congestion avoidance mechanism, and (3) a more effective slow-start mechanism. We summary the design principles of TCP Vegas as follows.

TCP Vegas adopts a more sophisticated bandwidth estimation scheme that tries to avoid rather than to react to congestion. It uses the measured round-trip time (RTT) to accurately calculate the amount of data packets that a source can send. Its window adjustment algorithm consists of three phases: slow-start (SS), congestion avoidance (CA), and fast retransmit and fast recovery (FF). The congestion window is updated based on the currently executing phase.

During the congestion avoidance phase, TCP Vegas does not continually increase the congestion window. Instead, it tries to detect incipient congestion by comparing the actual throughput to the expected throughput. Vegas estimates a proper amount of extra data to be kept in the network pipe and controls the congestion window size accordingly. It records the RTT and sets $BaseRTT$ to the minimum of ever measured round-trip times. The amount of extra data (Δ) is estimated as follows:

$$\Delta = (Expected - Actual) \times BaseRTT, \tag{2.1}$$

where $Expected$ throughput is the current congestion window size ($CWND$) divided by $BaseRTT$, and $Actual$ throughput represents the $CWND$ divided by the newly measured smoothed-RTT. The $CWND$ is kept constant when the Δ is between two thresholds α and β. If Δ is greater than β, it is taken as a sign for incipient congestion, thus the $CWND$ will be reduced. On the other hand, if the Δ is smaller than α, the available bandwidth may be under utilized. Hence, the $CWND$ will be increased. The updating of $CWND$ is per-RTT basis. The rule for congestion window adjustment can be expressed as follows:

$$CWND = \begin{cases} CWND + 1, & if\ \Delta < \alpha \\ CWND - 1, & if\ \Delta > \beta \\ CWND, & if\ \alpha \leq \Delta \leq \beta \end{cases} . \tag{2.2}$$

During the slow-start phase, TCP Vegas is similar to TCP Reno that allows a connection to quickly ramp up to the available bandwidth. However, to ensure that the sending rate will not increase too fast, TCP Vegas doubles the size of its congestion window only every other RTT. A similar congestion detection mechanism

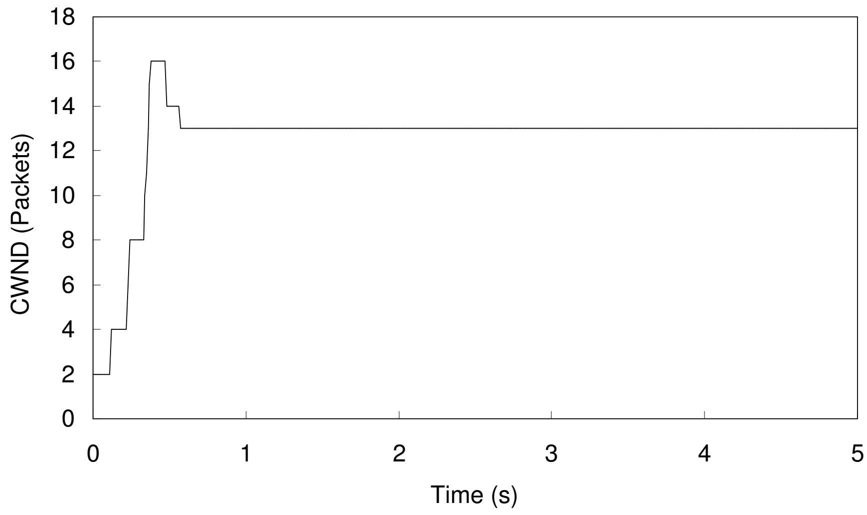


Figure 2.3: TCP Vegas' window evolution.

is applied during the slow-start to decide when to switch the phase. If the estimated amount of extra data is greater than γ, TCP Vegas leaves the slow-start phase, reduces its congestion window size by 1/8 and enters the congestion avoidance phase.

By keeping a proper amount of extra data in the network, TCP Vegas does not generate packet loss by itself. Ideally, it can maintain a stable window size as well as fully utilize the network resources if the network resources remain constant. An example of TCP Vegas' window evolution in a stable network environment can be referred to Fig. 2.3.

As in TCP Reno, a triple-duplicate acknowledgement (ACK) always results in packet retransmission. However, in order to retransmit the lost packets quickly, TCP Vegas extends TCP Reno's fast retransmission strategy. TCP Vegas measures the RTT for every packet sent based on fine-grained clock values. Using the fine-grained RTT measurements, a timeout period for each packet is computed. When a duplicate ACK is received, TCP Vegas will check whether the timeout period of the oldest unacknowledgement packet is expired. If so, the packet is retransmitted. This modification leads to packet retransmission after just one or two duplicate ACKs. When a non-duplicate ACK that is the first or second ACK after a fast

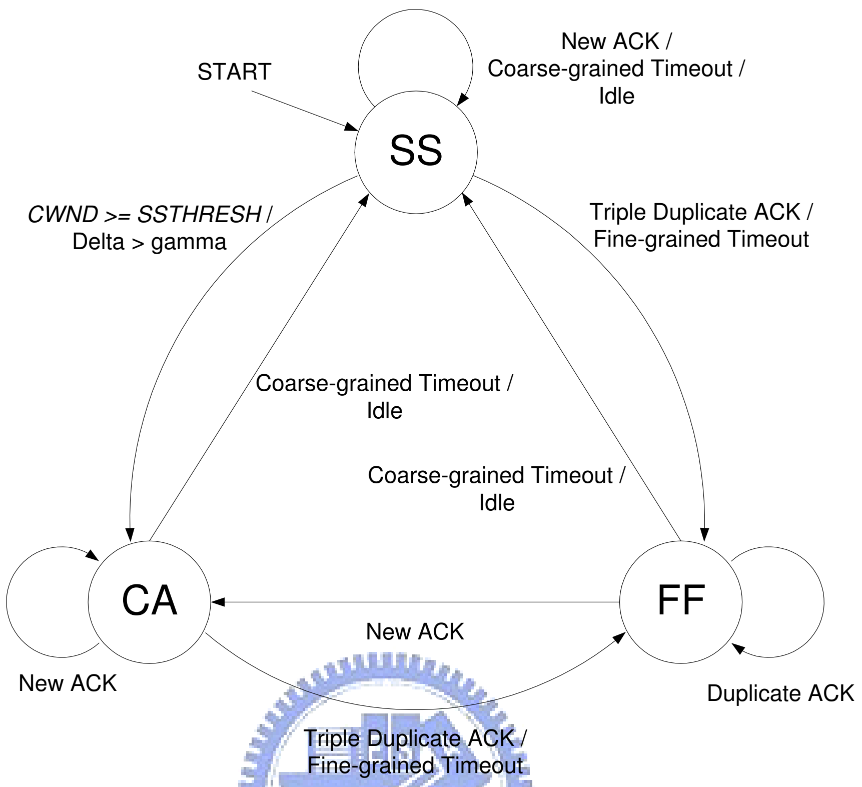


Figure 2.4: Phase transition diagram of TCP Vegas.

retransmission is received, TCP Vegas will again check for the expiration of the timer and may retransmit another packet. Note that, packet retransmission due to an expired fine-grained timer is conditioned on the reception of certain ACKs.

After a packet retransmission was triggered by a duplicate ACK and the ACK of the lost packet is received, the congestion window size will be reduced to alleviate the network congestion. There are two cases for TCP Vegas to set the *CWND*. If the lost packet has been transmitted just once, the *CWND* will be three fourth of the previous congestion window size. Otherwise, it is taken as a sign for more serious congestion, and one half of the previous congestion window size will be set to *CWND*. Notably, in case of multiple packet losses occurred during one round-trip time that trigger more than one fast retransmission, the congestion window will be reduced only for the first retransmission.

If a loss episode is severe enough that no ACKs are received to trigger fast

Table 2.2: Variable description of Fig. 2.5.

Variable	**Description**
ACKSeqNo	sequence number of the last successfully received packet
NumDupACK	number of duplicate ACK
RTO	duration of the coarse-grained retransmission timer
FGRTO	duration of the fine-grained retransmission timer
CWNDCT	last congestion window adjustment time due to a packet loss detection
SendTime	sending time of the lost packet
Delta	amount of extra data
LostSeqNo	squence number of the lost packet
NumTransmit	number of transmission times of the lost packet
NewCWND	congestion window size that will be used as a lost packet is recovered
IncrFlag	a flag used to adjust congestion window every other RTT
IncrAmt	increment amount of congestion window size for each new ACK is received
WorriedCtr	a counter used to check FGRTO after a lost packet is recovered

retransmit algorithm, eventually, the losses will be identified by Reno-style coarse-grained timeout. When this occurs, the slow-start threshold ($SSTHRESH$) will be set to one half of current $CWND$, then the $CWND$ will be reset to two, and finally the connection will restart from slow-start.

Figure 2.4 shows the phase transition diagram of TCP Vegas. A connection begins with the slow-start phase. The window-adjustment phase transition is owing to the specific events as depicted along the edges.

TCP congestion control is mainly based on the feedback of ACKs. The control procedure will be triggered whenever an ACK is received by the connection source. Figure 2.5 illustrates the detailed procedure of TCP Vegas as it receives an ACK. The description of variables used in Fig. 2.5 is shown in Table 2.2.

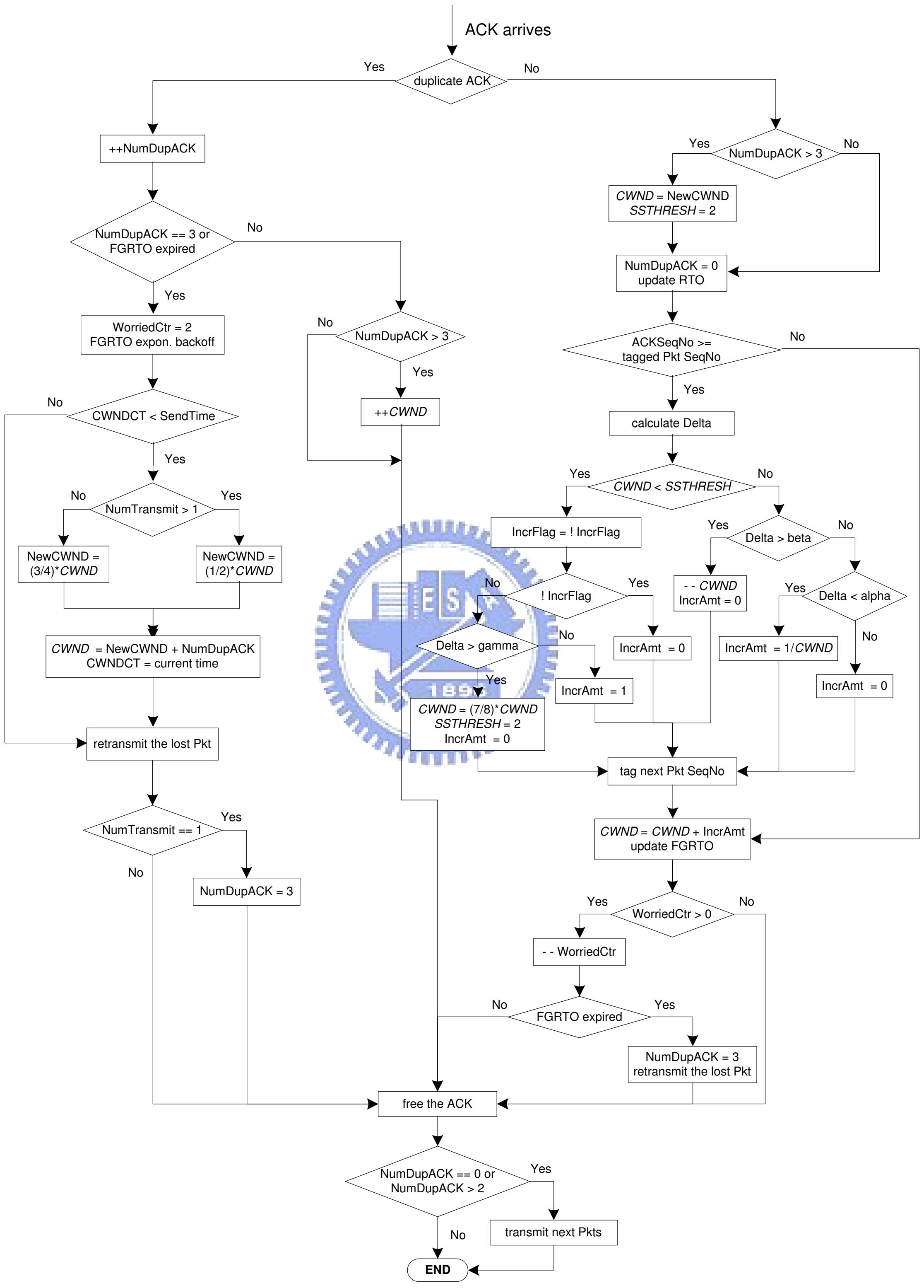


Figure 2.5: Flowchart of the procedure for TCP Vegas upon receiving an ACK.

2.3 Other Congestion Avoidance Mechanisms

TCP window size and the queue length of bottleneck node in the operation of TCP Reno often exhibit a clear oscillating behaviors when the traffic volume exceed the available resources. Such oscillation is inherent in the additive increase and multiplicative decrease algorithm and is used as a measure of probing resource changes.

In addition to TCP Vegas, many efforts of end-to-end congestion control mechanisms such as DUAL [18], CARD [19], Tri-S [20] have been paid since 1988 by steering system away from the periodic congestion losses and it is expected that a connection can operate in the equilibrium point. However, these proposals do not attract much attentions as compared to TCP Vegas. We briefly describe these mechanisms as follows.

The window in Jain's CARD [19] approach is increased by one packet size and decreased by one-eighth based on the gradient of delay-window curve, which is used to evaluate the optimal point of the system. The performance of the window control mechanism was studied with a deterministic simulation model of a connection in a wide-area network. Note that the window changes during every adjustment, that is, it oscillates around its optimal point.

DUAL scheme [18] defines one optimal point with queue length and uses the corresponding delay as the congestion signal. The congestion window normally uses fine-tuning to adjust window size, namely increases by $1/CWND$ for each ACK received. The algorithm decreases the congestion window by one-eighth if the current RTT is greater than the average of the minimum and maximum $RTTs$ observed so far for every two $RTTs$. If a timeout is detected, the algorithm assumes that substantial traffic increase and severe congestion have occurred. It uses quick-turning to reduce the window size, similar to TCP Tahoe timeout action ($CWND$ is set to 1 and $SSTHRESH$ is set to half of the window).

The Tri-S scheme proposed in [20], searches the operating point based on continuous evaluation of the current throughput gradient. For every RTT, the Tri-S increases window size by one packet and compares the throughput achieved to the throughput when the window was one packet smaller. It is this difference that determines the increase, decrease or unchange of the window.

TCP Santa Cruz [26] was designed with transmission-media heterogeneity in mind. With timestamp option in RFC 1323, it operates by summing the relative delays from the beginning of a session and then updating the measurements at discrete intervals. The bandwidth probing in this work is closely related to the Packet Pair [21], which uses the spacing of the ACKs to determine the available bandwidth in the networks. Similar to the proactive congestion avoidance mechanism in TCP Vegas [22, 23], this monitoring of the available bandwidth permits the detection of the incipient stage of congestion, and allows the congestion window to increase or decrease in response to early warning signs to reach a target optimal operating point.

2.4 Chapter Summary

In this chapter, we outline the design principles of TCP Reno and TCP Vegas. TCP Reno uses packet loss as a signal to indicate that network is congested and reduces its window size accordingly. Therefore, TCP Reno can be concluded as a reactive congestion control mechanism. An appealing alternative, TCP Vegas, uses a sophisticated bandwidth estimation scheme to keep a proper amount of extra data in the network. As a result, it may steer the system away from congestion loss before it actually occurs. Thus, TCP Vegas is a proactive congestion control mechanism. Some variants of TCP Reno and several innovative congestion avoidance mechanisms are also reviewed in this chapter.

Chapter 3

RoVegas: A Router-Assisted Congestion Avoidance Mechanism for TCP Vegas

The most innovative idea of TCP Vegas is its congestion avoidance mechanism. It uses queueing delay as the congestion measure to predict whether congestion is about to happen. Queueing delay may provide a more fine-grained information of the network status than the binary signal – packet loss. Based on the additional fine-grained information, TCP Vegas not only reacts to but also avoids congestion. As a result, it can prevent the performance degradation caused by AIMD strategy and may provide a more stable and efficient transmission as compared to that of TCP Reno.

However, the measurement of queueing delay is noisy. An inaccurate queueing delay estimation may incur serious impact on the performance. In this chapter, we propose a router-assisted congestion avoidance mechanism (RoVegas) for TCP Vegas. Through the proposed mechanism performed in routers along the round-trip path, RoVegas may obtain a more precise queueing delay and fixed delay measurement, and solve several problems that inhere in TCP Vegas.

The rest of this chapter is organized as follows. Section 3.1 describes the problems that inhere in the congestion avoidance mechanism of TCP Vegas. Section 3.2 discusses the RoVegas. In Section 3.3, related work is reviewed. Section 3.4 and 3.5 present the analysis and simulation results respectively. Lastly, we summarize this chapter in Section 3.6.

3.1 Problem Statements

In TCP Vegas, several problems may adversely affect the connection performance. We summarize these problems as follows.

Rerouting: Since TCP Vegas estimates the *BaseRTT* to compute the expected throughput and adjust its window size accordingly. Thus it is very important to estimate the quantity accurately for Vegas connections. Rerouting may cause a change of the fixed delay[1] that could result in substantial throughput degradation. When the routing path of a connection is changed, if the new route has a shorter fixed delay, it will not cause any serious problem for Vegas because most likely some packets will experience shorter round-trip time, and *BaseRTT* will be updated eventually. On the other hand, if the new route for the connection has a longer fixed delay, it would be unable to tell whether the increased round-trip time is due to network congestion or route change. The source host may misinterpret the increased round-trip time as a signal of congestion in the network and decrease its window size. This is just the opposite of what the source should do.

Persistent Congestion: Persistent congestion is another problem caused by the incorrect estimation of *BaseRTT* [9]. Overestimation of the *BaseRTT* in Vegas may cause a substantial influence on the performance. Suppose that a connection starts while many other active connections also exist, the network is congested and the packets are accumulated in the bottleneck. Then, due to the queuing delay caused by those packets from other connections, the packets of the new connection may experience a round-trip time that are considerably larger than the actual fixed

[1]The fixed delay is the sum of propagation delay and packet processing time along the round-trip path. In other words, the fixed delay is the round-trip time without queuing delay.

delay along the path. Hence, the window size of the new connection will be set to a value such that its expected amount of extra data lies between α and β; in fact, there may be much more extra data in the bottleneck queue due to the inaccurate estimation of the fixed delay. The situation can be more explicit described as follows.

TCP Vegas uses the following inequality to detect and control the extra data in the network pipe.

$$\alpha \leq (Expected - Actual) \times BaseRTT \leq \beta, \tag{3.1}$$

We can rewrite Eq. (3.1) as:

$$\alpha \leq CWND \times (1 - \frac{BaseRTT}{RTT}) \leq \beta. \tag{3.2}$$

An overestimated $BaseRTT$ will shrink the estimated amount of extra data (i. e., $CWND \times (1 - BaseRTT/RTT)$) and cause the Vegas source to misjudge that the network is not so congested. As a result, the Vegas source sets its window size larger than it should be and therefore puts more extra data on the bottleneck queue.

This scenario will repeat for each newly added connection, and it may cause the bottleneck node to remain in a persistent congestion. Persistent congestion is likely to happen in TCP Vegas due to its fine-tuned congestion avoidance mechanism.

Unfairness: Different from TCP Reno, TCP Vegas is not biased against the connections with longer round-trip time [9, 10]. However, there is still unfairness comeing with the nature of Vegas. According to the difference between the expected and actual throughputs, a Vegas source attempts to maintain an amount of extra data between two thresholds α and β by adjusting its congestion window size. The range between α and β induces uncertainty to the achievable throughput of connections. Since Vegas may keep different amount of extra data in the bottleneck even for the connections with the same round-trip path. Thus, it prohibits better fairness achievement among the competing connections.

Furthermore, the inaccurate computation of expected throughput may also lead to unfairness. Recall that the computation of expected throughput is based on the measurement of $BaseRTT$. If Vegas connections can not estimate the $BaseRTT$

accurately, it may affect the fairness achievement. When a new connection starts sending data while many other connections are also active, it may cause overestimation of the fixed delay and result in unfair distribution of bandwidth among the Vegas connections.

Network Asymmetry: Based on the estimated extra data kept in the bottleneck, Vegas updates its congestion window to avoid congestion as well as to maintain high throughput. However, a roughly measured RTT may lead to a coarse adjustment of congestion window size. If the network congestion occurs in the direction of ACK (backward path), it may underestimate the actual throughput and cause an unnecessary decreasing of the congestion window size. Ideally, congestion in the backward path should not affect the network throughput in the forward path, which is the data transfer direction. Obviously, the control mechanism must be able to distinguish whether congestion occurs in the forward path or not and adjust the congestion window size more intelligently.

Incompatibility: TCP Vegas adopts a proactive congestion avoidance scheme, it reduces its congestion window before an actual packet loss occurs. TCP Reno, on the other hand, employs a reactive congestion control mechanism. It keeps increasing its congestion window until a packet loss is detected. Researchers [9, 35, 36] have found that when Reno and Vegas perform head-to-head, Reno generally steals bandwidth from Vegas. This incompatibility between Vegas and Reno depress the adoption of Vegas on the Internet.

3.2 RoVegas

From the above discussion, we can find that the coarse estimation of fixed delay along the round-trip path, $BaseRTT$, results in problems such as issues of rerouting, persistent congestion, and unfairness. A Vegas source is unable to distinguish whether congestion occurs in the forward path or not, this further leads to unnecessary throughput degradation when the congestion occurs on the backward path. In this section, we propose a router-assisted congestion avoidance mechanism (RoVegas) for TCP Vegas to deal with these problems. The details of the proposed mechanism are described as follows.

3.2.1 Proposed Mechanism

TCP Vegas estimates a proper amount of extra data to be kept in the network pipe and controls the congestion window size accordingly. The amount is between two thresholds α and β, as shown in Eq. (3.1). When backward congestion occurs, the increased backward queuing time will affect the *Actual* throughput and enlarge the difference between the *Expected* throughput and *Actual* throughput. It results in decreasing the congestion window size. Since the network resources in the backward path should not affect the traffic in the forward path, it is unnecessary to reduce the congestion window size when only backward congestion occurs.

A measured RTT can be divided into four parts: forward fixed delay (i. e., propagation delay and packet processing time), forward queuing time, backward fixed delay, and backward queuing time. To utilize the network bandwidth efficiently, we redefine the *Actual* throughput as

$$Actual' = \frac{CWND}{RTT - QT_b}, \tag{3.3}$$

where RTT is the newly measured round-trip time, QT_b is the backward queuing time, and $CWND$ is the current congestion window size. Consequently, the $Actual'$ is a throughput that can be achieved if there is no backward queuing delay along the path.

To realize our scheme, we define a new IP option named AQT (accumulate queuing time) to collect the queuing time along the path. According to the general format of IP options described in [44], the fields of an AQT option are created as in Fig. 3.1. The option type and length fields indicate the type and length of this IP option. The AQT field expresses the accumulated queuing time that a packet experienced along the routing path. The AQT-Echo field echoes the accumulated queuing time value of an AQT option that was sent by the remote TCP.

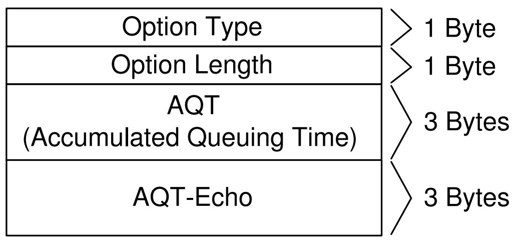


Figure 3.1: Fields of an AQT option.

A probing packet is a normal TCP packet (data or ACK) with AQT option in its IP header. When a RoVegas source sends out a probing packet, it sets the AQT field to zero. An AQT-enabled router (i. e., a router that is capable of AQT option processing) adds the queuing delay of a received probing packet to the AQT field. The queuing time is computed based on the queuing disciplines. The details regarding how to compute the queuing time of each received probing packet in various queuing disciplines is beyond the scope of our discussion.

Whenever a RoVegas destination acknowledges a probing packet, it inserts an AQT option into the ACK. The AQT-Echo field is set to the value of the AQT field of the received packet, then the AQT field is reset to zero. Through the AQT-enabled routers along the round-trip path, a RoVegas source is able to obtain both the forward queuing time (the value of AQT-Echo field) and backward queuing time (the value of AQT field) from the received probing packet. Moreover, for each probing packet received by a RoVegas source, the $BaseRTT$ can be derived as follows:

$$BaseRTT = RTT - (\text{AQT} + \text{AQT-Echo}). \tag{3.4}$$

Notice that, the derived $BaseRTT$ of a connection will be identical for each probing packet received when both the route and size of the probing packets are fixed. The derived $BaseRTT$ of RoVegas represents the actual fixed delay along the round-trip path, if the path of a connection is rerouted and the fixed delay is changed, the newly derived $BaseRTT$ may reflect the rerouting information. As a result, the issue of rerouting can be solved. Furthermore, since each connection of RoVegas is able to measure the fixed delay without bias, the problem of persistent congestion can be avoided and the fairness among the competitive connections can also be improved.

To avoid the unnecessary reduction of congestion window size, the proposed router-assisted congestion avoidance mechanism is described as follows:

- Derive the *Expected* throughput that is defined as the current congestion window size divided by *BaseRTT*.
- Calculate the $Actual'$ as the current congestion window size divided by the difference between the newly measured RTT and backward queuing time.
- Let $Diff = (Expected - Actual') \times BaseRTT$.
- Let w_{cur} and w_{next} be the congestion window sizes for the current RTT and the next RTT, respectively. The rule for congestion window adjustment is as follows:

$$w_{next} = \begin{cases} w_{cur} + 1, & if\ Diff < \alpha \\ w_{cur} - 1, & if\ Diff > \beta \\ w_{cur}, & if\ \alpha \le Diff \le \beta \end{cases} . \tag{3.5}$$

3.2.2 Implementation Issue

RoVegas relies on probing packets to probe the network status, therefore, how often a probing packet will be sent for a connection is an important issue. Since the window adjustment of RoVegas is performed on per-RTT basis. Inserting probing packets frequently makes the proposed mechanism robust against the network congestion, however, it also imposes more overhead on RoVegas. For the overhead induced by the probing packets, we consider the worst case that every packet with the AQT option. If the data packet size is 1500 bytes, which is the maximum transmission unit of Ethernet, the overhead ratio of data packets is 8/1500, which is about 0.53 %. In the practical implementation, the number of the probing packets per-RTT can be dynamically adjusted depending on the network status. That is, the more severe the backward congestion is, the more frequent the AQT option should be inserted into a data packet. Through this way, the overhead induced by the AQT option can be reduced to an even smaller amount.

We make every packet to be a probing packet and demonstrate that the proposed mechanism is effective to improve the performance of TCP Vegas by the results of both analysis and simulation shown in Sections 3.4 and 3.5.

3.3 Related Work

Congestion control for TCP is an active research area. Since Brakmo et al. [22, 23] proposed TCP Vegas in 1994 with claiming to achieve higher throughput and one-fifth to one-half the losses of TCP Reno, there have been quite a lot of studies focusing on TCP Vegas.

Ahn et al. [24] performed some live Internet experiments with TCP Vegas. They reproduced claims in [22, 23] with varying background traffic and concluded that Vegas indeed offers improved throughput of at least 3 to 8 percent over Reno. TCP Vegas is also found to retransmit fewer packets and to have a lower average and a lower variance of RTT than Reno.

By using the fluid model and simulations, Mo et al. [9] show that Vegas is not biased against connections with longer round-trip time like Reno does. It achieves better fairness of bandwidth sharing among the competitive connections with different propagation delays. However, they also pointed out that TCP Vegas does not receive a fair share of bandwidth in the presence of a TCP Reno connection.

Two problems of Vegas that could have serious impact on its performance are also described in [9]. One is the rerouting problem. Rerouting may lead to the change of fixed delay and therefore bring about inaccurate estimation of $BaseRTT$. This may erroneously affect the adjustment of the congestion window size. The other is the persistent congestion, which is still caused by the inaccurate estimation of $BaseRTT$.

Hasegawa et al. [10] focus on the fairness and stability of the congestion control mechanisms for TCP. They use an analytical model to derive that TCP Vegas can offer higher performance and much stable operation than Reno. However, because of the default values of α and β in the implementation, connections of Vegas could not share the total bandwidth in a fair manner. Thus Hasegawa et al. propose an enhanced Vegas which sets α equal to β to remove the uncertainty induced by the range between α and β.

Through the analytical study and simulation, Boutremans et al. [27] show that in addition to the setting of α and β, the fairness of TCP Vegas critically requires an accurate estimation of propagation delay. Nevertheless, they think there is no obvious way to achieve this.

To prevent the performance degradation of TCP Vegas in asymmetric networks, Elloumi et al. [29] proposed a modified algorithm. It divides a round-trip time into a forward trip time and a backward trip time in order to remove the effects of backward path congestion. However, it seems unlikely to work without clock synchronization.

Another mechanism for solving the issue of Vegas in asymmetric networks is proposed in [30, 31]. Fu et al. employ an end-to-end method to measure the actual flow rate on the forward path at a source of TCP Vegas. Based on the differences between the expected rate along the round-trip path and the actual flow rate on the forward path, the source adjusts the congestion window size accordingly. However, in a backward congestion environment the self-clocking behavior of TCP will be disturbed. Then the TCP traffic with bursty nature will make the source hard to decide the measure interval between two consecutive tagged packets. Moreover, the actual flow rate on the forward path measured by the source may be usually greater than the expected rate along the round-trip path. It may lead to an over-increased congestion window size, and causes congestion along the forward path.

To enhance the throughput of Vegas when it performs with TCP Reno head-to-head, Lai [35] suggests two approaches, one is using the random early detection (RED) mechanism in the router, the other is adjusting parameters of Vegas. Both may improve the performance of Vegas.

Feng et al. [36] show that the default configuration of Vegas is indeed incompatible with TCP Reno. However, with a careful analysis of how Reno and Vegas use

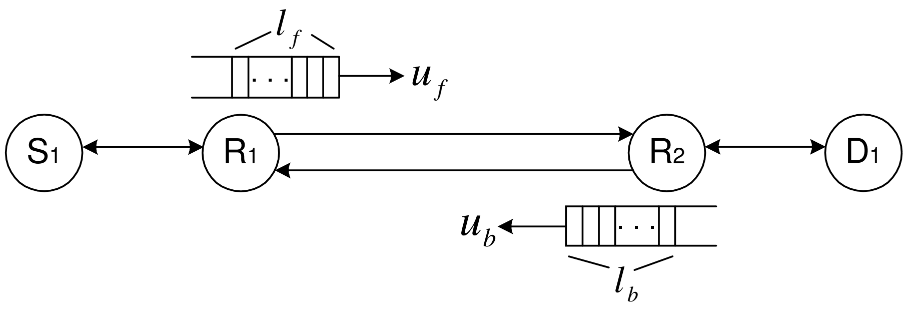


Figure 3.2: Network model for analysis.

buffer space in the routers, Vegas and Reno can be compatible with one another if Vegas is configured properly. Nevertheless, no mechanism has been proposed to configure Vegas automatically.

3.4 Performance Analysis

In this section, we present a steady-state performance analysis of both Vegas and RoVegas. By investigating the queue length of the bottleneck buffer through the analytical approach, we can clarify the essential nature of these two mechanisms. The network model used in the analysis is depicted in Fig. 3.2.

Assuming the source S_1 is a greedy source. The destination D_1 generates an ACK immediately upon receiving a data packet sent from S_1. Either the forward or backward link between two routers R_1 and R_2 is the bottleneck along the path. The forward link between two routers has a capacity of u_f (data packets per second) and backward link has a capacity u_b (ACKs per second). To facilitate the analysis as the backward path is congested, a normalized asymmetric factor k, $k{=}u_f/u_b$, is introduced [33]. The network is defined as asymmetric if the asymmetric factor is greater than one.

The service discipline is assumed to be First-In-First-Out (FIFO). Let τ be the $BaseRTT$ (without any queuing delay), l_f and l_b be the mean numbers of packets queued in the forward and backward bottleneck buffer respectively. Since the window size of Vegas converges to a fixed value in steady state, the mean number of packets queued in bottleneck buffer should also be converged to a fixed level [10].

3.4.1 Analysis on Vegas

The congestion avoidance mechanism of Vegas shown in Eq. (3.1) can be rewritten as below:

$$\frac{RTT}{RTT - BaseRTT} \times \alpha \leq CWND \leq \frac{RTT}{RTT - BaseRTT} \times \beta. \tag{3.6}$$

The $BaseRTT$ and RTT can be expressed as follows:

$$BaseRTT = \tau, \tag{3.7}$$

$$RTT = \tau + \frac{l_f}{u_f} + \frac{l_b}{u_b}. \tag{3.8}$$

After substitution of Eq. (3.7) and Eq. (3.8), Eq. (3.6) can be rewritten as:

$$\frac{\tau u_f u_b + l_f u_b + l_b u_f}{l_f u_b + l_b u_f} \times \alpha \leq CWND \leq \frac{\tau u_f u_b + l_f u_b + l_b u_f}{l_f u_b + l_b u_f} \times \beta. \tag{3.9}$$

Symmetric Network ($k \leq 1$): If the bottleneck is in the forward path, packets will be accumulated in the forward bottleneck queue and no packets will be queued in the backward path, that is $l_b = 0$, thus Eq. (3.9) can be simplified as:

$$\frac{\tau u_f + l_f}{l_f} \times \alpha \leq CWND \leq \frac{\tau u_f + l_f}{l_f} \times \beta. \tag{3.10}$$

Since S$_1$ is the only traffic source in the network thus it may occupy all the bandwidth of the bottleneck. Based on the fluid approximation, the congestion window size of S$_1$ can be obtained through the bandwidth-delay product of the bottleneck as follows:

$$CWND = u_f \times (\tau + \frac{l_f}{u_f}). \tag{3.11}$$

By substituting Eq. (3.11) into Eq. (3.10), we have

$$\alpha \leq l_f \leq \beta. \tag{3.12}$$

The throughput T of S_1 can also be derived from Eq. (3.8) and Eq. (3.11) as:

$$T = \frac{CWND}{RTT} = u_f. \tag{3.13}$$

From Eq. (3.12) and Eq. (3.13), we observe that when the bottleneck appears in the forward path, the mean number of packets queued in forward bottleneck buffer is kept stable between α and β, and the link bandwidth is always fully utilized in steady state. This observation matches the design goal of Vegas.

Asymmetric Network ($k > 1$): If the bottleneck exists in the backward path then the queue of the backward bottleneck node will be built up and no packets will be queued in the forward path, that is $l_f = 0$, therefore Eq. (3.9) can be rewritten as:

$$\frac{\tau u_b + l_b}{l_b} \times \alpha \leq CWND \leq \frac{\tau u_b + l_b}{l_b} \times \beta. \tag{3.14}$$

Similar to the Eq. (3.11), the window size of S_1 can also be obtained by the bandwidth-delay product of the bottleneck link:

$$CWND = u_b \times (\tau + \frac{l_b}{u_b}). \tag{3.15}$$

By substituting Eq. (3.15) into Eq. (3.14), we have

$$\alpha \leq l_b \leq \beta. \tag{3.16}$$

In the meantime, the throughput T of S_1 can be derived from Eq. (3.8) and Eq. (3.15) as:

$$T = \frac{CWND}{RTT} = u_b = \frac{u_f}{k}. \tag{3.17}$$

From Eq. (3.16) we can find that, Vegas is unable to distinguish whether congestion occurs in the forward path or not. It keeps a steady quantity of extra data between α and β on the backward path. This may lead to poor utilization of forward path. As shown in Eq. (3.17), the throughput of S_1 is limited by the capacity of backward path. Notably, an ACK in the backward path implies that a data packet has arrived at its destination. Therefore, the throughput of S_1 is u_f/k (data packets per second).

3.4.2 Analysis on RoVegas

The congestion avoidance mechanism of RoVegas can be briefly expressed as follows:

$$\frac{\alpha}{BaseRTT} \leq \frac{CWND}{BaseRTT} - \frac{CWND}{RTT - \frac{l_b}{u_b}} \leq \frac{\beta}{BaseRTT}. \tag{3.18}$$

By Eq. (3.18), we have the congestion window size of RoVegas as:

$$\frac{RTT}{RTT - BaseRTT - \frac{l_b}{u_b}} \times \alpha \leq CWND \leq \frac{RTT}{RTT - BaseRTT - \frac{l_b}{u_b}} \times \beta. \tag{3.19}$$

From Eq. (3.7) and Eq. (3.8), Eq. (3.19) can be rewritten as:

$$\frac{\tau u_f + l_f}{l_f} \times \alpha \leq CWND \leq \frac{\tau u_f + l_f}{l_f} \times \beta. \tag{3.20}$$

Since the result of Eq. (3.20) is identical to that of Eq. (3.10). If the bottleneck is in the forward path (i.e., $l_b = 0$), the behavior of RoVegas will be same as Vegas. However, the result of Eq. (3.20) reveals that the throughput of RoVegas in the case of backward congestion is not simply limited by the bandwidth of backward path as that of Vegas.

As shown in Eq. (3.18), RoVegas always attempts to maintain a proper amount of extra data in the forward path regardless of where the congestion occurs. However, TCP is a “self-clocking” protocol, that is, it uses ACKs as a “clock” to strobe new packets into the network [2]. Hence, as the backward path is congested, the rate of data flow in the forward direction will be throttled in a manner by the rate of ACK flow.

There exists a further restriction in Vegas that may limit the growth of the congestion window. The congestion window will not be increased if the source is unable to keep up with, that is, the difference between the congestion window size and the amount of outstanding data is larger than two maximum-sized packets [25]. Be a variant of TCP Vegas, RoVegas also complies with this restriction.

In an asymmetric network, for example $k = 8$, assuming that in steady state the forward path can be fully utilized by S$_1$, it means that 7/8 of ACKs will be dropped in the backward path. With TCP, the ACKs are cumulative [45], that is, later

ACKs carry all the information contained in earlier ACKs. In this case, a survived ACK may represent that eight data packets have arrived at the destination. Once a survived ACK is received by the source, the difference between the congestion window size and the amount of outstanding data is eight packets. It will restrict the growth of the congestion window. Actually, the forward path may not be fully utilized by RoVegas with $k = 8$.

For an asymmetric network, if the dropping ratio of ACKs reaches 2/3, the congestion window of RoVegas will not be increased. Since for each ACK received by the RoVegas source, the difference between the congestion window size and the amount of outstanding data will be three packets. In such situation, RoVegas enters the steady state and the growth of congestion window stops. For each ACK received, the RoVegas source may send three packets back-to-back.

Let F be the throughput ratio of RoVegas to Vegas (i.e., $F = \frac{\text{throughput of RoVegas}}{\text{throughput of Vegas}}$). In asymmetric networks, we have the throughput relationship of Vegas and RoVegas as follows:

$$1 < F \leq 3, \ \forall k > 1. \tag{3.21}$$

Note that, the throughput of RoVegas contains the overhead induced by the AQT option. So the actual throughput ratio of RoVegas to Vegas should be slightly smaller than F. Equation (3.21) will be further verified by the following performance evaluation.

3.5 Performance Evaluation

In this section, we compare the performance of TCP RoVegas with TCP Vegas by using the network simulator ns-2.1b9a [46]. We show the performance results in backward congestion environments, the bias experiments, the fairness investigations among the competitive connections, and the study of gradual deployment.

The FIFO service discipline is assumed. Every packet of RoVegas is a probing packet. Whenever a throughput of RoVegas is computed, the overhead induced by

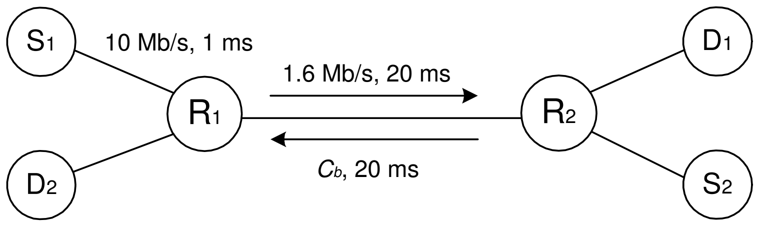


Figure 3.3: A single bottleneck network topology for investigating throughputs of Vegas and RoVegas when the congestion occurs on the backward path.

the AQT option will be subtracted from the throughput. Several VBR sources are used to generate backward traffic. These VBR sources are exponentially distributed ON-OFF sources. During ON periods, the VBR source sends data at 3.2 Mb/s. Unless stated otherwise, the size of each FIFO queue used in routers is 50 packets, the size of data packet is 1 Kbytes, and the sizes of ACKs are 40 and 48 bytes for Vegas and RoVegas respectively. To ease the comparison, we assume that the sources always have data to send.

3.5.1 Throughput Improvement

Improving the throughput of a connection when the congestion occurs in the backward path is one of the design goals of RoVegas. In this subsection, we investigate the throughputs of Vegas and RoVegas in two types of backward congestion. One is the congestion caused by network asymmetry, the other is the congestion caused by additional backward traffic.

The first network topology for the simulations is shown in Fig. 3.3. Sources, destinations, and routers are expressed as S_i, D_i, and R_i respectively. A source and a destination with the same suffix value represent a traffic pair. The bandwidth and propagation delay are 10 Mb/s and 1 ms for each full-duplex access link, 1.6 Mb/s and 20 ms for the connection link from R_1 to R_2, and C_b and 20 ms for the connection link from R_2 to R_1, respectively. C_b is set based on the normalized asymmetric factor k. For example, if $k = 4$ and the size of data packet and ACK are 1 Kbytes and 40 bytes respectively, then C_b is set to 16 Kb/s.

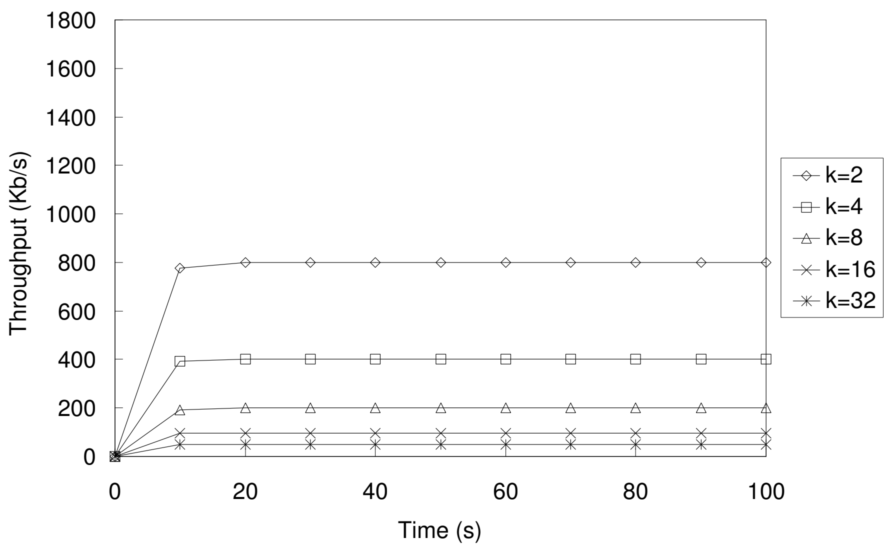


Figure 3.4: Throughput of Vegas in asymmetric networks.

Asymmetric Networks: To evaluate the throughputs of Vegas and RoVegas in asymmetric networks, different values of k are used. A source S_1 of either Vegas or RoVegas sends data packet to its destination D_1. The size of each FIFO queue used in routers is 10 packets. Figures 3.4 and 3.5 exhibit the throughput performance of Vegas and RoVegas in asymmetric networks respectively.

By observing the results shown in Fig. 3.4, with the increasing value of k from 2 to 32, the throughput of Vegas degrades accordingly. As our analysis depicted in Eq. (3.17), the throughput of Vegas in this scenario should be u_f/k (data packets per second). Obviously, the simulation results conform to our previous analysis.

Comparing the results of Fig. 3.5 with that of Fig. 3.4, we can find that the throughput of RoVegas is much greater than that of Vegas. With $k = 2$, RoVegas maintains a high throughput at 1587.2 Kb/s in which the backward congestion seems not existing. The throughput ratios of RoVegas to Vegas in steady state are about 2 and 3 for k=2, 4 and k=8, 16, 32 respectively. Notably, all the simulation results shown in Fig. 3.4 and Fig. 3.5 are consistent with our previous analysis.

Symmetric Network With Backward Traffic: Asymmetric networks should not be the only reason that causes backward congestion. Actually, even in a sym-

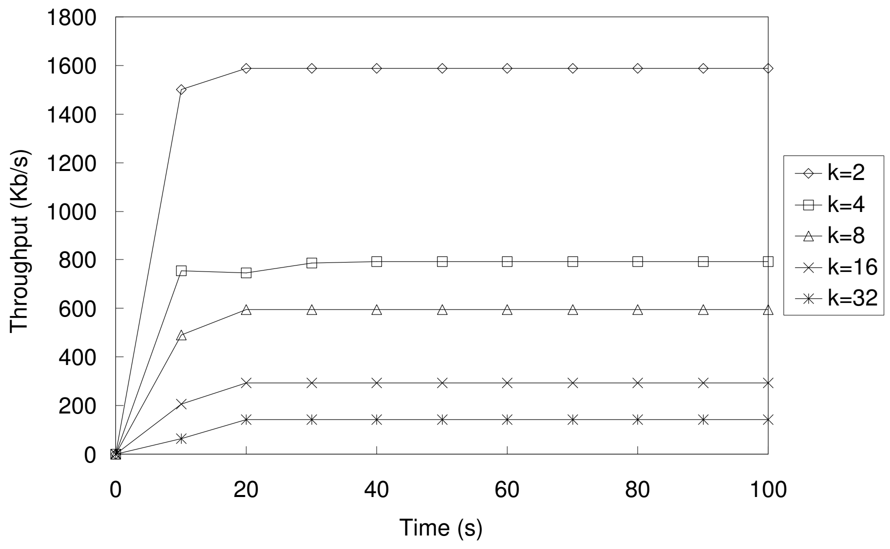


Figure 3.5: Throughput of RoVegas in asymmetric networks.

metric network the backward congestion may still occur. We use a VBR source with 1.44 Mb/s averaged sending rate to examine the throughputs of Vegas and RoVegas separately in a single bottleneck network as shown in Fig. 3.3. The capacity of the backward bottleneck, C_b, is set to 1.6 Mb/s. A source of either Vegas or RoVegas is attached to S_1 and a VBR source is attached to S_2. The S_1 starts sending data at 0 second, while S_2 starts at 50 second. Figure 3.6 depicts the throughput comparison between Vegas and RoVegas.

As shown in Fig. 3.6, when the traffic source is Vegas only (0–50 second), it achieves high throughput and stabilizes at 1.6 Mb/s. However, the performance of Vegas degrades dramatically as the VBR source starts sending data. Although the overhead induced by AQT option slightly lower the throughput of RoVegas (0.8 %) during the preceding 50 seconds, nevertheless, RoVegas maintains a much higher throughput than that of Vegas while the backward congestion occurs. With the inference of the backward VBR traffic, the average throughput of Vegas is 521 Kb/s and RoVegas is 1092 Kb/s. Since we use the same traffic pattern of the VBR source while the throughput of Vegas or RoVegas is examined. Thus there are some synchronized throughput fluctuations between Vegas and RoVegas.

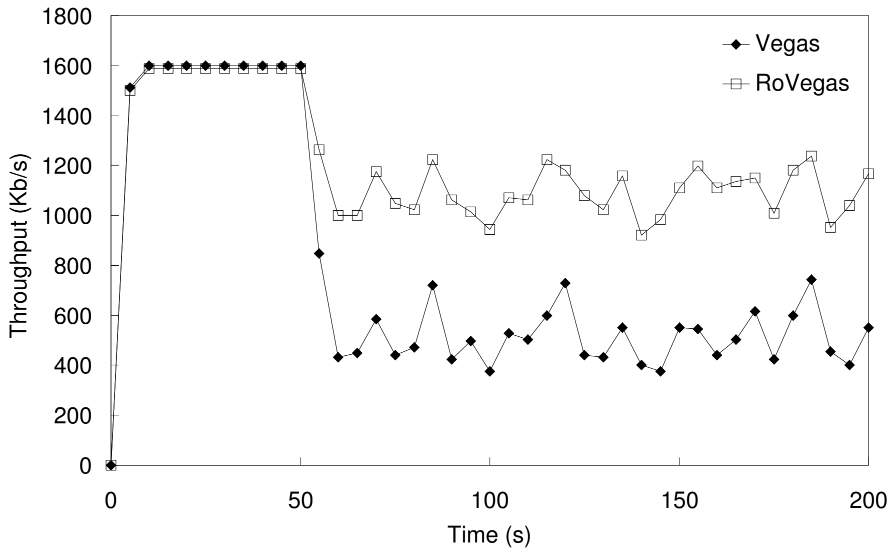


Figure 3.6: Throughput comparison between Vegas and RoVegas with the backward traffic load is 0.9 in the single bottleneck network topology.

To evaluate the average throughputs of Vegas and RoVegas with different backward traffic loads, we set the VBR traffic loads to vary from 0 to 1. The traffic sources are the same as the above descriptions but the sources of either Vegas or RoVegas and VBR start at 0 second. The simulation period is 200 seconds for each sample point. From the simulation results shown in Fig. 3.7, we can find that when the backward traffic load is not zero, RoVegas always achieves a higher average throughput than Vegas. For example, as the backward traffic load is 1, RoVegas achieves a 4.1 times higher average throughput in comparison with that of Vegas.

In the parking lot configuration as shown in Fig. 3.8, we use three VBR sources each with 1.28 Mb/s averaged sending rate to examine the throughputs of Vegas and RoVegas. The bandwidth and propagation delay of each full-duplex access link and connection link are 10 Mb/s, 1 ms and 1.6 Mb/s, 10 ms respectively. The source of either Vegas or RoVegas are attached to S_1, and three VBRs are attached to S_2 to S_4 respectively. The TCP source from either Vegas or RoVegas starts sending data at 0 second, and then three VBR sources from S_2 to S_4 successively enter the network every 100 seconds.

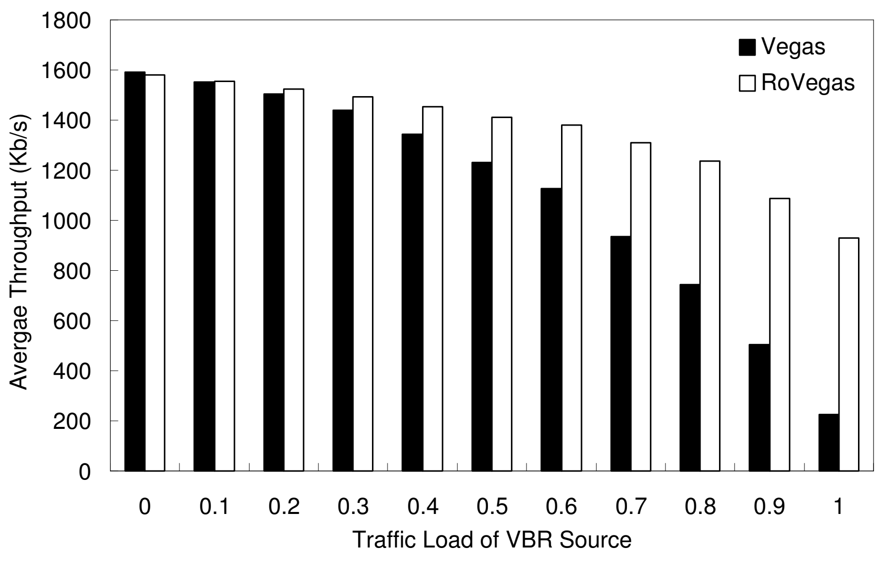


Figure 3.7: Average throughput versus different backward traffic loads for Vegas and RoVegas in the single bottleneck network topology.

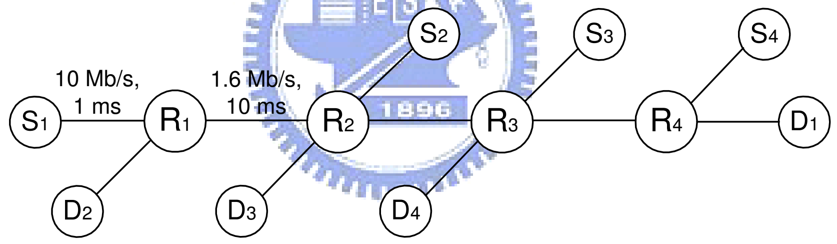


Figure 3.8: A parking lot network topology for investigating throughputs of Vegas and RoVegas when the congestion occurs on the backward path.

From the simulation results presented in Fig. 3.9 we can observe that when the traffic source is TCP only (0–100 second) both Vegas and RoVegas could fully utilize the bandwidth (due to the overhead induced by AQT option, the throughput of RoVegas is slightly lower than Vegas). However, as the VBR sources successively enter the network, Vegas suffers a serious throughput reduction. Under the same environment, RoVegas features a much better throughput performance compared with that of Vegas. The average throughput ratio of RoVegas to Vegas during 100–200, 200–300, and 300–400 second are 2.00, 2.77, and 3.46 respectively.

The average throughputs of Vegas and RoVegas with different backward traffic

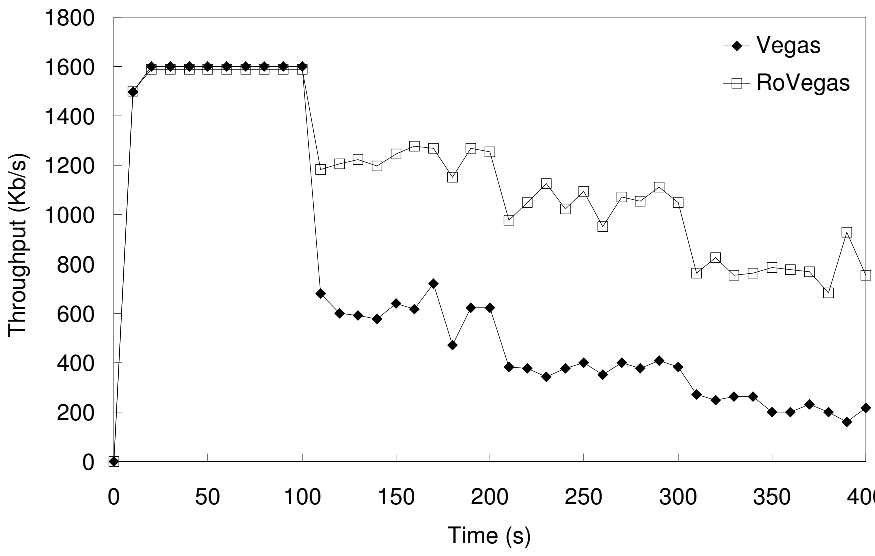


Figure 3.9: Throughput comparison between Vegas and RoVegas with the backward traffic load is 0.8 in the parking lot network topology.

loads in the parking lot network are also examined. The traffic sources of either Vegas or RoVegas and three VBRs start at 0 second. The VBR traffic loads vary from 0 to 1 accordingly. From the simulation results shown in Fig. 3.10 we can find that as the backward traffic load is not zero, RoVegas always achieves a higher average throughput than Vegas, especially when the backward traffic load is heavy. For example, as the backward traffic load is 1, the average throughput ratio of RoVegas to Vegas is 14.09.

Obviously, we have demonstrated that RoVegas significantly improves the connection throughput when the backward path is congested.

3.5.2 Persistent Congestion

As a connection starts when there exist many other connections, the new connection may experience round-trip times that are considerably larger than the actual fixed delay along the path. Thus the *BaseRTT* of this new connection will be larger than it should be. Therefore the new connection will put a larger amount of extra data than its expected amount on the network. This bias may possibly drive the system

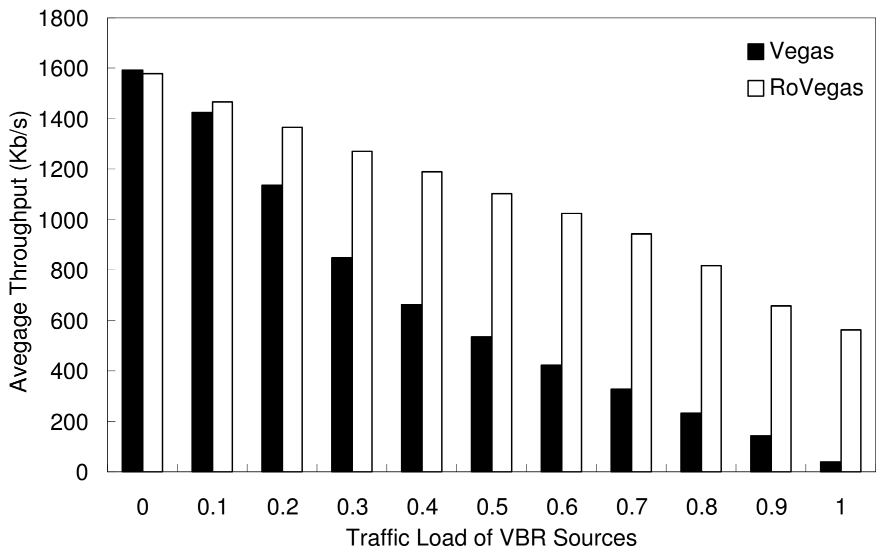


Figure 3.10: Average throughput versus different backward traffic loads for Vegas and RoVegas in the parking lot network topology.

to a persistent congestion.

In this subsection, we study the bias of Vegas through simulation. The simulation network topology is illustrated in Fig. 3.11 in which the bandwidth and propagation delay for each full-duplex link are depicted.

Eight connections of Vegas from S_1 to S_8 successively enter the network every 20 seconds. The α and β of Vegas are set to 1 and 3 respectively. Thus, the amount of extra data for each connection should be kept between 1 and 3 packets. From the results shown in Fig. 3.12, we can observe that when the fourth connection of Vegas joins the network, the queue occupancy of the bottleneck increases to 15 packets. This amount of extra data is larger than the expected maximum amount (12 packets). Even worse, as the eighth connection starts, the queue occupancy of the bottleneck is 40 packets. That is, averagely each connection of Vegas contributes 5 packets to the bottleneck. This situation will become worse and worse when more connections enter the network.

In contrast to the Vegas connections, each RoVegas connection keeps a proper amount of extra data in the bottleneck. When the eighth connection of RoVegas

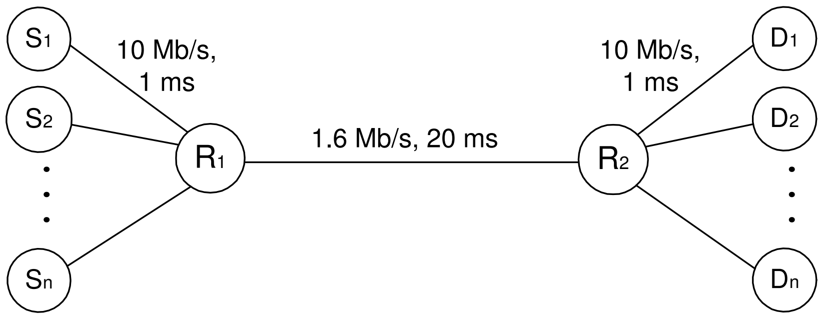


Figure 3.11: Network topology for studying the bias and fairness issues of Vegas and RoVegas.

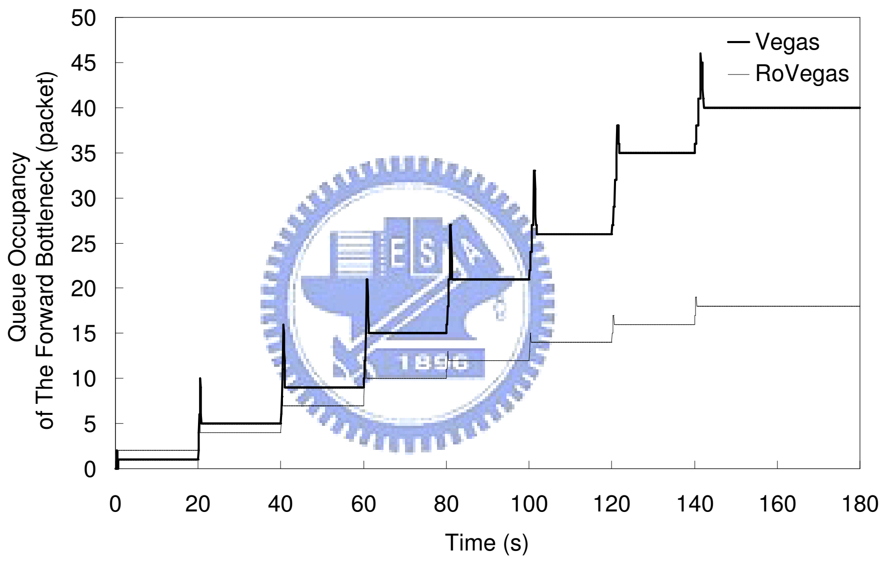


Figure 3.12: Queue occupancy of the forward bottleneck for Vegas and RoVegas.

joins the network, the queue occupancy of the bottleneck is 18 packets. Since each connection of RoVegas is able to derive the fixed delay along the round-trip path, the bias of Vegas no longer exists in RoVegas.

3.5.3 Fairness Enhancement

Fairness is another important issue of Vegas. Although Vegas is not biased against the connections with longer round-trip time like Reno does, there is still unfairness occurred in Vegas. In this subsection, we investigate the fairness metric of Vegas and

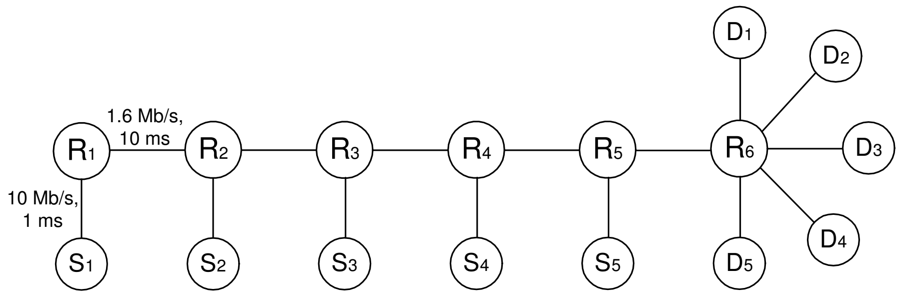


Figure 3.13: Network topology for exploring the fairness issue of Vegas and RoVegas, in which the traffic pairs are featured by different propagation delay.

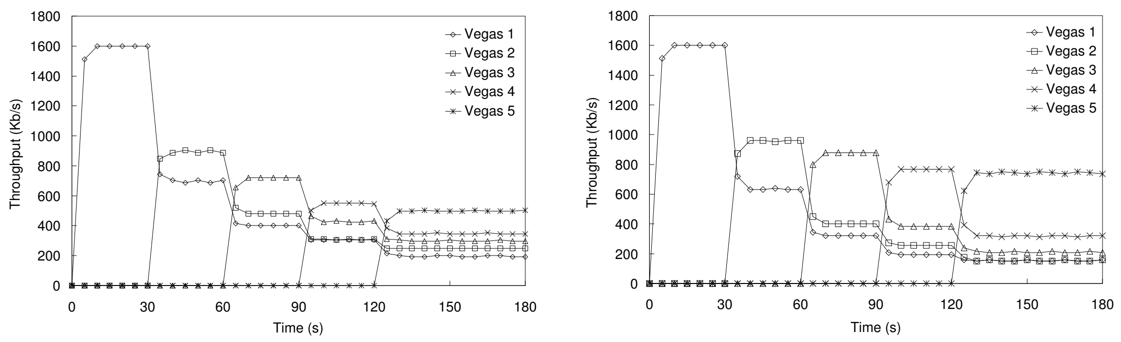
RoVegas. Two network topologies used in the simulations are depicted in Fig. 3.11 and Fig. 3.13.

The first network topology for the simulation is shown in Fig. 3.11, in which all traffic pairs features the same propagation delay. Five connections of either Vegas or RoVegas from S_1 to S_5 successively join the network every 30 seconds. In order to remove the uncertainty induced by the range between α and β, we set α equal to β in two simulation scenarios. Figure 3.14 shows the results of simulations.

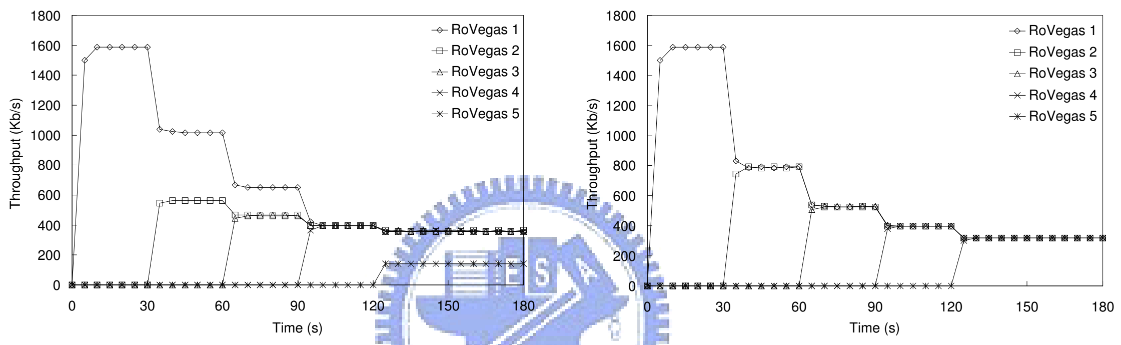
From the simulation results of Vegas presented in Fig. 3.14(a) and Fig. 3.14(b) we can see that no matter the values of α and β are equal or not, connections are unable to share the bandwidth fairly. According to our previous discussion, there are two criteria for achieving the fairness among the competitive connections of Vegas. One is that the measured $BaseRTT$ must be precise enough. The other is that the uncertainty induced by the range between α and β must be removed. Connections in Fig. 3.14(a) do not meet both criteria. Connections in Fig. 3.14(b) do not conform to the first criterion. Therefore, both connections in these two figures are unable to fairly share the bandwidth of the bottleneck.

Observing the results of RoVegas shown in Fig. 3.14(c) and Fig. 3.14(d). Since connections in Fig. 3.14(c) do not meet the second criterion of fairness, the throughputs of these connections are not identical. Finally, connections in Fig. 3.14(d) meet both criteria, hence, all the connections share the bandwidth fairly.

The second network topology for exploring the fairness issue of Vegas and RoVe-



(a) (b)



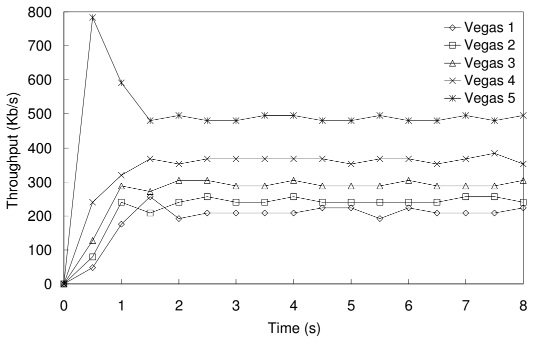
(c) (d)

Figure 3.14: Fairness investigation of Vegas and RoVegas in which connections with the same propagation delay and successively enter the network every 30 second. (a) Vegas ($\alpha = 1,\ \beta = 3$). (b) Vegas ($\alpha = \beta = 2$). (c) RoVegas ($\alpha = 1,\ \beta = 3$). (d) RoVegas ($\alpha = \beta = 2$).

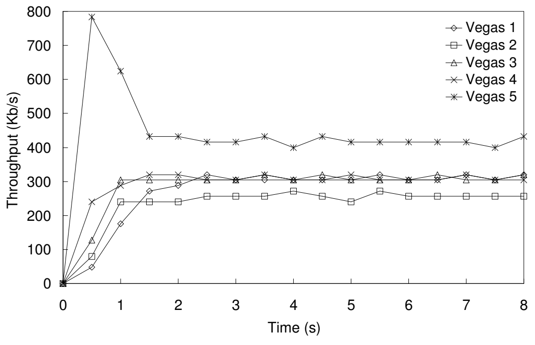
gas is depicted in Fig. 3.13, in which traffic pairs are featured by different propagation delays. The bandwidth and propagation delay of each full-duplex access link and connection link are 10 Mb/s, 1 ms and 1.6 Mb/s, 10 ms respectively.

Five connections of either Vegas or RoVegas from S_1 to S_5 start at the same time. Same as the previous simulations, we set α equal to β in two simulation scenarios to remove the uncertainty induced by the range between α and β. Figure 3.15 represents the results of simulations.

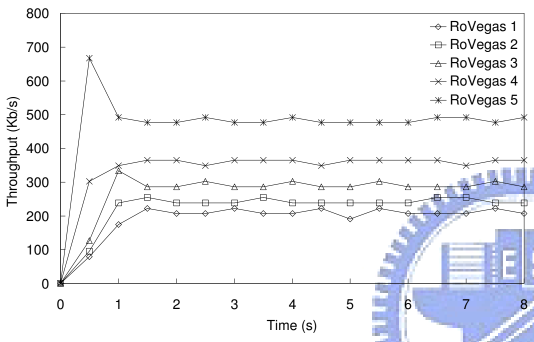
Since the Vegas connections in Fig. 3.15(a) do not meet both criteria of fairness and those in Fig. 3.15(b) do not conform to the first criterion. Therefore, the Vegas



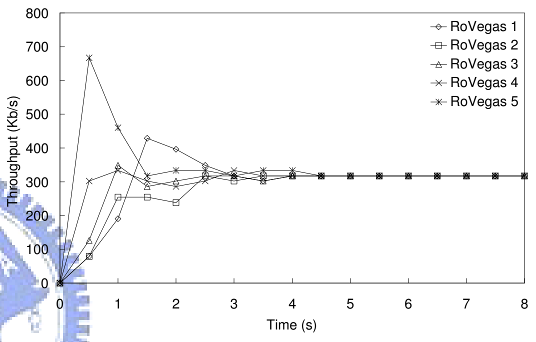
(a)



(b)



(c)



(d)

Figure 3.15: Fairness investigation of Vegas and RoVegas in which connections with different propagation delay and enter the network at the same time. (a) Vegas ($\alpha = 1,\ \beta = 3$). (b) Vegas ($\alpha = \beta = 2$). (c) RoVegas ($\alpha = 1,\ \beta = 3$). (d) RoVegas ($\alpha = \beta = 2$).

connections could not share the bandwidth fairly with each other no matter the values of α and β are equal or not. The RoVegas connections in Fig. 3.15(c) do not obey the second criterion, thus the bandwidth sharing of bottleneck is unfair. However, the RoVegas connections in Fig. 3.15(d) comply with both criteria, and hence the fairness among the competitive connections indeed has been achieved.

3.5.4 Gradual Deployment

It can not be expected that all routers on the Internet are AQT-enabled while the AQT option is a newly defined IP option. To consider the gradual deployment issue, for each ACK received by a RoVegas source, the $BaseRTT$ should be measured as the following pseudo codes:

```
**if** (the ACK is a probing packet)
    $BaseRTT_{temp} = RTT -$ (AQT + AQT-Echo)
    /* where $RTT$ is the newly meausred round-trip time */
    **if** ($BaseRTT_{temp} < BaseRTT$)
        $BaseRTT = BaseRTT_{temp}$
**else** /* the ACK is not a probing packet) */
    **if** ($RTT < BaseRTT$)
        $BaseRTT = RTT$
```

If all bottleneck routers along the round-trip path are not AQT-enabled, RoVegas may behave like Vegas. Since RoVegas cannot obtain the backward queuing time (QT_b) to reduce the impacts of backward congestion, and may not estimate a precise $BaseRTT$ to enhance the fairness and solve the persistent congestion. However, we try to explore whether a single AQT-enabled router on the end-to-end path may achieve the benefits from the RoVegas mechanism.

A parking lot network as shown in Fig. 3.8 is used to examine the throughputs of Vegas and RoVegas separately, here only R_2 is AQT-enabled. Three VBR sources each with 1.28 Mb/s averaged sending to generate backward traffic. A source of either Vegas or RoVegas from S_1 and three VBRs from S_2–S_4 start sending data at 0 second. Despite only one AQT-enabled router R_2 is locating on the routing path, we can find that RoVegas still maintains a higher throughput than that of Vegas, as depicted in Fig. 3.16. The simulation results imply that the proposed mechanism is amenable to gradual deployment for reducing the impacts of backward congestion. This feature may encourage the gradual adoption of RoVegas in the Internet.

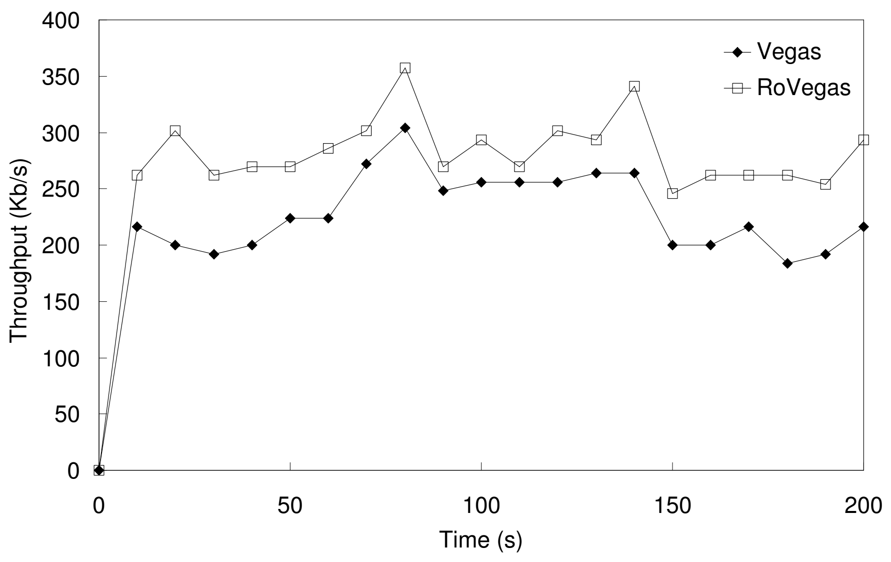


Figure 3.16: Throughput comparison between Vegas and RoVegas for only R_2 is AQT-enabled in the parking lot network.

3.6 Chapter Summary

In this chapter, we carefully examine the problems existed in current TCP Vegas scheme and point out the roughly estimated *BaseRTT* is the problem source of issues of rerouting, unfairness, and persistent congestion. A router-assisted congestion avoidance mechanism, RoVegas, is hence proposed. Comparing with other previous studies, RoVegas provides a more effective way to solve the problems of rerouting and persistent congestion, to enhance the fairness among the competitive connections, and to improve the throughput when congestion occurs on the backward path. The limitations of RoVegas and a gradual deployment scheme are also discussed. Through the results of both analysis and simulation, the effectiveness of RoVegas is shown. However, there is still an issue that needs more attentions. It is enhancing the throughput of Vegas when it performs with Reno head-to-head. Therefore, how to enable compatibility between Reno and RoVegas would be our future work.

Chapter 4

Enhanced Vegas: An End-to-End Approach of TCP Vegas for Asymmetric Networks

Network with high degree of bandwidth asymmetry can adversely impact the performance of feedback-based transport protocols such as TCP. The reason is that even if the network path in the direction of data flow is not congested, congestion in the backward direction can disrupt the flow of feedback.

Several networking technologies with asymmetric network characteristics, such as asymmetric digital subscriber line (ADSL), cable modem, and satellite-based networks, have been widely deployed. These networks feature a large capacity difference between the two communicating directions. This will greatly increase the possibilities of backward path congestion. Both TCP Reno and TCP Vegas may suffer severe performance degradation when the backward path is congested [31, 47], especially for TCP Vegas [31]. Therefore, how to amend the deficiency of TCP Vegas in such situation becomes an important issue.

In the previous chapter, we have proposed a router-assisted solution which can improve the performance of TCP Vegas in the case of backward path congestion. In this chapter, we try to solve the problem under a constraint of keeping TCP in end-to-end semantics. The proposed mechanism, Enhanced Vegas, uses TCP timestamps option to estimate queueing delay on the forward and backward path separately without clock synchronization. Through distinguishing whether congestion occurs in the forward path or not, it significantly advances the connection throughput when the backward path is congested.

The rest of this chapter is organized as follows. Section 4.1 describes the proposed mechanism. Section 4.2 gives some simulation results. Lastly, we conclude this chapter in Section 4.3.

4.1 Enhanced Vegas

To improve the performance in backward path congestion, there are two critical issues of TCP Vegas: (1) how to estimate the backward queueing delay from the measured round-trip time and (2) how to make use of the estimation to adjust the congestion window size. For the issue (2), we adopt the same policy as that in the previous chapter. The *Actual* throughput is redefined as

$$Actual' = \frac{CWND}{RTT - QT(backward)}, \tag{4.1}$$

where *RTT* is the newly measured *RTT*, *QD(backward)* is the backward queuing time, and *CWND* is the current congestion window size. The $Actual'$ is therefore a throughput that can be achieved if there is no backward queuing delay along the path.

To solve the issue (1), we make use of the TCP timestamps option to obtain the backward queuing time. When a connection source sends a packet, it inserts a timestamp into the TCP header. As the destination acknowledges this packet, it copies the forward timestamp and adds a backward timestamp to the ACK packet. Assume there is a TCP source a and its destination b. Let t_{ab} and t_{ba} be the end-to-end trip time of the forward data packet and the backward ACK packet respectively. We can acquire t_{ab} by subtracting the forward timestamp from backward timestamp and t_{ba} by subtracting the backward timestamp from the receiving time in the source (i.e., system time of source a when it receives this ACK packet). Assume td_{ab} is the difference of system clocks in source a and destination b, and $t_{ab}(Min)$ ($t_{ba}(Min)$) denotes the minimum t_{ab} (t_{ba}) that the source a ever measured. The trip time of a packet between two hosts is consisted of the fixed delay time and the queuing delay time. Let $t_{ab}(FD)$ ($t_{ba}(FD)$) denotes the fixed delay time from a (b) to b (a) and $t_{ab}(QD)$ ($t_{ba}(QD)$) represents the forward (backward) queuing time from a (b) to b (a). We have the following equations:

$$\begin{cases} t_{ab} = t_{ab}(FD) + t_{ab}(QD) + td_{ab} \\ t_{ba} = t_{ba}(FD) + t_{ba}(QD) - td_{ab} \end{cases}. \tag{4.2}$$

Since $t_{ab}(Min)$ ($t_{ba}(Min)$), the minimum delay from a (b) to b (a), occurs when the queuing delay $t_{ab}(QD)$ ($t_{ba}(QD)$) approaches zero, therefore we obtain

$$\begin{cases} t_{ab}(FD) = t_{ab}(Min) - td_{ab} \\ t_{ba}(FD) = t_{ba}(Min) + td_{ab} \end{cases}. \tag{4.3}$$

Here, the fixed delay time from a to b is expressed as $t_{ab}(Min) - td_{ab}$. Since t_{ab}, t_{ba}, $t_{ab}(Min)$, and $t_{ba}(Min)$ all can be measured by source a. Thus the forward queuing time $t_{ab}(QD)$ and the backward queuing time $t_{ba}(QD)$ can be derived from Eq. (4.2) and Eq. (4.3) as follows:

$$\begin{cases} t_{ab}(QD) = t_{ab} - t_{ab}(Min) \\ t_{ba}(QD) = t_{ab} - t_{ba}(Min) \end{cases}. \tag{4.4}$$

The $t_{ba}(QD)$ is just the *QD(backward)* that we need in Eq. (4.1). Furthermore, the *BaseRTT* can be more precisely estimated by the sum of $t_{ab}(FD)$ and $t_{ba}(FD)$ (i.e., the sum of $t_{ab}(Min)$ and $t_{ba}(Min)$). To Avoid the unnecessary reduction of TCP congestion window size, the rule for congestion window adjustment of Enhanced Vegas is same as Eq. (3.5).

The proposed mechanism estimates *BaseRTT* and queuing time on both directions based on tracking the minimum end-to-end trip time (i.e., $t_{ab}(Min)$ and $t_{ba}(Min)$). However, if the clock speed is different between the source and the destination, the accumulated time differences caused by clock skews may result in

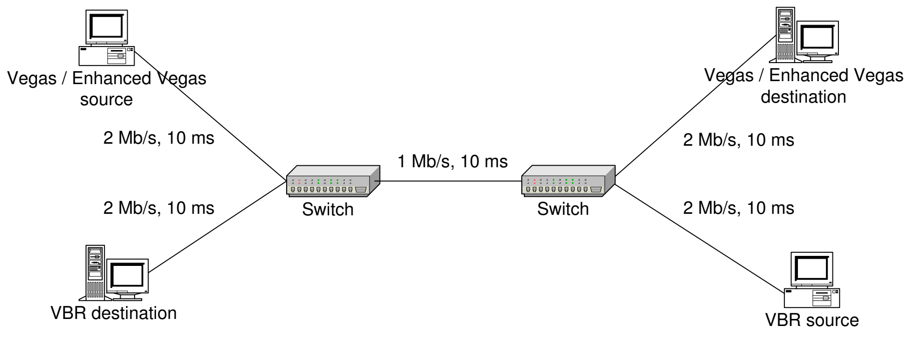


Figure 4.1: Network configuration for the simulations.

incorrect measurement of minimum end-to-end trip time. Certain efficient algorithms have been proposed to estimate clock skews from network delay measurements [48, 49]. Let C_a and C_b be the clock speed of source a and destination b respectively. The clock ratio is denoted by ρ, $\rho = C_b/C_a$. Then we have the following equations to adjust the minimum end-to-end trip time:

$$\begin{cases} t_{ab}^{x+\Delta x}(Min) = t_{ab}^{x}(Min) - \Delta x(1-\rho) \\ t_{ba}^{x+\Delta x}(Min) = t_{ba}^{x}(Min) + \Delta x(1-\rho) \end{cases}, \quad \Delta x \geq 0 \tag{4.5}$$

where $t_{ab}^{x+\Delta x}(Min)$ ($t_{ba}^{x+\Delta x}(Min)$) and $t_{ab}^{x}(Min)$ ($t_{ba}^{x}(Min)$) denote the $t_{ab}(Min)$ ($t_{ba}(Min)$) on the time $(x + \Delta x)$ and x respectively.

4.2 Performance Evaluation

We perform the simulations using ns-2 [46] to compare the throughputs between Vegas and our proposed Enhanced Vegas. A VBR source is used to generate the backward traffic. This VBR source is an exponentially distributed ON-OFF source. During ON periods, the VBR source sends data at 2 Mb/s. Several VBR sources with different average sending rates are used to examine our mechanism. All parameters of both Vegas and Enhanced Vegas are the same. Without loss of generality, the packet size is set at 1 Kbytes. To ease the comparison, we assume that the sources always have data to send. The network configuration we used is shown in Fig. 4.1, in which the bandwidth and delay of each full duplex link are depicted.

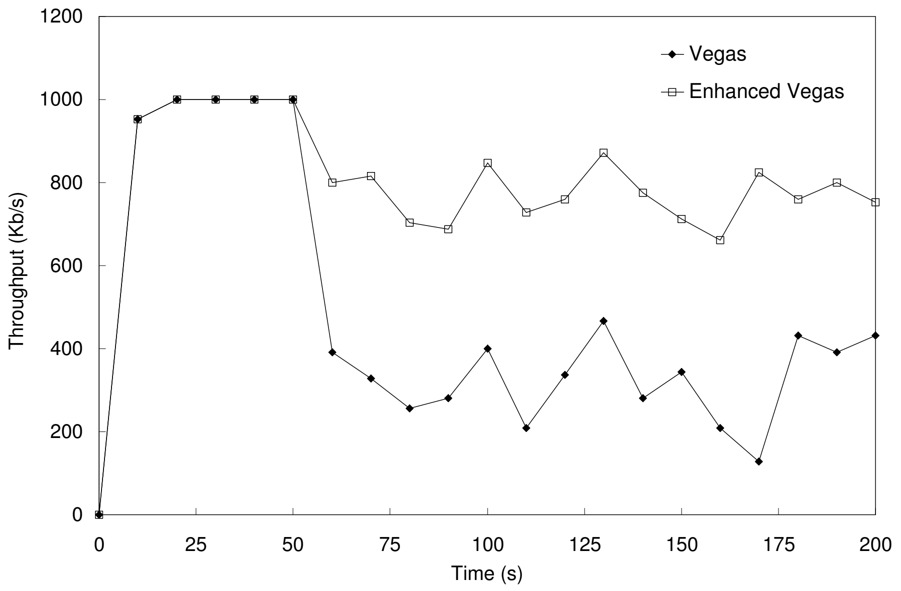


Figure 4.2: Throughput comparison between Vegas and Enhanced Vegas with the backward traffic load 0.9.

In the first simulation, we use a VBR source with 900 Kb/s average sending rate to examine the throughput of Vegas and Enhanced Vegas separately. A source from either Vegas or Enhanced Vegas starts sending data at 0 second, while VBR source starts at 50 second. By observing the result in Fig. 4.2, when traffic source is Vegas only, it can achieve high throughput and stabilize at 1,000 Kb/s until the VBR source starts sending data. However, it shows that performance of Vegas drops dramatically as the VBR traffic starts. On the contrary, Enhanced Vegas maintains a much higher throughput than Vegas. During the active period of VBR source, the average throughput of Vegas is 325 Kb/s and Enhanced Vegas is 767 Kb/s. Since the traffic pattern of the VBR source keeps the same when the throughput of Vegas or Enhanced Vegas is examined. Thus there are some synchronized fluctuations of throughput for Vegas and Enhanced Vegas. The simulation results demonstrate that the proposed scheme significantly improves the throughput of Vegas when the backward path is congested.

In the second simulation, we evaluate the average throughput of Vegas and Enhanced Vegas with different backward traffic loads separately. Sources of either

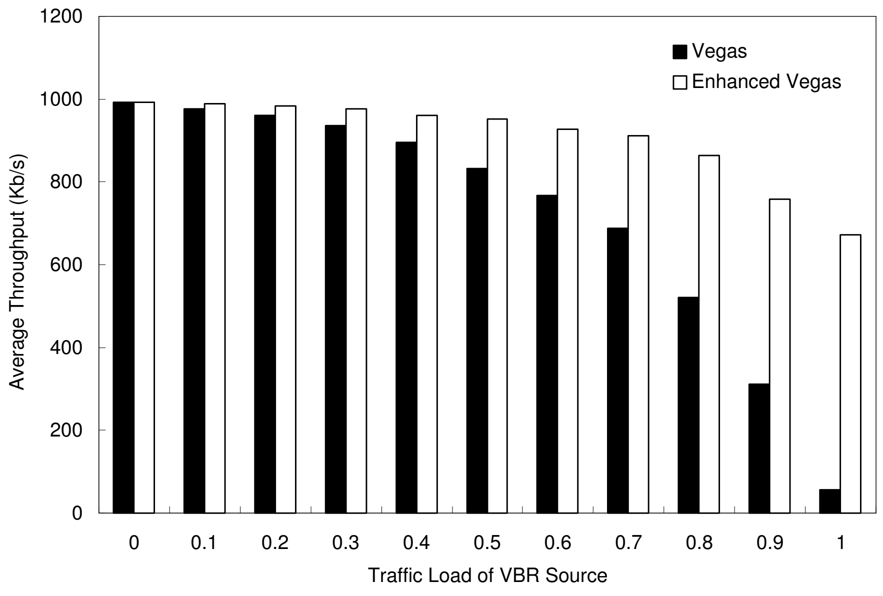


Figure 4.3: Average throughput versus backward traffic load for Vegas and Enhanced Vegas.

Vegas or Enhanced Vegas and VBR start at 0 second. The VBR traffic loads vary from 0 to 1. The simulation period is 200 seconds for each sample point. From the simulation results shown in Fig. 4.3, we can find that the Enhanced Vegas obtains a much higher average throughput than TCP Vegas, especially when the backward traffic load is heavy. For example, when the backward traffic load is 1, Enhanced Vegas achieves a 12 times higher average throughput than that of Vegas.

4.3 Chapter Summary

In this chapter, we propose an end-to-end approach of TCP Vegas for asymmetric networks. Comparing with other studies [29, 30, 47, 26], Enhanced Vegas provides a much easier way to improve the connection throughput when the backward path is congested. The simulation results show the effectiveness of our proposed mechanism. Nevertheless, clock skew issue is still a problem of Enhanced Vegas, such as the convergence speed of the clock ratio measurement. In the future research, we will try to eliminate the clock issues from Enhanced Vegas.

Chapter 5

Quick Vegas: Improving Performance of TCP Vegas for High Bandwidth-Delay Product Networks

A critical design issue of TCP is its congestion control that allows the protocol to adjust the end-to-end communication rate based on the available bandwidth of the bottleneck link. However, TCP congestion control may function poorly in high bandwidth-delay product (BDP) networks because of its slow response with large congestion windows. In this chapter, we propose an improved version of TCP Vegas called Quick Vegas, in which we present an efficient congestion window control algorithm for a TCP source. Our algorithm is based on the increment history and estimated amount of extra data to update the congestion window intelligently. Simulation results show that Quick Vegas significantly improve the performance of connections as well as remain fair and stable when the bandwidth-delay product increases.

The rest of this chapter is organized as follows. Section 5.1 addresses the problems of TCP Reno and TCP Vegas in high BDP networks. In Section 5.2, related work is reviewed. Section 5.3 describes the proposed Quick Vegas. Section 5.4 presents the simulation results. Lastly, we summarize this chapter in Section 5.5.

5.1 Problem Statements

Many Internet applications use TCP as its transport protocol. The behavior of TCP is therefore tightly coupled with the overall Internet performance. TCP performs at an acceptable efficiency over today's Internet. However, theory and experiments show that, when the per-flow product of bandwidth and latency increases, TCP becomes inefficient [50]. This will be problematic for TCP as the bandwidth-delay product (BDP) of Internet continues to grow.

TCP Reno takes packet loss as an indiction of congestion. In order to probe available bandwidth along the end-to-end path, it periodically creates packet losses by itself. It is well-known that TCP Reno may feature poor utilization of bottleneck link under high BDP networks. Since TCP Reno uses additive increase - multiplicative decrease (AIMD) algorithm to adjust its window size, when packet losses occur, it cuts the congestion window size to half and linearly increases the congestion window until next congestion event is detected. The additive increase policy limits TCP's ability to acquire spare bandwidth at one packet per round-trip time (RTT). The BDP of a single connection over very high bandwidth links may be thousands of packets, thus TCP Reno might waste thousands of $RTTs$ to ramp up to full utilization of the link. For example, the time of a connection to converge to an optimal bandwidth value can take the order of minutes in a high BDP network which with 1 Gb/s available bandwidth and 100 ms round-trip time. Thus, if TCP's convergence mechanism is too sluggish, TCP will eventually become a performance bottleneck itself.

Unlike TCP Reno which uses binary congestion signal, packet loss, to adjust its window size, TCP Vegas adopts a more fine-grained signal, queuing delay, to avoid congestion. It can successfully prevent the problems caused by AIMD algorithm. However, in high BDP networks, Vegas tends to prematurely stop the exponentially-increasing slow-start phase and enter the slower congestion avoidance phase until it reaches its equilibrium congestion window size [32]. As a result, a new Vegas connection may experience a very long transient period and throughput suffers. In addition, the availability of network resources and the number of competing users may vary over time unpredictably. It is sure that the available bandwidth is not varied linearly [51]. Since Vegas adjusts its congestion window linearly in the congestion avoidance phase, this prevents Vegas from quickly adapt to the changing environments.

5.2 Related Work

Several studies have been made to improve the connection performance over high-speed and long-delay links. These approaches can be divided into two categories. One is simpler and needs only easily-deployable changes to the current protocols, for example, HighSpeed TCP [52] and Scalable TCP [53]. The other needs more complex changes with a new transport protocol, or more explicit feedback from the routers, examples are XCP [50] and QuickStart [54].

HighSpeed TCP involves a subtle change in the congestion avoidance response function to allow connections to capture available bandwidth more readily. Scalable TCP is similar to HighSpeed TCP in that the congestion window response function for large windows is modified to recover more quickly from loss events and hence reduce the penalty for probing the available bandwidth.

The same as TCP Reno, both HighSpeed TCP and Scalable TCP use packet loss as an indication for congestion. This causes periodic oscillations in the congestion window size, round-trip delay, and queue length of the bottleneck node. These drawbacks may not be appropriate for emerging Internet applications [11, 12].

XCP is a new transport protocol designed for high BDP networks. It separates the efficiency and fairness policies of congestion control, and enables connections to quickly make use of available bandwidth. However, because XCP requires all routers along the path to participate, deployment feasibility is a concern.

QuickStart is a mechanism that uses IP options for allowing an end host to request a high initial sending rate along the end-to-end path. The feasibility of QuickStart relies on the cooperation of the end host and routers. Again, the difficulty in deployment is an issue to be overcome.

5.3 Quick Vegas

In high BDP networks, the equilibrium congestion window size is larger than that of small BDP networks. Besides, network resources and competing users may vary over time unpredictably. In order to react faster and better to high BDP networks, the window adjustment algorithm should be more aggressive than it has been.

TCP Vegas updates its congestion window linearly in the congestion avoidance phase, it is too sluggish for a high BDP network. Depending on the information given by the estimated extra data, it is worth to try a more aggressive strategy. The details of Quick Vegas is described as follows.

For the increment of congestion window, Quick Vegas keeps the history to guide the changes of window size. Since there is no direct knowledge of current available bandwidth, Quick Vegas records the number of consecutive increments due to $\Delta < \alpha$ and refers to this value as *succ*. Whenever the congestion window should be increased due to $\Delta < \alpha$, it is updated as follows:

$$CWND = CWND + (\beta - \Delta) \times succ. \tag{5.1}$$

Thus the congestion window size will be increased by $(\beta - \Delta)$ at the first estimation of $\Delta < \alpha$, and by $(\beta - \Delta) \times 2$ at the next consecutive estimation of $\Delta < \alpha$, and so on. The *succ* will be reset whenever $\Delta \geq \alpha$. The idea is that if the increment was successful it might be the case that there is enough bandwidth and it is worthwhile to move to a more aggressive increasing strategy. However, to ensure that the congestion window will not be increased too fast, Quick Vegas can at most double the size of congestion window for every estimation of $\Delta < \alpha$.

For the decrement of congestion window, Quick Vegas uses the difference of Δ

and $(\alpha + \beta)/2$ as the amount of decrement for every estimation of $\Delta > \beta$. The decrement rule can be expressed as follows:

$$CWND = CWND - (\Delta - \frac{\alpha + \beta}{2}). \tag{5.2}$$

Since the estimated amount of extra data gives us a good suggestion of how many extra data are beyond the ideal volume that should be kept in the network pipe. Therefore, Quick Vegas subtracts the excess amount from the congestion window directly.

Compared with TCP Vegas, the window adjustment algorithm of Quick Vegas is more aggressive. To ensure that the estimation of extra data is valid and the adjustment does not overshoot the real need, Quick Vegas adjusts its congestion window only every other *RTT*, just like the updating in slow-start phase. Besides, in order to achieve a higher fairness between the competing connections, Quick Vegas intends every connection to keep an equal amount, that is $(\alpha + \beta)/2$, of extra data in the network pipe. If the estimated amount of extra data is between α and β, Quick Vegas will adjust its congestion window linearly toward the ideal amount. The window adjustment algorithm of Quick Vegas can be presented as the following pseudo codes:

```
if ($\Delta > \beta$)
    $CWND = CWND - (\Delta - \frac{\alpha+\beta}{2})$
    $incr = 0;\ succ = 0$
else if ($\Delta < \alpha$)
    $succ = succ + 1$
    if ($(\beta - \Delta) \times succ > CWND$)
        $incr = 1$
    else
        $incr = \frac{\beta - \Delta}{CWND} \times succ$
else if ($\Delta > \frac{\alpha+\beta}{2}$)
    $CWND = CWND - 1;\ incr = 0;\ succ = 0$
else if ($\Delta < \frac{\alpha+\beta}{2}$)
```

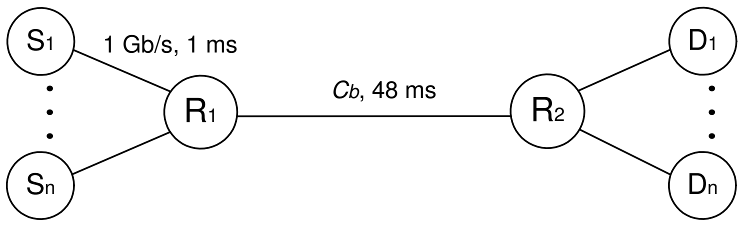


Figure 5.1: Network configuration for the simulations.

```
        incr = $\frac{1}{CWND}$; $succ = 0$
    **else** /* $\Delta == \frac{\alpha+\beta}{2}$ */
        $incr = 0$; $succ = 0$
```

To reduce the bursty effect of increment, the *incr* is served as the increment amount of congestion window after each ACK is received by a Quick Vegas source.

5.4 Performance Evaluation

We use the network simulator ns-2.1b9a [46] to execute the performance evaluation. All parameter settings of both Vegas and Quick Vegas are the same. Especially, $\gamma = 1$, $\alpha = 2$, and $\beta = 4$ that are same as in [23]. Unless stated otherwise, the buffer size in routers is large enough so that packet loss is negligible. The sizes of data packets and ACKs are 1 Kbytes and 40 bytes respectively. To ease the comparison, we assume that the sources always have data to send.

The network configuration for the simulations is shown in Fig. 5.1. Sources, destinations, and routers are expressed as S_i, D_i, and R_i respectively. A source and a destination with the same subscript value represent a traffic pair. The bandwidth and propagation delay are 1 Gb/s and 1 ms for each full-duplex access link, and C_b and 48 ms for the full-duplex connection link between R_1 and R_2. The C_b is set based on the need of simulation scenarios.

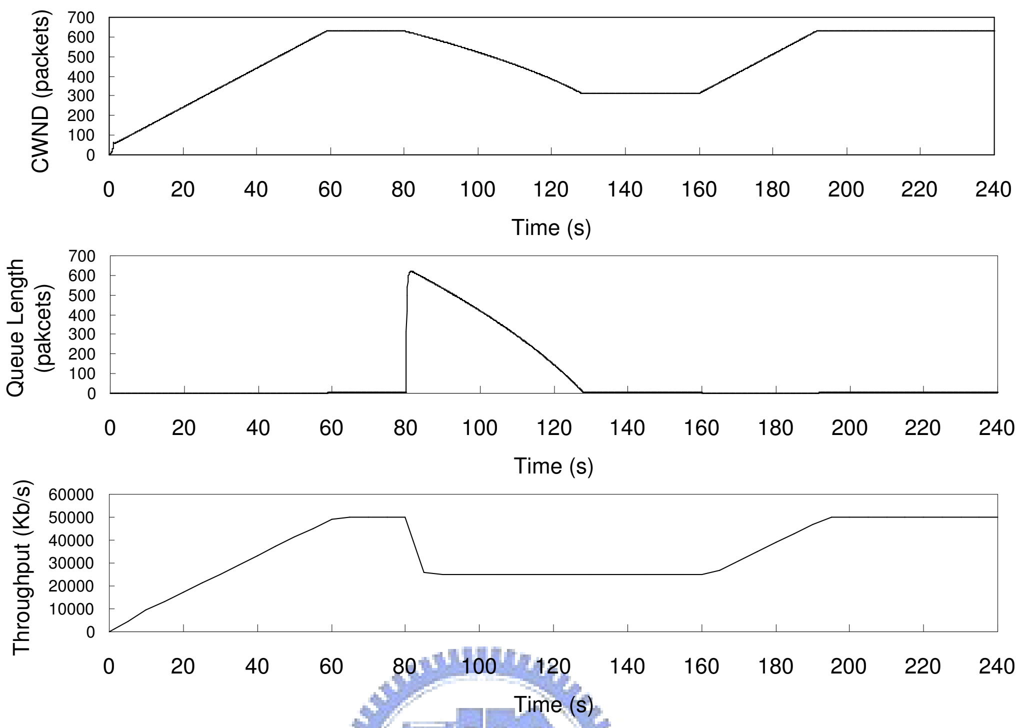


Figure 5.2: Basic behavior of TCP Vegas.

5.4.1 Basic Behavior

In this subsection, we compare the basic behavior between TCP Vegas and Quick Vegas in the aspects of congestion window size, queue length, and throughput. The bottleneck capacity C_b is set at 50 Mb/s. A TCP connection of either Vegas or Quick Vegas from S_1 to D_1 starts sending data at 0 second and a CBR traffic flow from S_2 to D_2 with 25 Mb/s rate starts at 80 second and stops at 160 second. The objective of the simulation scenario is to explore how fast for a new connection can ramp up to equilibrium and how fast a connection can converge to a steady state as the available bandwidth is changed. Figure 5.2 and 5.3 exhibit the basic behavior of Vegas and Quick Vegas respectively.

By observing the congestion window evolution shown in Fig. 5.2 we can find that the transient period for a new Vegas connection is quite long. Vegas prematurely stop the exponentially-increasing slow-start phase at 1.9 second and enter the linearly-increasing congestion avoidance phase. It takes 59 seconds to reach equi-

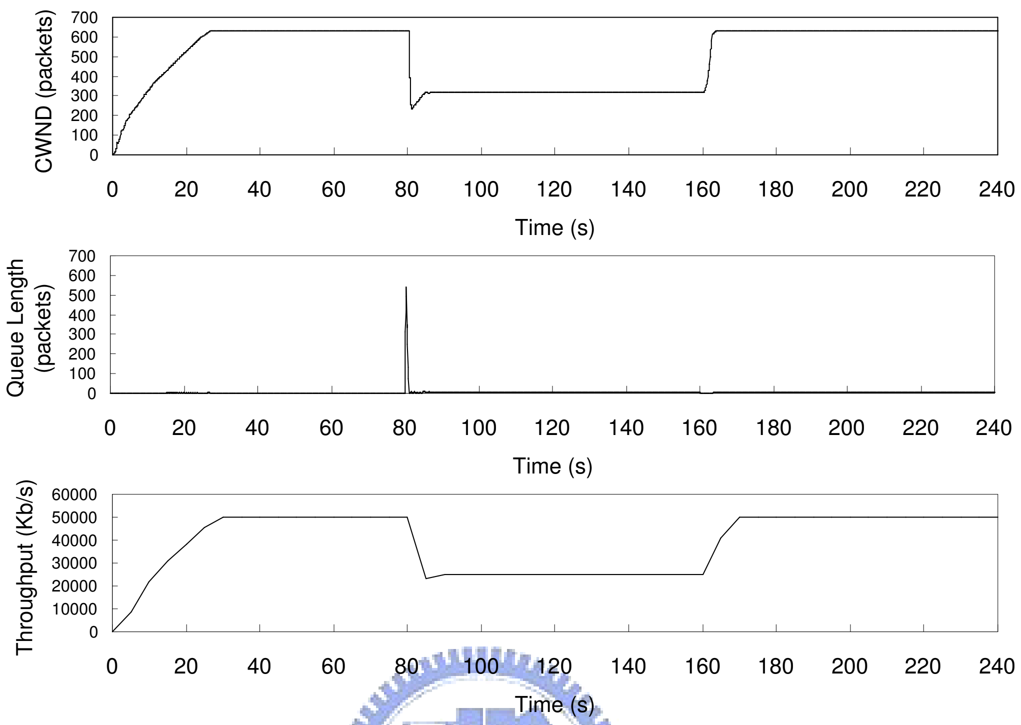


Figure 5.3: Basic behavior of Quick Vegas.

librium. When the available bandwidth is halved at 80 seconds, Vegas takes 47.9 seconds to converge to a new steady state. As the available bandwidth is doubled at 160 second, there is a 31.8 seconds transient period for Vegas.

The queue length at bottleneck shown in Fig. 5.2 also reveals that Vegas can not quickly adapt to the changed bandwidth. When the available bandwidth is halved at 80 seconds, the queue is built up quickly. The maximum queue length is 620 packets and it also takes 47.9 seconds for Vegas to recover the normal queue length.

In comparison with Vegas, Quick Vegas reacts faster and better, as shown in Fig. 5.3. The ramp up time of Quick Vegas is 27 seconds, and it takes 6.7 and 3.9 seconds to converge as the available bandwidth is halved and doubled respectively. Note that due to the bursty nature of a new TCP connection, the estimation of extra data will be disturbed [32]. The consecutive increment number (*succ*) may not be accumulated to a large number. Therefore, the ramp up time can not be greatly improved as compared with the convergence period of the available bandwidth when it is halved or doubled.

The queue length at bottleneck shown in Fig. 5.3 also exhibits that Quick Vegas can quickly adapt to the changed bandwidth. When the available bandwidth is halved at 80 seconds, the built up queue is quickly removed. The maximum queue length is 540 packets, which is also smaller than that of Vegas (620 packets).

Based on the simulation results of throughput shown in Fig. 5.2 and 5.3, obviously, Quick Vegas has a better performance than Vegas when a connection is either in the beginning (0–60 second) or the available bandwidth is doubled (160–190 second). Although the throughput of Quick Vegas (23.2 Mb/s) is smaller than that of Vegas (25.9 Mb/s) at 85 second, however, Vegas has a larger maximum queue length. In the simulation we assume a large queue size at bottleneck so that packet losses will not occur. In more realistic scenarios, the queue size of bottleneck may not be large enough. A larger maximum queue length means a higher probability of packet losses occur, which in turn would cause a lower throughput.

5.4.2 Convergence Time

With high BDP networks, the transient period of TCP can greatly affect overall performance. In this subsection, we use a metric "convergence time" [32] to capture the transient performance of TCP. Convergence time indicates how many *BaseRTTs* are required to reach a new stable state.

The traffic sources are same as that in the previous subsection. The bottleneck capacity C_b is varied for different BDP. At some time instant, the CBR traffic source starts or stops sending packets to halve or double the available bandwidth, respectively.

Figure 5.4 presents the convergence time for a new connection to reach equilibrium. Theoretically, Quick Vegas doubles the increment rate that results in logarithm convergence time in contrast to Vegas which converges linearly. However, due to the bursty nature of a new TCP connection, the *succ* may not be consecutively accumulated. The convergence time of Quick Vegas is about half of that of Vegas

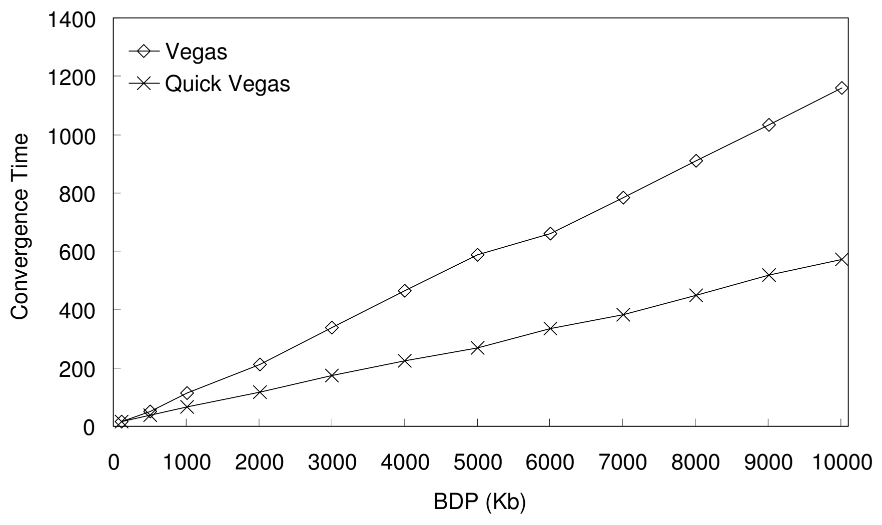


Figure 5.4: Convergence time of new connections.

as the BDP is greater than 500 Kb.

Figure 5.5 and 5.6 display the convergence time as the available bandwidth is halved and doubled respectively. Obviously, Quick Vegas greatly improves the transient performance of connection in both scenarios as compared to Vegas.

5.4.3 Utilization, Queue Length, and Fairness

The simulations presented in this subsection intend to demonstrate link utilization of the bottleneck, fairness between the connections, and queue length at the bottleneck buffer where connections join and leave the network. The bottleneck capacity C_b is set at 1 Gb/s. Connections C_1–C_{20}, C_{21}–C_{40}, and C_{41}–C_{60} start at 0, 100, and 200 second respectively. Each connection with the same active period is 300 seconds. The size of bottleneck buffer is 1250 packets.

Table 5.1 shows the bottleneck link utilization in which connections of Vegas, Quick Vegas, and New Reno [13] are evaluated. When Vegas connections enter the empty network, it takes 55 seconds to reach equilibrium, while Quick Vegas takes 25 seconds. Since severe packet losses occur in the exponentially increasing slow-start phase, the link utilization of New Reno during 0–25 second is quite low (0.170).

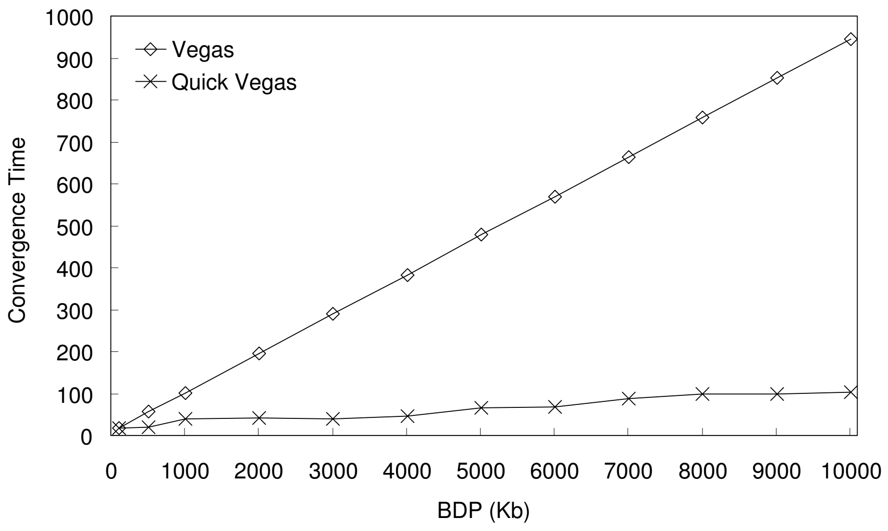


Figure 5.5: Convergence time of connections when available bandwidth is halved.

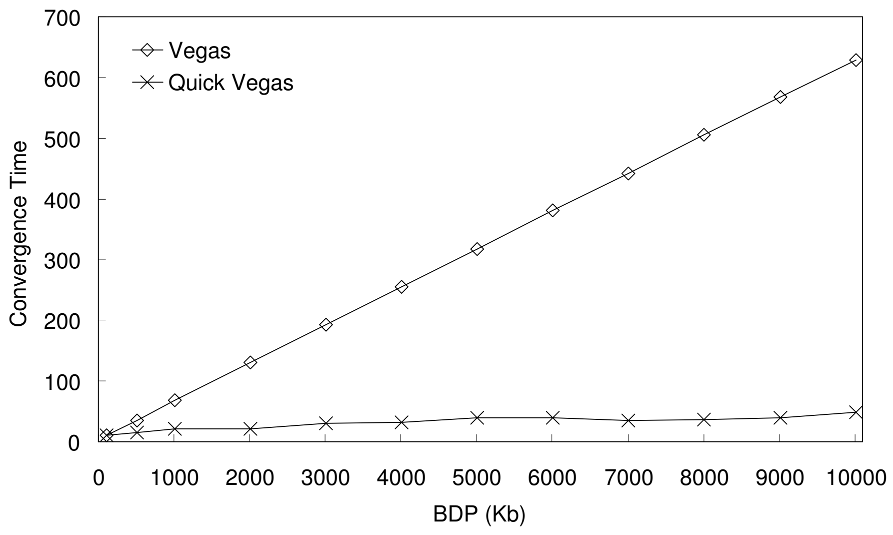


Figure 5.6: Convergence time of connections when available bandwidth is doubled.

Table 5.1: Link utilization of the bottleneck.

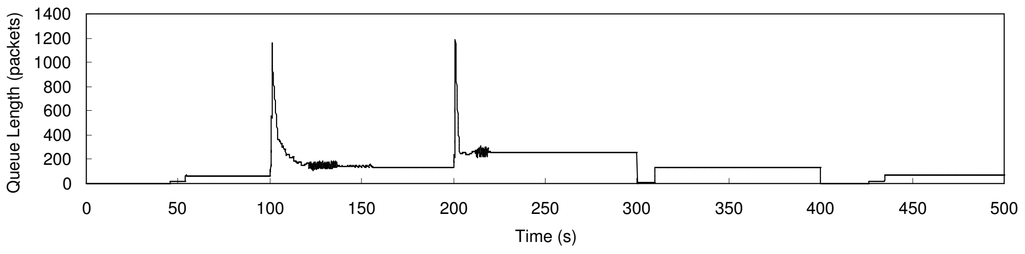
Time (s)	**Vegas**	**Quick Vegas**	**New Reno**
0–25	0.335	0.632	0.170
25–55	0.780	1.000	0.647
55–300	1.000	1.000	0.788
300–310	0.853	0.872	0.706
310–400	1.000	1.000	0.797
400–420	0.607	0.785	0.665
420–435	0.886	1.000	0.921
435–500	1.000	1.000	0.781

As the new connections C_{21}–C_{40} and C_{41}–C_{60} enter the network at 100 and 200 second, both Vegas and Quick Vegas can fully utilize the bottleneck link. However, by observing the queue status shown in Fig. 5.7 we can find that Quick Vegas features a smaller maximum queue length (1017 packets) as compared with that of Vegas (1188 packets). A smaller maximum queue length implies that Quick Vegas can adapt to the changing network environment much quickly and prevent packet losses much effectively.

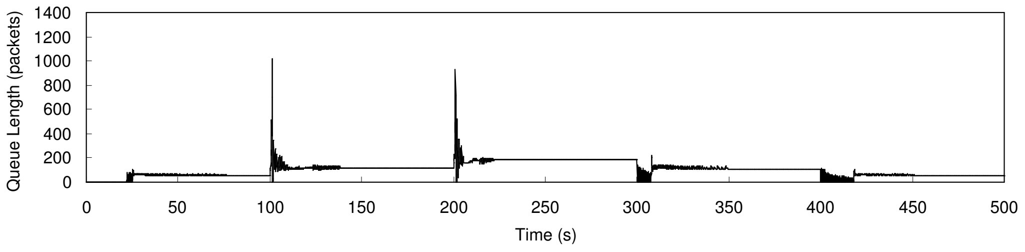
When the available bandwidth increases substantially due to connections C_1–C_{20} and C_{21}–C_{40} leave the network at 300 and 400 second, the remaining connections of Quick Vegas can also quickly adapt to the newly available bandwidth. As a result, the bottleneck link utilization of Quick Vegas during 300–310 and 400–435 second are higher than that of Vegas.

Different from Vegas or Quick Vegas, New Reno can not maintain a stable queue length as shown in Fig. 5.7(c). Since New Reno needs to create packet losses by itself to probe the available bandwidth along the path. Therefore, packet losses occur periodically and certain amount of throughput is wasted. It is obvious that New Reno can not maintain such high link utilization like that of Vegas or Quick Vegas as depicted in Table 5.1.

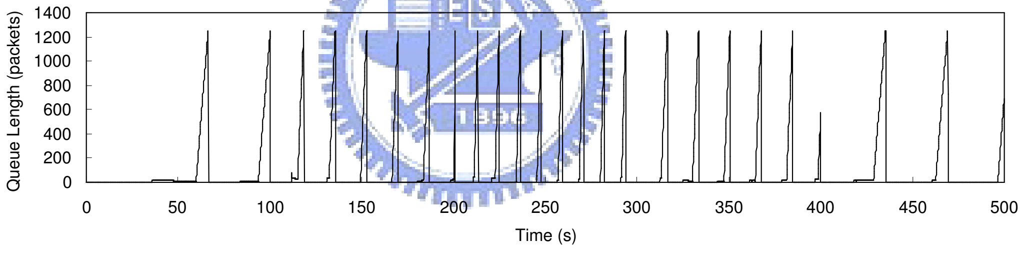
The average queue length at the bottleneck of Vegas and Quick Vegas are ex-



(a) Vegas



(b) Quick Vegas



(c) New Reno

Figure 5.7: Queue status of the bottleneck.

hibited in Table 5.2. Due to the ability of quickly grabing the available bandwidth, Quick Vegas has slightly higher average queue length during 0–100 and 400-500 second. However, when the network is congested (100–300 second), Quick Vegas features not only smaller maximum queue length but also lower average queue length.

To evaluate the fairness among connections, we use the fairness index proposed in [55]. Given a set of throughput $(x_1, x_2, \ldots, x_n)$, the fairness index of the set is defined as:

$$f(x) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} {x_i}^2}. \tag{5.3}$$

The value of fairness index is between 0 and 1. If the throughput of all connections

Table 5.2: Average queue length (packets).

Time (s)	0–100	100–200	200–300	300–400	400–500	0–500
Vegas	28	174	268	122	47	128
Quick Vegas	45	121	187	107	51	102

Table 5.3: Fairness index.

Time (s)	0–100	100–200	200–300	300–400	400–500
Active Connections	C_1–C_{20}	C_1–C_{40}	C_1–C_{60}	C_{21}–C_{60}	C_{41}–C_{60}
Vegas	0.969	0.941	0.972	0.987	0.999
Quick Vegas	0.965	0.960	0.980	0.977	0.981

is the same, the index will take the value of 1.

Table 5.3 shows the fairness index of Vegas and Quick Vegas for each 100 seconds time period. Although Quick Vegas adopts a more aggressive strategy to adjust the congestion window size, however, Quick Vegas keeps the similar fairness index values as that of Vegas. The simulation result suggests that Quick Vegas preserves the good characteristic of fairness as that of original TCP Vegas.

5.5 Chapter Summary

In this chapter, we propose an improved version of TCP Vegas named Quick Vegas for high bandwidth-delay product networks. Quick Vegas presents a modification of congestion window update algorithm at the connection source. Based on the increment history and estimated amount of extra data, Quick Vegas adopts a more intelligent and aggressive way to adjust its window size. Simulation results show that Quick Vegas reacts faster and better to changing environments and therefore improves the overall performance.

For the future research of Quick Vegas, we will try to develop a new slow-start mechanism. Due to the bursty nature of new TCP connections, the estimation of extra data is always disturbed. It makes connections in high BDP networks tend to stop the slow-start phase too early and lead to a longer transient period. With a suitable slow-start mechanism, Quick Vegas can be excellent in high BDP networks.

Chapter 6

RedVegas: Performance Improvement of TCP Vegas over Heterogeneous Networks

Nowadays IP network has become the dominant paradigm for all networking environments. The significant cause of packet losses in such heterogenous networks is no longer limited to network congestion. Traditional TCP interprets every packet loss as caused by congestion which may be not the case in the current Internet. Misinterpretation of wireless random loss as an indication of network congestion results in TCP slowing down its sending rate unnecessarily. In this chapter, we propose a new variant of TCP Vegas named RedVegas. By using the innate nature of Vegas and congestion indications marked by routers, RedVegas may detect random packet losses precisely. Through the packet loss differentiation, RedVegas reacts appropriately to the losses, and therefore the throughput of connection over heterogeneous networks can be significantly improved.

The rest of this chapter is organized as follows. Section 6.1 addresses the problem of TCP in heterogeneous networks and explains why we propose a solution for TCP Vegas. In Section 6.2, the related work is reviewed. Section 6.3 describes the proposed RedVegas. Section 6.4 presents the simulation results. Lastly, we conclude this chapter in Section 6.5.

6.1 Problem and Motivation

Owing to the great advancement in wireless networking technology and new emerging applications, providing ubiquitous mobile Internet access becomes increasingly important. The well-known problem in providing TCP congestion control over heterogeneous networks (wired/wireless environment) is that current TCP implementations rely on packet loss as an indicator of congestion. In the wired networks, a congestion is indeed a likely reason of packet loss. On the other hand, a noisy, mobile, and fading radio channel is the most likely cause of loss in the wireless networks. The effective bit error rates in wireless networks are significantly higher than that in wired networks. Since TCP does not have any mechanism to differentiate between congestion losses and wireless random losses, the latter may cause a severe throughput degradation.

The purpose of congestion control is to dynamically adapt the end-to-end transmission rate of a connection to the currently available capacity. TCP performs at an acceptable efficiency over the traditional wired networks where packet losses are caused by network congestion. However, when TCP observes random losses, it misinterprets such losses and reduces its window size, this causes the reduction of throughput unnecessarily. Therefore, TCP's performance drops rapidly in the presence of frequent random losses [56, 34].

The throughput deterioration problem of TCP over wireless networks has been addressed in [57, 58, 59, 60, 61, 62, 63, 64, 65, 66]. However, part of solutions are designed especially for TCP Reno [57, 58], TCP Vegas has not been given equal attention.

TCP Vegas exhibits many superior features than the most widely deployed TCP Reno [9, 10, 11, 12, 22, 23, 24]. Even in wireless Multi-hop networks, Vegas also keeps better performance than that Reno can achieve [67]. It has the potential to provide a more stable and efficient network environment. For this reason, we propose a random error detection mechanism (RedVegas) for TCP Vegas and expect that the proposed RedVegas may drive TCP Vegas to a real success.

6.2 Related Work

Several approaches have been proposed to optimize TCP for wireless networks. We summarizes them as follows.

Link Layer Mechanisms: Link layer mechanisms try to improve the quality of the lossy wireless link. They hide the characteristics of the wireless link from the transport layer and try to solve the problem at the link layer. The intuition behind link layer mechanisms is to treat the problem as local, and to solve it locally. Forward error correction (FEC) and automatic repeat request (ARQ) are two typical techniques used in link layer mechanisms [59, 60].

End-to-End Approaches: In the end-to-end approaches [57, 58, 61], the TCP source attempts to handle the losses in a way that improves the performance of a connection which runs on wireless networks. In wireless environment, the major cause of packet losses is not limited to network congestion, thus some rules are used to infer the cause of packet losses. Based on the inferences, a source may react appropriately to the losses. These approaches maintain the end-to-end semantics of TCP and thus do not need any extra support from the intermediate hops along the path.

Base Station Schemes: If only the last hop connecting to a mobile host is a wireless link, a TCP-aware agent can be run on the base station to improve the performance of connections.

Split connection approaches [62, 63, 64] isolate mobility and wireless related problems from the existing network protocols. This is done by splitting the TCP connection between the mobile host and the fixed host into two separate connections: a wired connection between the fixed host and the base station, and a wireless connection between the base station and the mobile host. In this way the wired connection does not need any change in existing software on the fixed hosts, and the wireless connection can use a specialized mobile protocol to provide better performance.

The snoop protocol [65] introduces a snoop module at a base station to monitor every packet transmitted through the connection in either direction. By caching recently transmitted TCP packets sent to a mobile host and monitoring the associated acknowledgment packets returning to the source, the snoop module can quickly resend a cached copy of the lost packet to the mobile host. The snoop protocol hides the packet loss from the fixed host and hence avoids the unnecessary invocation of congestion control mechanism.

Explicit loss notification [66] is a mechanism runs on the base station to watch passing TCP packets to deduce whether there is a packet loss due to corruption. It sets a special bit in the returning acknowledgment packets to notify the source of a connection that the recent packet loss may be resulted from corruption instead of congestion. Based on the notification, the source may react appropriately to the loss and therefore improve the performance of connections.

6.3 RedVegas

The issue of packet losses differentiation can be divided into two parts: (1) how to distinguish between congestion loss and random loss, and (2) how to make use of the information to refine the congestion window adjustment process. The success of our RedVegas relies on the cooperation of the end-hosts and routers. It assumes that the routers are capable of marking packets when congestion occurs.

The key idea of RedVegas is described as follows. Since Vegas always attempts to keep the amount of extra data between two thresholds α and β in the network pipe. In fact, the extra data is the major part of in-flight packets that stays in the bottleneck buffer. Whenever an arriving packet is dropped due to the congestion in the bottleneck link, it is very likely that some preceding packets belonging to the same connection have been queued in the bottleneck buffer. Specifically, the amount of these preceding packets should be between α and β. If the router detects congestion and marks a congestion indication bit on all packets in the current buffer, the consecutive packets of the same connection prior to the dropped packet will very likely be marked. Based on the ACKs of the marked packets, a RedVegas source may infer the cause of packet loss accordingly.

Like the Explicit Congestion Notification (ECN) mechanism [43], RedVegas uses two bits in IP header as the congestion indication field (CI) and one bit in TCP header as the CI-echo flag. The first bit of CI represents the CI-capable transport (CICT) of a packet and the second bit serves as congestion experienced flag. When a RedVegas source wants to send a packet, it sets the CICT bit. Whenever a router drops a packet due to congestion, it will mark all packets' congestion experienced flag of those packets queued in the buffer with CICT bit set. Every time a RedVegas destination acknowledges a received packet, it checks whether the congestion experienced flag in the received packet is marked or not. If so, it will set the CI-echo flag in the ACK packet. As a RedVegas source receives an ACK, it checks the CI-echo flag. If the flag is set, the sequence number[1] of the newest acknowledged packet will be recorded in the variable *NCSEQ*.

As in Vegas, RedVegas has three ways to detect packet loss, a triple-duplicate ACK, a fine-grained timeout, and a coarse-grained timeout. Whenever a packet loss is identified by a triple-duplicate ACK or a fine-grained timeout, RedVegas will try to infer the cause of loss. When a packet loss is detected and the difference between the sequence number of the lost packet and *NCSEQ* is no greater than β, RedVegas assumes that the loss is a congestion loss. Otherwise, random loss is inferred. If the losses are identified by a coarse-grained timeout, RedVegas does not intend to infer the cause of losses. Since the losses are severe and the information carried by the received ACKs may be passe. RedVegas leaves this part of algorithm to be intact.

In the proposed scheme, the ACKs with CI-echo flag of both the preceding and succeeding packets of a lost packet may assist the inference. To ease the illustration,

[1]For a practical TCP implementation, a sequence number identifies the byte in the stream of data. In this paper, to ease the description of the proposed mechanism, we use a sequence number to represent the packet.

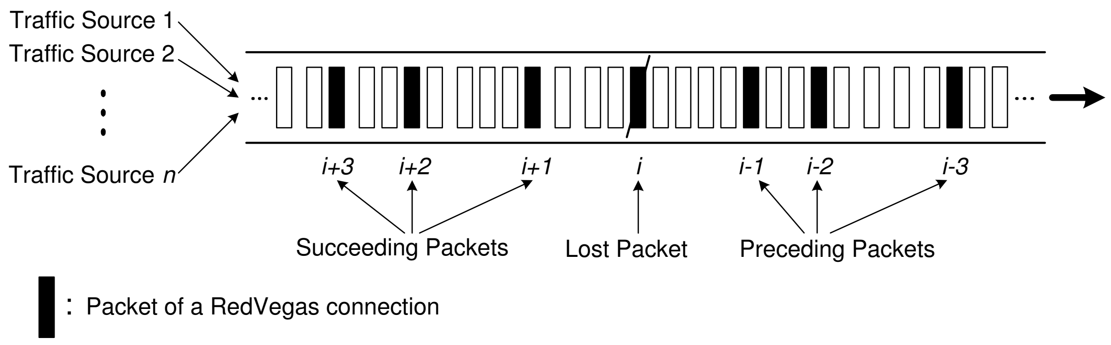


Figure 6.1: Snapshot of the consecutive packets in network pipe.

we assume the part of consecutive packets of a connection that pass through the network pipe are packet (*i-3*) to packet (*i+3*), as shown in Fig. 6.1. The lost packet (*i*) is eventually identified by a triple-duplicate ACK, the value of β is three, and all ACKs of the consecutive packets successfully reach the source. If a congested router marks any preceding packet from packet (*i-3*) to packet (*i-1*) or succeeding packet from packet (*i+1*) to packet (*i+3*)[2], the loss packet (*i*) will not be judged as a random loss. Since the difference between the sequence number of the lost packet and the *NCSEQ* will be equal to or smaller than β, a congestion loss is inferred.

In some situations, the ACKs with CI-echo flag of the preceding packets may fail to reach the connection source. Using the CI information carried by the ACKs of the succeeding packets may help RedVegas to reduce the possibility of misinterpreting a congestion loss as a random loss. Since the misinterpreting a congestion loss as a random loss may result in a wrong reaction that will violate the objective of congestion control. RedVegas intends to keep a high accuracy of random loss detection.

After the lost packet is recovered, RedVegas will adjust the congestion window size according to the loss differentiation. A connection needs not to reduce the sending rate if the loss was not caused by congestion. Thus, if the lost packet is detected as a random loss, the *CWND* will not be changed, that is, the *CWND*

[2]The marking of succeeding packets is not guaranteed by RedVegas, it depends on the network status after the lost packet (*i*) is dropped.

will be equal to the congestion window size when the loss is detected. However, if a congestion loss is perceived, the same window-adjustment mechanism as that in Vegas will be adopted.

TCP congestion control is mainly based on the feedback of ACKs. The control procedure will be triggered whenever an ACK is received by the connection source. Figure 6.2 illustrates the detailed procedure of Vegas/RedVegas as it receives an ACK. Shady blocks are for RedVegas especially. The description of variables used in Fig. 6.2 is shown in Table 2.2.

6.4 Performance Evaluation

In this section, we compare the performance among the three TCP variants (Vegas, Reno, and RedVegas) and study the effectiveness of distinguishing losses in RedVegas by using the network simulator ns-2.1b9a [46]. The FIFO service discipline is assumed. All parameter settings of both Vegas and RedVegas are the same. Especially, $\alpha = 1$ and $\beta = 3$. The size of each FIFO queue used in routers is 16 packets, the sizes of data packets and ACKs are 1 Kbytes and 40 bytes respectively. To ease the comparison, we assume that the sources always have data to send.

The network configuration for the simulations is shown in Fig. 6.3, in which the bandwidth and delay of each full duplex link are depicted. Sources, destinations, and routers are expressed as S_i, D_i, and R_i respectively. The link between R_2 and D_1 is a wireless link on which we assume all random losses occur. Typically, wireless links are subject to fading phenomena, which results in random loss in bursty manner. However, if random losses appear in bursty manner, it is easy for RedVegas to recognize them. Since RedVegas takes every packet loss to be random loss when there is no packet to be marked. Therefore, we use a more complicated case, uniformly distributed loss model, to evaluate RedVegas. In wireless environments, if the bit error is uniformly distributed, the larger a packet is, the more likely the packet will be corrupted [68, 69]. Keeping this in mind, when we apply a random loss rate to data packets, we always set the proportional random loss rate to ACKs.

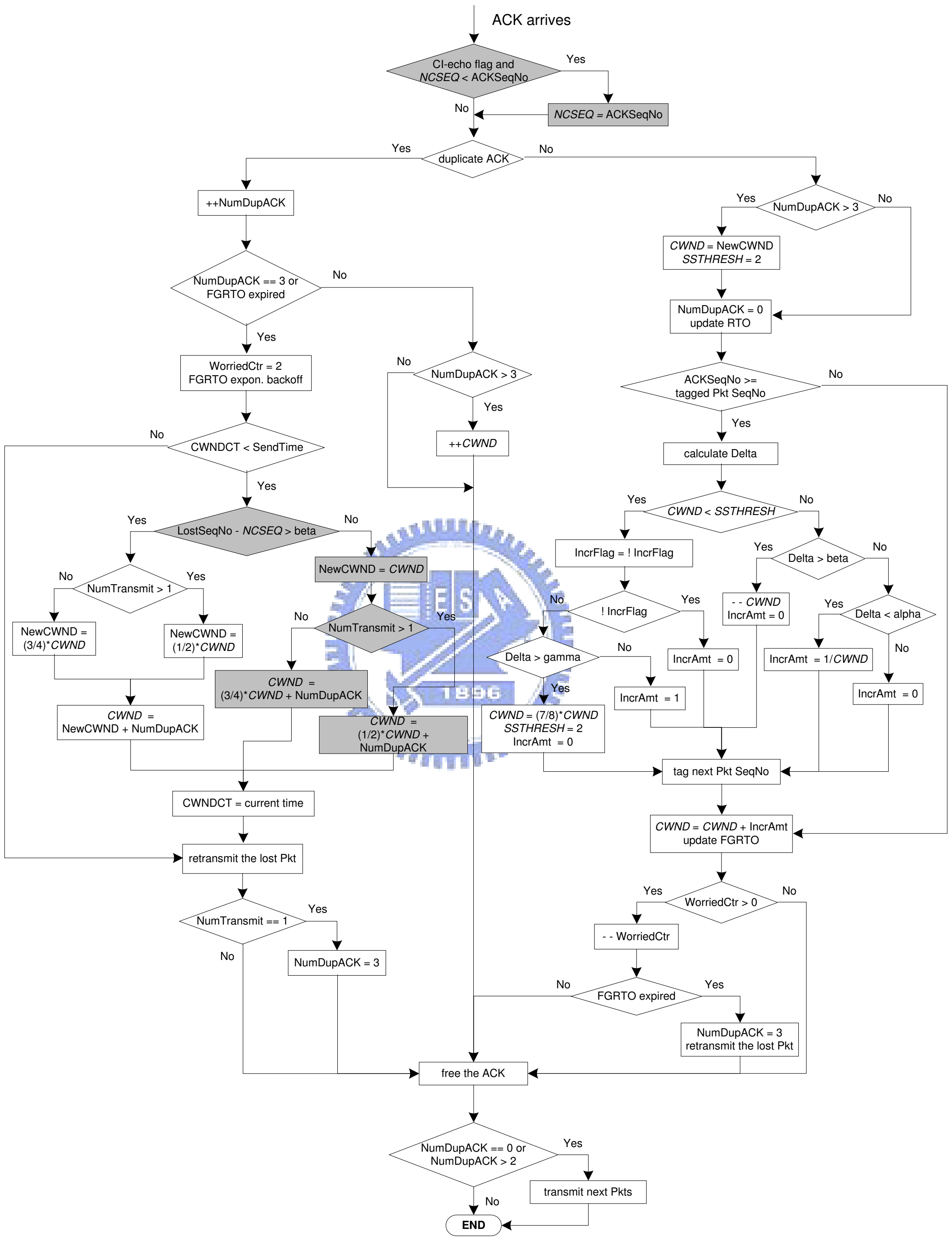


Figure 6.2: Flowchart to illustrate the procedure of Vegas/RedVegas as it receives an ACK. Shady blocks are for RedVegas especially.

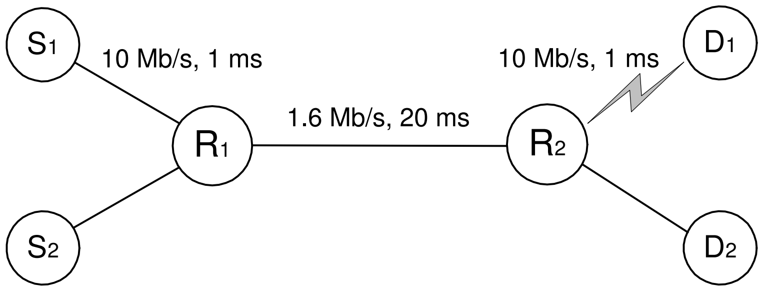


Figure 6.3: Network configuration for the simulations.

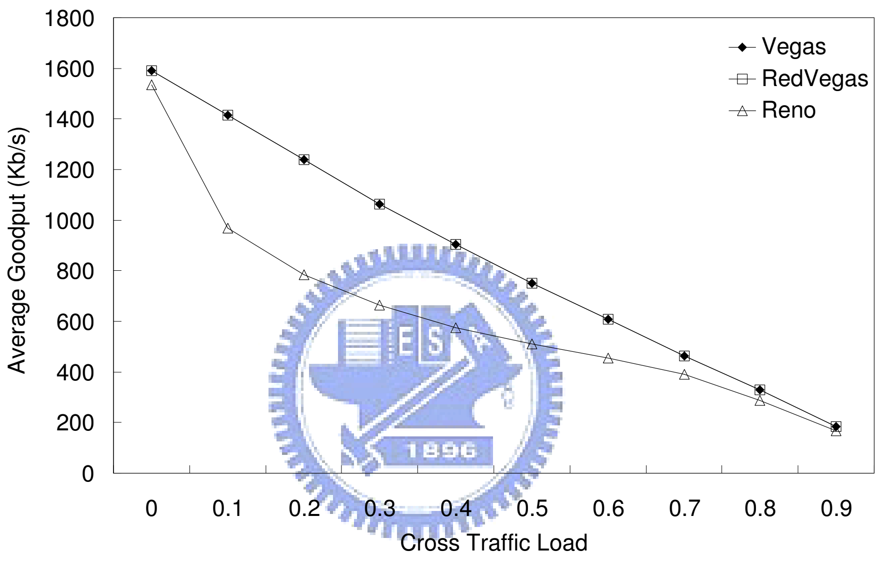


Figure 6.4: Average goodput versus cross traffic load for the three TCP variants.

A TCP connection is established from S_1 to D_1, and a VBR source is used to generate cross traffic from S_2 to D_2. Complying with the study of Internet simulating [70], the VBR source is a Pareto distribution ON-OFF source with shape parameter 1.5. During ON periods, the VBR source sends data at 1.6 Mb/s. In the following simulations, unless stated otherwise, the execution time of each sample point is 10 hours.

6.4.1 Basic Behavior

The design goal of RedVegas is to improve performance for TCP Vegas over heterogeneous networks. It is obvious that if the packet losses are due to congestion,

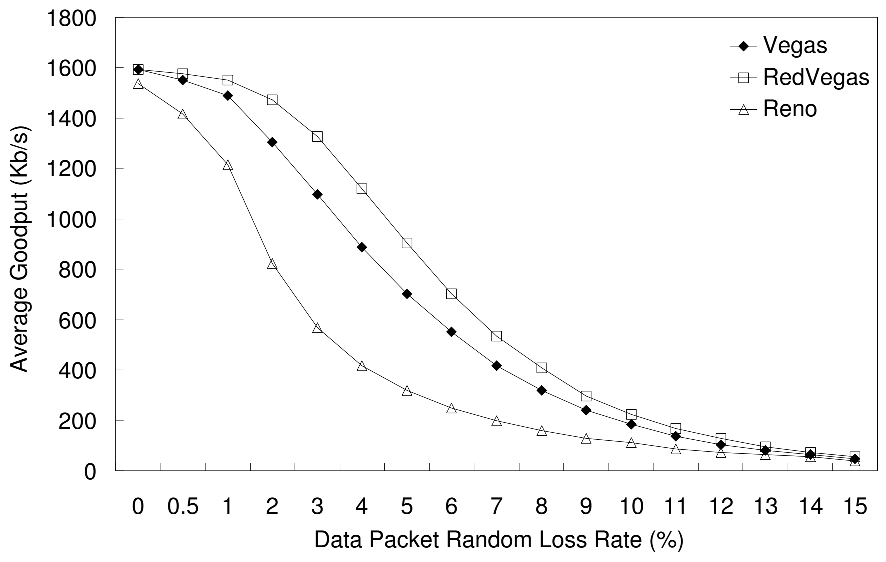


Figure 6.5: Average goodput versus data packet random loss rate for the three TCP variants.

the behaviors of Vegas and RedVegas should be the same. In this subsection, we examine the average goodputs among the three TCP variants with different cross traffic loads. The difference between the throughput and goodput is that the later only counts those packets effectively received once. Each TCP variant is examined separately and the results can be found in Fig. 6.4.

With the increasing cross traffic load, the average goodputs of the three TCP variants degrade accordingly. Note that the goodputs of Vegas and RedVegas are always identical. The results demonstrate that the behaviors of Vegas and RedVegas are the same when there is no random loss. It implies that RedVegas does not misinterpret congestion loss as random loss in such simulation scenarios. From the simulation results, Vegas always surpasses Reno in goodput with different cross traffic loads, this conforms to the previous studies [22, 23, 24].

6.4.2 Impact of Random Loss

In this subsection, we compare the average goodputs among the three TCP variants with different random loss rates. No cross traffic is introduced in the simulations. By

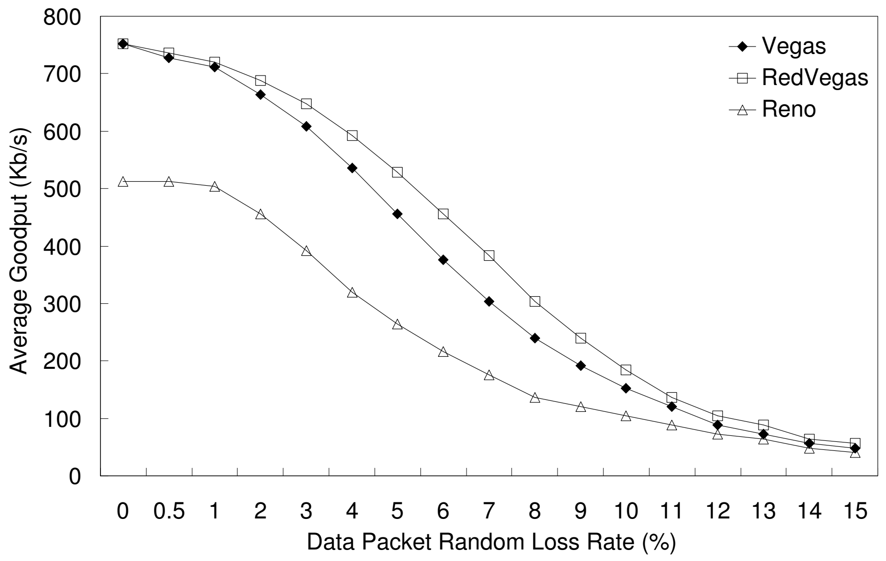


Figure 6.6: Average goodput versus data packet random loss rate for the three TCP variants with the cross traffic load is 0.5.

observing the results shown in Fig. 6.5, both Vegas and RedVegas can fully utilize the bottleneck link when the random loss rate is zero. However, Reno can not maintain such high goodput with the same condition. Since Reno needs to create packet losses by itself to probe the available bandwidth along the path. Therefore, certain amount of goodput is lost.

With the increasing random loss rate, the goodput improvement of RedVegas becomes obvious. When the random loss rate is 6 %, the goodput of RedVegas is about 2.84 times higher than that of Reno. When the random loss rate is between 3 % and 13 %, RedVegas always keeps a goodput improvement of larger than 20 % in comparison with Vegas. Notably, with 7 % random loss rate, the goodput improvement is up to 28.9 %. Moreover, the goodput improvements are 16.7 % and 40 % as compared with Vegas and Reno respectively in a severe radom loss rate (15 %).

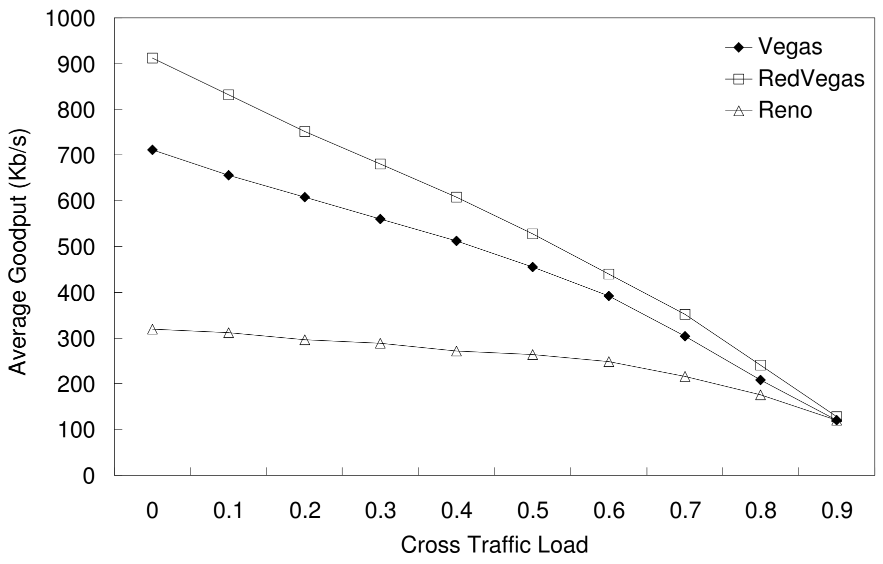


Figure 6.7: Average goodput versus cross traffic load for the three TCP variants with the data packet random loss rate is 0.05.

6.4.3 Impact of Random Loss and Cross Traffic

TCP connections over heterogeneous networks may experience both random losses and congestion losses. In this subsection we introduce random losses and cross traffic into the simulation to examine the goodputs of three TCP variants. The results are shown in Fig. 6.6 and 6.7.

Figure 6.6 depicts the goodputs of the three TCP variants with the random loss rate varying from 0 to 15 % and the VBR source with 800 Kb/s averaged sending rate to generate cross traffic. The simulation results demonstrate that both Vegas and RedVegas can always maintain higher goodputs than Reno. As compared with Vegas, when the random loss rate is greater than 3 %, RedVegas always achieves a more than 10 % goodput improvement. In particular, when the random loss rate is 8 %, the goodput improvement of RedVegas reaches 26.7 %.

As the random loss rate is fixed at 5 % and the cross traffic load varies from 0 to 0.9, the simulation results also illustrate that the goodputs of RedVegas are higher than Vegas and Reno as shown in Fig. 6.7. Compared with Vegas, the goodput improvement of RedVegas is kept between 6.7 % and 28.1 %; while it is kept between

6.7 % and 185 % compared with Reno .

6.4.4 Numeric Analysis

Through the above simulation results, we have demonstrated that the goodput of RedVegas is higher than that of Vegas whenever random losses are introduced. However, the effectiveness of packet loss distinguishing scheme in RedVegas has not been verified. To this end, we change the cross traffic loads from 0 to 90 % to study the loss differentiation accuracy of RedVegas with the random loss rate fixed at 1 %, 5 %, 10 %, or 15 %. The execution time is 500 seconds for each sampling statistics. Table 6.1 represents the simulation results.

As the random loss rate is 1 % and the cross traffic load is 40 %, RedVegas detects 472 random losses and 36 congestion losses. For the latter there are 20 wrong judgements. For the 472 detected random losses, there is only one loss caused by congestion. As the random loss rate is 5 % and the cross traffic load is 20 %, RedVegas infers 1498 random losses and 23 congestion losses. Among the 23 detected congestion losses, RedVegas has 10 correct judgements. However, all the 1498 random losses inferred are correct. Based on the results shown in Table 6.1, we mainly have the following three observations: (1) the number of congestion losses is quite small in most of cases, (2) the accuracy of congestion loss detection is not very high, and (3) the accuracy of random loss detection is close to 100 %.

Since RedVegas adopts the congestion avoidance mechanism used in Vegas, it precisely control the amount of extra data between α and β in the bottleneck buffer. Therefore, the congestion losses can be effectively avoided. That is why the number of congestion losses is quite small in most of cases. Recall that RedVegas intends to keep a high accuracy of random loss detection by using the CI information carried by the ACKs of both the preceding and succeeding packets, consequently, some random losses may be possibly misinterpreted as congestion losses. The misinterpreting random loss as congestion loss may degrade throughput, this is same as in Vegas. At most, the misdiagnosis does not bring any goodput improvement to RedVegas. Fi-

Table 6.1: Numeric analysis of packet loss differentiation for RedVegas.

(a)Data Packet Random Loss Rate = 1 %

	Cross Traffic Load									
	0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Actual Random Loss[†]/	862/	742/	668/	554/	471/	402/	328/	224/	132/	52/
Random Loss Detected	862	743	668	554	472	402	328	224	133	53
Actual Congestion Loss[‡]/	0/	4/	8/	20/	16/	23/	37/	40/	61/	66/
Congestion Loss Detected	0	11	16	33	36	39	67	60	116	116

[†]Actual Random Loss does not mean the total number of actual random loss occur,
it means the number of actual random loss occurs in the Random Loss Detected.
[‡]Actual Congestion Loss means the number of actual congestion loss occurs in the Congestion Loss Detected.

(b)Data Packet Random Loss Rate = 5 %

	Cross Traffic Load									
	0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Actual Random Loss/	1805/	1759/	1498/	1567/	1257/	1137/	1027/	615/	436/	118/
Random Loss Detected	1805	1760	1498	1568	1258	1137	1027	615	436	118
Actual Congestion Loss/	0/	3/	10/	11/	15/	18/	23/	34/	48/	62/
Congestion Loss Detected	0	8	23	28	47	51	57	88	152	168

(c)Data Packet Random Loss Rate = 10 %

	Cross Traffic Load									
	0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Actual Random Loss/	719/	555/	694/	602/	695/	616/	441/	453/	321/	89/
Random Loss Detected	719	555	694	602	695	616	441	453	322	89
Actual Congestion Loss/	0/	2/	1/	6/	2/	6/	3/	9/	13/	29/
Congestion Loss Detected	0	2	5	7	6	16	15	31	73	102

(d)Data Packet Random Loss Rate = 15 %

	Cross Traffic Load									
	0 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Actual Random Loss/	167/	236/	235/	185/	219/	169/	168/	136/	123/	51/
Random Loss Detected	167	236	235	185	219	169	168	137	124	51
Actual Congestion Loss/	0/	1/	0/	0/	0/	0/	0/	3/	5/	6/
Congestion Loss Detected	0	1	0	3	4	2	0	6	17	17

nally, the most remarkable observation is that the accuracy of random loss detection is close to 100 %. The results demonstrate the goodput improvement of RedVegas is indeed based on the correct random loss detections.

6.5 Chapter Summary

In this chapter, we propose an improved scheme, RedVegas, for TCP Vegas. By using the innate nature of Vegas and congestion indications marked by routers, RedVegas may detect random packet losses precisely. With the ability of precise random loss detection, RedVegas reacts appropriately to the loss which is either caused by network congestion or transmission error, and consequently enhances the goodput of a connection over heterogeneous networks. Simulation results show the effectiveness of our proposed mechanism. In the future work, we will focus on a new design of fast recovery mechanism to further improve the utilization of bottleneck link when the random loss rate is high.

Chapter 7

Conclusions and Future Work

TCP is the main congestion control method for the Internet, it must remain effective and efficient as the Internet evolves. In this dissertation, several important issues regarding TCP Vegas over the Internet are investigated and improved. We now conclude the dissertation by summarizing our contributions and briefly discussing the future work.

7.1 Summary of Contributions

In this dissertation, we propose four enhanced mechanisms to deal with the problems that may be obstacles of TCP Vegas for achieving a success. The proposed mechanisms can be divided into two categories. One is router-assisted solutions such as RoVegas and RedVegas, the other is end-to-end schemes those are Enhanced Vegas and Quick Vegas. We summarize their contributions as follows:

RoVegas: In Chapter 3, we carefully examine the problems existed in current TCP Vegas scheme and point out the wrongfully estimated *BaseRTT* is the problem source of issues of rerouting, unfairness, and persistent congestion. A router-assisted congestion avoidance mechanism, RoVegas, is hence proposed. Comparing with other previous studies, RoVegas provides a more effective way to solve the problems of rerouting and persistent congestion, to enhance the fairness among the competitive connections, and to improve the throughput when congestion occurs on the backward path. Through the results of both analysis and simulation, the effectiveness of RoVegas is demonstrated.

Enhanced Vegas: Different from RoVegas that is a router-assisted solution. In Chapter 4, we try to improve the performance of TCP Vegas under a constraint of keeping TCP in end-to-end semantics. Enhanced Vegas uses TCP timestamps option to estimate queueing delay on the forward and backward path separately without clock synchronization. Through distinguishing whether congestion occurs in the forward path or not, it significantly advances the connection throughput when the backward path is congested.

Quick Vegas: TCP congestion control may function poorly in high BDP networks because of its slow response with large congestion window size. In Chapter 5, Quick Vegas is proposed to improve this problem. Based on the increment history and estimated amount of extra data, Quick Vegas adopts a more intelligent and aggressive way to adjust its window size. Simulation results show that Quick Vegas reacts faster and better to changing environments and therefore improves the overall performance.

RedVegas: In Chapter 6, we propose a random error detection mechanism for TCP Vegas. By using the innate nature of Vegas and congestion indications marked by routers, RedVegas may detect random packet losses precisely. The simulation results show that the accuracy of random loss detection is close to 100 %. Through the packet loss differentiation, RedVegas reacts appropriately to the loss which is either caused by network congestion or transmission error, and consequently enhances the performance of a connection over heterogeneous networks.

The operation fields in packet header for each proposed mechanism are independent as shown in Fig. 7.1. RedVegas uses two bits in IP header as the CI field and one bit in TCP header as the CI-echo flag. RoVegas adds a AQT option to IP header to generate a probing packet. Enhanced Vegas employs TCP timestamps option to estimate queueing time. Nothing needs to be added in packet header for Quick Vegas. The independent operation fields will facilitate the integration of proposed mechanisms.

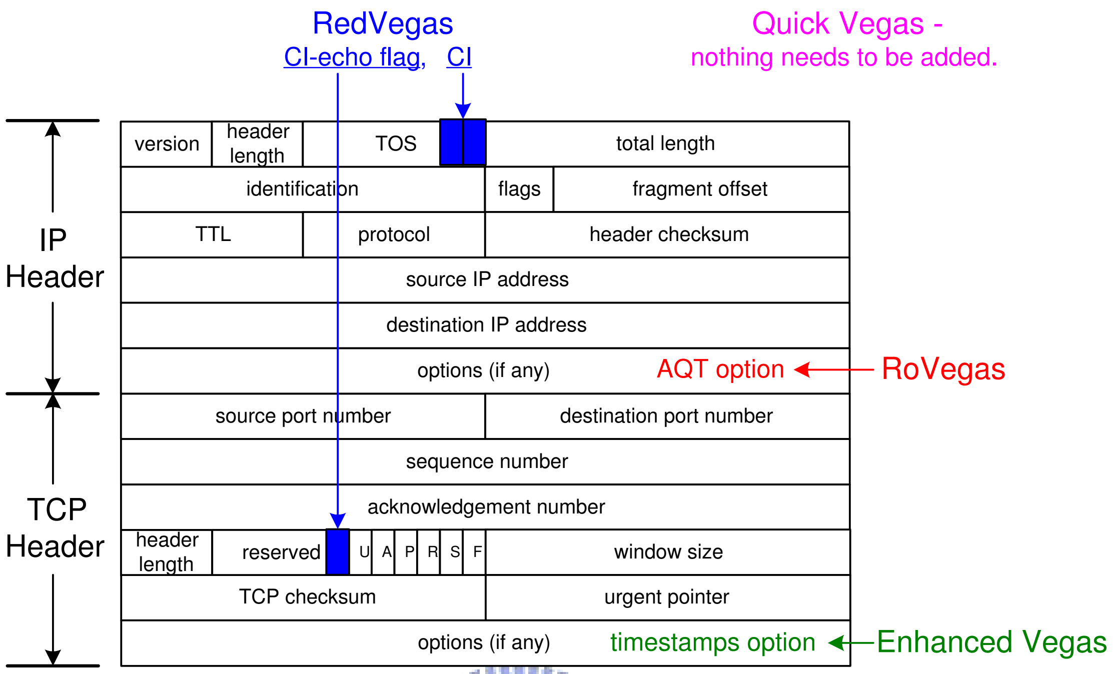


Figure 7.1: Operation fields of proposed mechanisms.

7.2 Future Work

To futher advance this study, future work is needed as we have pointed out in the end of Chapter 3 – Chapter 6. Among these research directions, enabling compatibility between TCP Reno and TCP Vegas may be an attractive topic. One possible reason for TCP Vegas is still unpopular in the Internet is that, when Reno and Vegas coexist in the same bottleneck, Reno generally steals bandwidth from Vegas. Therefore, if TCP Vegas may compete well with TCP Reno in this situation, it will great motivate Internet users to adopt TCP Vegas.

Bibliography

[1] J. Postel, "Transmission Control Protocol," *IETF RFC 793*, September 1981.

[2] V. Jacobson, "Congestion avoidance and control," in *Proc. of ACM SIGCOMM*, pp. 314–329, August 1988.

[3] V. Jacobson, "Modified TCP congestion avoidance algorithm," tech. rep., April 1990.

[4] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple modle and its empiriacl validation," in *Proc. of ACM SIGCOMM*, pp. 303–314, August 1998.

[5] 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, 1996.

[6] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

[7] R. Morris, "TCP behavior with many flows," in *Proc. of IEEE ICNP*, pp. 205–211, October 1997.

[8] C. Samios and M. K. Vernon, "Modeling the throughput of TCP Vegas," *ACM SIGMETRICS*, vol. 31, no. 1, pp. 71–81, 2003.

[9] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP Reno and Vegas," in *Proc. of IEEE INFOCOM*, pp. 1556–1563, March 1999.

[10] G. Hasegawa, M. Murata, and H. Miyahara, "Fairness and stability of congestion control mechanism of TCP," *Telecommunication Systems Journal*, pp. 167–184, November 2000.

[11] W. Feng and P. Tinnakornsrisuphap, "The failure of TCP in high-performance computational grids," in *Proc. of SC 2000: High-performance Networking and Computing Conf.*, November 2000.

[12] A. Veres and M. Boda, "The chaotic nature of TCP congestion control," in *Proc. of IEEE INFOCOM*, pp. 1715–1723, March 2000.

[13] J. C. Hoe, "Start-up dynamics of TCP's congestion control and avoidance schemes." Master's thesis, MIT, Jun. 1995.

[14] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options," *IETF RFC 2018*, October 1996.

[15] M. Mathis and J. Mahdavi, "Forward acknowlegement: Refining TCP congestion control," in *Proc. of ACM SIGCOMM*, pp. 181–191, August 1996.

[16] D. Lin and H. T. Kung, "TCP fast recovery strategies: Analysis and improvements," in *Proc. of IEEE INFOCOM*, pp. 263–271, March 1998.

[17] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit," *IETF RFC 3042*, Jan. 2001.

[18] Z. Wang and J. Crowcroft, "Eliminating periodic packet losses in 4.3-Tahoe BSD TCP congestion control algorithm," *ACM Computer Communication Review*, vol. 22, no. 2, pp. 9–16, 1992.

[19] R. Jain, “A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks,” *ACM Computer Communication Review*, vol. 19, no. 5, 1989.

[20] Z. Wang and J. Crowcroft, “A new congestion control scheme: Slow start and search (Tri-S),” *ACM Computer Communication Review*, vol. 21, no. 1, pp. 32–43, 1991.

[21] S. Keshav, “A control-theoretic approach to flow control,” *ACM Computer Communication Review*, vol. 25, no. 1, pp. 189–201, 1995.

[22] L. S. Brakmo, S. W. O’Malley, and L. L. Peterson, “TCP Vegas: New techniques for congestion detection and avoidance,” in *Proc. of ACM SIGCOMM*, pp. 24–35, August 1994.

[23] L. S. Brakmo and L. L. Peterson, “TCP Vegas: End to end congestion avoidance on a global Internet,” *IEEE J. Select. Areas Commun.*, vol. 13, pp. 1465–1480, 1995.

[24] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, “Evaluation of TCP Vegas: Emulation and experiment,” in *Proc. of ACM SIGCOMM*, pp. 185–195, August 1995.

[25] U. Hengartner, J. Bolliger, and T. Gross, “TCP Vegas revisited,” in *Proc. of IEEE INFORCOM*, pp. 1546–1555, March 2000.

[26] C. Parsa and J. J. Garcia-Luna-Aceves, “Improving TCP congestion control over Internet with heterogeneous transmission media,” in *Proc. of IEEE ICNP*, pp. 213–221, Nov. 1999.

[27] C. Boutremans and J. L. Boudec, “A note on the fairness of TCP Vegas,” in *Proc. of International Zurich Seminar on Broadband Communications*, pp. 163–170, February 2000.

[28] D. D. Luong and J.Biro, "On the proportional fairness of TCP Vegas," in *Proc. of IEEE GLOBECOM*, pp. 1718–1722, November 2001.

[29] O. Elloumi, H. Afifi, and M. Hamdi, "Improving congestion avoidance algorithms for asymmetric networks," in *Proc. of IEEE Int. Conf. Communications*, pp. 1417–1421, June 1997.

[30] C. P. Fu and S. C. Liew, "A remedy for performance degradation of TCP Vegas in asymmetric networks," *IEEE Commun. Lett.*, vol. 7, no. 1, pp. 42–44, 2003.

[31] C. P. Fu, L. C. Chung, and S. C. Liew, "Performance degradation of TCP Vegas in asymmetric networks and its remedies," in *Proc. of IEEE Int. Conf. Communications*, pp. 3229–3236, June 2001.

[32] S. Vanichpun and W. Feng, "On the transient behavior of TCP Vegas," in *Proc. of IEEE ICCCN*, pp. 504–508, Oct. 2002.

[33] T. V. Lakshman, U. Madhow, and B. Suter, "Window-based error recovery and flow control with a slow acknowledgement channel: A study of TCP/IP performance," in *Proc. of IEEE INFOCOM*, pp. 1199–1209, April 1997.

[34] H. Balakrishnan, V. N. Padmanabhan, S. Sechan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Trans. Networking*, vol. 5, no. 6, pp. 756–769, 1997.

[35] Y. Lai, "Improving the performance of TCP Vegas in heterogeneous environment," in *Proc. of Int. Conf. Parallel and Distributed Systems*, pp. 581–587, June 2001.

[36] W. Feng and S. Vanichpun, "Enabling compatibility between TCP Reno and TCP Vegas," in *Proc. of IEEE Symposium on Applications and the Internet*, pp. 301–308, January 2003.

[37] R. Braden, "Requirements for Internet Hosts - Communication Layers," *IETF RFC 1122*, October 1989.

[38] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *IETF RFC 2581*, April 1999.

[39] S. Floyd, "Congestion Control Principles," *IETF RFC 2914*, September 2000.

[40] M. Allman, S. Floyd, C. Partridge, "Increasing TCP's Initial Window," *IETF RFC 3390*, October 2002.

[41] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," *IETF RFC 1323*, May 1992.

[42] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," *IETF RFC 2582*, April 1999.

[43] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," *IETF RFC 3168*, September 2001.

[44] J. Postel, "Internet Protocol," *IETF RFC 791*, September 1981.

[45] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols.* Addison-Wesley, 1994.

[46] UCB/LBNL/VINT Network Simulator (ns), "." http://www.isi.edu/nsnam/ns/.

[47] H. Balakrishnan and V. N. Padmanabhan, "How network asymmetry affects TCP," *IEEE Commun. Magazine*, vol. 39, pp. 60–67, April 2001.

[48] S. B. Moon, P. Skelly, and D. Towsley, "Estimation and removal of clock skew from network delay measurement," in *Proc. of IEEE INFOCOM*, pp. 227–234, March 1999.

[49] L. Zhang, Z. Liu, and C. H. Xia, "Clock synchronization algorithms for network measurements," in *Proc. of IEEE INFOCOM*, pp. 160–169, June 2002.

[50] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. of ACM SIGCOMM*, August 2002.

[51] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, “On the self-similar Nature of Ethernet Traffic,” *IEEE/ACM Trans. Networking*, vol. 2, no. 1, p. Feb., 1994.

[52] S. Floyd, “HighSpeed TCP for large congestion windows.” Internet draft draft-floyd-tcp-highspeed-02.txt, Feb. 2003.

[53] T. Kelly, “Sclable TCP: Improving performance in highspeed wide area networks,” *ACM Computer Communication Review*, vol. 33, no. 2, pp. 83–91, 2003.

[54] A. Jain and S. Floyd, “QuickStart for TCP and IP.” Internet draft draft-amit-quick-start-02.txt, Oct. 2002.

[55] R. Jain, *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation and modeling.* New York: Wiley, 1991.

[56] T. V. Lakshman and U. Madhow, “The performance of TCP/IP for networks with high bandwidth-delay products and random loss,” *IEEE/ACM Trans. Networking*, vol. 5, no. 3, pp. 336–350, 1997.

[57] C. P. Fu and S. C. Liew, “TCP Veno: TCP enhancement for transmission over wireless access networks,” *IEEE J. Select. Areas Commun.*, vol. 21, no. 2, pp. 216–228, 2003.

[58] C. L. Lee, C. F. Liu, and Y. C. Chen, “On the use of loss history for performance improvement of TCP over wireless networks,” *IEICE Trans. Commun.*, vol. E85-B, no. 11, pp. 2457–2467, 2002.

[59] S. Namda, R. Ejzak, and B. T. Doshi, “A retransmission scheme for circuit-mode data on wireless links,” *IEEE J. Select. Areas Commun.*, vol. 12, no. 8, pp. 1338–1352, 1994.

[60] E. Ayanoglu, S. Paul, T. F. LaPorta, K. K. Sabnani, and R. D. Gitlin, "AIR-MAIL: A link-layer protocol for wireless networks," *Wireless Networks*, vol. 1, pp. 47–60, 1995.

[61] D. Bansal, A. Chandra, and R. Shorey, "An extension of the TCP flow control algorithm for wireless networks," in *Proc. of IEEE ICPWC*, pp. 207–210, Feb. 1999.

[62] A. V. Bakre and B. R. Badrinath, "Implementation and performance evaluation of indirect TCP," *IEEE Trans. Computers*, vol. 46, no. 3, pp. 260–278, 1997.

[63] R. Yavatkar and N. Bhagawat, "Improving end-to-end performance of TCP over mobile internetworks," in *Proc. of IEEE Workshop on Mobile Computing Systems and Applications*, pp. 146–152, Dec. 1995.

[64] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *ACM Computer Communication Review*, vol. 27, no. 5, pp. 19–43, 1997.

[65] H. Balakrishnan, S. Sechan, E. Amir, and R. H. Katz, "Improving TCP/IP performance over wireless networks," in *Proc. of ACM MOBICOM*, pp. 2–11, Nov. 1995.

[66] N. Samaraweera and G. Fairhurst, "Reinforcement of TCP error recovery for wireless communcation," *ACM Computer Communication Review*, vol. 28, no. 2, pp. 30–38, 1998.

[67] S. Xu, T. Saadawi, and M. Lee, "On TCP over wireless Multi-hop networks," in *Proc. of IEEE MILCOM*, pp. 282–288, Oct. 2001.

[68] D. Duchamp and N. F. Reynolds, "Measured performance of a wireless LAN," in *Proc. of IEEE International Conference on Local Computer Network*, pp. 494–499, Sept. 1992.

[69] D. Eckhardt and P. Steenkiste, "Improving wireless LAN performance via adaptive local error control," in *Proc. of IEEE ICNP*, pp. 327–338, Oct. 1998.

[70] S. Floyd and V. Paxson, "Difficulties in simulating the Internet," *IEEE/ACM Trans. Networking*, vol. 9, no. 4, pp. 392–403, 2001.

Curriculum Vitae

Yi-Cheng Chan was born in Taichung, Taiwan, R.O.C., in 1967. He received his B.S. degree in Computer Science from Tamkang University in 1990, and the M.S. degree in Computer Science and Engineering from Yuan-Ze University in 1994. He is currently pursuing the Ph.D. degree in Computer Science and Information Engineering, National Chiao Tung University. His research interests include design and analysis of network protocols, switch architecture design, and active queue management.

Publication List

- Journal Paper

  1. Yi-Cheng Chan, Chia-Tai Chan, and Yaw-Chung Chen, "An Enhanced Congestion Avoidance Mechanism for TCP Vegas," *IEEE Communications Letters*, vol. 7, issue 7, pp. 343–345, July 2003.
  2. Yi-Cheng Chan, Chia-Tai Chan, and Yaw-Chung Chen, "Design and performance evaluation of an impoved TCP congestion avoidance scheme," *IEE Proceedings – Communications*, vol. 151, no. 1, pp. 107–111, February 2004.
  3. Yi-Cheng Chan, Chia-Tai Chan, and Yaw-Chung Chen, "RedVegas: Performance Improvement of TCP Vegas over Heterogeneous Networks," accepted by *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences.*
  4. Yi-Cheng Chan, Chia-Tai Chan, and Yaw-Chung Chen, "RoVegas: A router-based congestion avoidance mechanism for TCP Vegas," submitted to *Computer Communications* (in revision).
  5. Yi-Cheng Chan, Chia-Tai Chan, and Yaw-Chung Chen, "Quick Vegas: Improving Performance of TCP Vegas for High Bandwidth-Delay Product Networks," submitted to *IEICE Transations on Information and Systems.*

- Conference Paper

  1. Yi-Cheng Chan, Chia-Tai Chan, and Yaw-Chung Chen, "Performance Improvement of TCP Vegas over Heterogeneous Networks," in *Proc. of the 24th IEEE International Conference on Distributed Computing Systems Workshops, ICDCS 2004 Workshop*, Tokyo, Japan, pp. 36–41, March 2004.

2. Yi-Cheng Chan, Chia-Tai Chan, Yaw-Chung Chen, and Cheng-Yuan Ho, "Performance Improvement of Congestion Avoidance Mechanism for TCP Vegas," accepted by *the Tenth IEEE International Conference on Parallel and Distributed Systems, ICPADS 2004.*