國 立 交 通 大 學

資 訊 工 程 學 系

博 士 論 文

一個端對端的壅塞控制機制的設計與效能分析

Design and Performance Evaluation of an End-to-end Congestion Control Algorithm

研 究 生：何承遠

指導教授：陳耀宗 博士

中 華 民 國 九 十 七 年 六 月

一個端對端的壅塞控制機制的設計與效能分析

Design and Performance Evaluation of an End-to-end Congestion Control Algorithm

研 究 生：何承遠　　Student: Cheng-Yuan Ho

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
College of Computer Science
National Chiao Tung University
In Partial Fulfillment of the requirements
For the Degree of
Doctor of Philosophy
in
Computer Science

June 2008
Hsinchu, Taiwan, Republic of China

中 華 民 國 九十七 年 六 月

Design and Performance Evaluation of an End-to-end Congestion Control Algorithm

Student: Cheng-Yuan Ho

Advisor: Dr. Yaw-Chung Chen

A Dissertation Submitted to

the Department of Computer Science

College of Computer Science

National Chiao Tung University

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

Hsinchu, Taiwan, Republic of China

June 2008

一個端對端的壅塞控制機制的設計與效能分析

學生：何承遠　　　　　　　　　　指導教授：陳耀宗博士

國立交通大學資訊工程學系

摘　　要

廣泛且簡易的網路應用造就網際網路的成功，而 TCP 是目前網際網路上最被廣為使用的傳輸層協定。這是因為 TCP 利用其可靠性的傳輸以及端點對端點的擁塞控制之特點，為傳統的網路環境中，提供了一個可接受的服務品質。無論在學術界還是工業界，TCP/IP 的研究始終是一個熱門的課題。

隨著蓬勃發展的網路硬體技術與網路應用，可預見的是，使用者的網路頻寬會越來越大與無線環境也會越來越普遍。這將會造成網際網路的訊務流量快速成長以及產生多樣化的傳輸方式，再加上使用者對於網路效能的要求也會不斷增加。面對此一改變，如何有效的使用網路資源、適應網際網路寬頻化與異質化網路環境(有線、無線網路的混接)以及滿足使用者的要求是一個成功的壅塞控制機制所要面臨的種種問題。

另外，假若 TCP 的改良無法齊頭並進，那麼很快的，TCP 本身的壅塞控制機制將成為效能上的瓶頸。至今 TCP 被創造出許多不同的版本，用來改進網路的使用效能。這些版本可以分為兩大類：loss-based

TCP 和 delay-based TCP。雖然很多研究報告已經指出 delay-based TCP 的機制可以做出較精確的控制、擁有更好的特性，使得 delay-based TCP 在很多方面都要優於 loss-based TCP，例如：整體網路的使用率、穩定性、公平性以及傳輸速率…等。但是 loss-based TCP 仍為現今主流，而且目前關於 TCP 的研究大多集中在 loss-based TCP 上，原因在於這類 TCP 的方法實做起來較簡單且競爭力較另一類 TCP 來得要強，故 TCP SACK 方案為現今網際網路上最被廣為使用的 TCP 版本。很可惜的是，實際上這兩類 TCP 都仍然有一些屬於自己的缺點存在於它的壅塞控制機制中，因此衍生出一個問題：要如何設計 TCP 以減少本身的缺點。

在這份論文中，我們提出一個新的端對端的壅塞控制機制用於適應新的網路環境、充分利用網路資源以滿足使用者的要求以及克服 TCP 本身缺點所造成的負面影響，並命名為 Medley TCP。與傳統的 TCP 相比較，最大的不同在於我們重新設計 TCP 整體的壅塞控制機制與架構。具體來說，Medley TCP 是以 loss-based TCP 的演算法做為基礎模組，搭配 delay-based TCP 的估算機制與控制模式來調整最終的傳輸速率，嘗試結合這兩類 TCP 的優點與精神。為了讓連線在高頻寬-延遲乘積網路中能有較好的表現，Medley TCP 利用連線估測堆積在佇列中的封包數量與其推算之值為依據，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。另外，Medley TCP 本身的特性可以準確判斷出封包遺失是屬於網路的壅塞還是傳輸失誤所造成，透過封包遺失原因的分類，Medley TCP 可以適切的對不同原因的封包遺失做出不同的反應，因此改善了在異質化網路環境中TCP 傳輸的效能。

通過廣泛網路模擬軟體的實驗測試和網際網路的實地測試，Medley TCP 被證實能在高頻寬-延遲乘積網路與異質化的網路環境下獲得大幅度的效能提高，而且 Medley TCP 也保留了 delay-based TCP 在整體網路的使用率、穩定性、公平性以及傳輸速率…等方面的優異特性。更重要的是 Medley TCP 可以與現存的 TCP SACK 和諧共存而不降低其效能，這是因為 Medley TCP 可以更有效的利用網際網路以獲得頻寬，而不是搶奪其他連線所應佔用的資源。最後，Medley TCP 不用修改連線的接受端或者中間節點，僅需要在資料的發送端做修改即可。因此，Medley TCP 可以容易地安裝於實際的網際網路中。

Design and Performance Evaluation of an End-to-end Congestion Control Algorithm

Student: Cheng-Yuan Ho

Advisor: Dr. Yaw-Chung Chen

Institute of Computer Science

National Chiao Tung University

ABSTRACT

The success of the Internet can be attributed to the large number of useful applications which are easily executed by a user for running on the Internet. Transmission Control Protocol (TCP) is a widely deployed end-to-end transport protocol in the current Internet. This is because TCP provides an acceptable service with a reliable data transport as well as controls the connection's bandwidth usage to avoid network congestion for two end hosts in the Internet. Nowadays, the research of TCP/IP is still a hot topic in both the academia and the industry.

With the fast growth of Internet hardware, technologies, and applications, the network bandwidth of a user is getting higher and wireless links are more and more popular everywhere. This will make Internet traffic increase quickly and the ways of data transport vary. Also, users' demands for network performance are getting stricter. Facing these challenges, how to efficiently utilize network resources, how to work well in both high bandwidth network and heterogeneous network (mixed with wired and wireless networks), and how to satisfy users' requests are essential issues to a successful congestion control mechanism.

In addition, if the modification of TCP's congestion control does not catch up the Internet change, the performance bottleneck will soon be TCP itself. Up to now, in order to improve network utilization, TCP has several implementation versions which can be classified into two categories, loss-based TCPs and delay-based

TCPs. Although it has been demonstrated that the delay-based TCP outperforms the loss-based TCP in the aspects of overall network utilization, stability, fairness, throughput, etc., in the real Internet, loss-based TCPs are still the mainstream and remained as the dominant algorithm used in practice. Furthermore, the implementation of a loss-based TCP is easier than that of a delay-based TCP, and a loss-based TCP could get more resources than a delay-based TCP could when they coexist in the same network, so many studies focus on loss-based TCP algorithms, and one version of loss-based TCP mechanisms, TCP SACK, has been widely deployed on the Internet. However, in fact, loss-based and delay-based TCPs have their own advantages and shortcomings. Hence, there is an issue "how do we design the algorithm and architecture of TCP?"

In this dissertation, we propose a novel end-to-end congestion control mechanism called Medley TCP that is able to efficiently utilize Internet resources, adapt itself to a new network circumstance, satisfy users' requests, and accommodate shortcomings of the conventional TCP. Medley TCP differs from the traditional TCP extremely in that we redesign the algorithm and architecture of the whole congestion control mechanism of TCP. Specifically, Medley TCP tries to combine advantages and characteristics of both loss-based and delay-based TCPs, and therefore incorporates a scalable delay-based component into a loss-based TCP algorithm. This scalable delay-based component has a rapid window increase rule when the network is sensed to be under-utilized and gracefully reduces the sending rate once the bottleneck queue is built. Therefore, Medley TCP connections will react faster and better to high BDP networks and improve the overall performance. Moreover, we utilize the innate nature of Medley TCP to detect congestion losses or random packet losses precisely. Through the packet loss differentiation, Medley TCP reacts appropriately to the losses and consequently the throughput of connection over heterogeneous networks can be significantly improved.

Extensive experiments of a network simulator and real world Internet traffic measurements have been conducted and show that Medley TCP not only can improve the performance significantly over high BDP networks and heterogeneous networks but also can keep the good characteristics of delay-based TCPs in the aspects of overall network utilization, stability, fairness, throughput, and so on. Importantly, Medley TCP does not cause any detrimental effects to TCP SACK, and vice versa when they share same resources in the networks: that is, Medley TCP connections achieve higher throughput by using bandwidth that will not be used by SACK connections anyway, not "stealing" bandwidth from SACK's connections. Finally, Medley TCP only involves modification at the sender without requiring changes of the receiver protocol stack or intermediate router nodes. As a result, it can be easily deployed in the current Internet.

Keywords: Transmission Control Protocol, transport protocol, loss-based TCPs, delay-based TCPs, fairness, TCP friendly, high bandwidth-delay product networks, heterogeneous networks, wireless networks, Internet.

Acknowledgements

Wow, time flies so fast. Five years ago, I started my graduate live at National Chiao Tung University. After one year, I enrolled into Ph. D. program directly. Now, I obtain my Ph. D. degree fortunately. I deeply appreciated the people who influenced and helped me in these years. There is no doubt that, without their help, it would be impossible to complete this dissertation. I hope that I can remember everyone who helped me through this difficult yet rewarding process.

First and foremost, I would like to thank my advisor, Prof. Yaw-Chung Chen, for taking me as his student and leading me to involve into this exciting research area. His creative idea and quick thinking impress me deeply. He is such a nice advisor who never gives me too much pressure but will help me at the right time. He also encourages me to feel, observe, and learn from the real life, not to being a computer nerd. No matter whether in life or in research, he is my best advisor.

Second I would thank my senior, Dr. Yi-Cheng Chan. Before enrolling into Ph. D. program, I had no idea of the research and was a little bit scared to ask my advisor about this. However, I took the first step by co-working with Dr. Yi-Cheng Chan. He not only gives me invaluable suggestions and timely support on my research, but also makes me understand a lot of knowledge through our happy discussions. And all the way, he stays beside me as a best friend as well as a good teacher.

My other thanks go to Kun Tan who is a researcher in the wireless and networking group, Microsoft Research, Asia, Beijing, China, and all members (including students) of this group. This is because I was an intern there from March to September 2006. During this period, I benefited greatly from them for their invaluable simulations, experiments, and discussions. I hope that I can live up to such high standards.

Finally, I owe a special debt of gratitude to my parents and younger brother. Playing video game with my younger brother makes me relaxed and recharge my exhausting energy. Words cannot express my thankfulness to my father and mother because they make me what I am today: without them, without me. I want to dedicate my Ph. D. thesis to my dear parents.

Cheng-Yuan Ho

National Chiao Tung University in Hsinchu, Taiwan.

June 2008

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Transmission Control Protocol (TCP) is a reliable data transfer protocol used widely over the Internet for numerous applications, from long-lived TCP flows such as File Transfer Protocol (FTP) to short-lived TCP connections like Hypertext Transmission Protocol (HTTP). TCP is also a connection-oriented, end-to-end, and error-free in-order protocol. A TCP connection is a virtual circuit between two end hosts that is conceptually similar to a telephone connection. The difference between TCP and telephone connections is that TCP provides a reliable data transport as well as controls the connection's bandwidth usage to avoid network congestion. The behavior of TCP is tightly coupled with the overall Internet performance because TCP is treated as the underlying communication protocol by most of Internet applications nowadays.

The essential strategy of TCP is as follows. A source host divides the data stream into segments (also called packets). Each segment is labelled with an explicit sequence number to guarantee the ordering and reliability. When a receiver gets a segment successfully, it sends a cumulative acknowledgment (ACK) in return, notifying the sender that all of the data preceding that segment's sequence number has been received. If an out-of-order (or out-of-sequence) segment is received, the receiver sends an ACK indicating the sequence number of the expected segment. This is called duplicate ACK. If the outstanding data is not acknowledged for a pre-defined period of time, the sender will timeout and retransmit the unacknowledged segments.

The 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 can be thought as a reactive congestion control scheme or classified as a loss-based TCP. This is because it uses packet loss as an indicator for congestion. In order to probe the available bandwidth along the end-to-end path, TCP Reno increases its congestion window size until a packet loss is detected, at that point the congestion window is halved and a linear increase algorithm will take over until further packet loss is encountered.

TCP congestion control is an active research area. Over the years, considerable research regarding the techniques of TCP has been done [4, 5, 6, 7, 8, 9, 10, 11]. It is known that TCP Reno may periodically generate packet loss by itself and can not efficiently recover multiple packet losses of a window worth of data. To alleviate the performance degradation problem of packet loss, many researchers attempted to refine the fast recovery algorithm embedded in TCP Reno. New proposals include TCP NewReno [12], SACK [13], FACK [14], Net Reno [15], and LT [16], and so on. These algorithms bring performance improvement for a connection after a packet loss is detected. Among these mechanisms, SACK is currently the dominating TCP version deployed in the Internet. Recently, Compound TCP [17] is proposed by Microsoft in order to increase SACK's sending rate during AIMD phase.

However, the AIMD strategy of a loss-based TCP leads to periodic oscillations in the aspects of congestion window size, round-trip delay, and queue occupancy[1] of the bottleneck node. Recent works have shown that the oscillation may induce chaotic behavior into the network thus adversely affects overall network performance [18, 19]. To combat the inherent oscillation problem of loss-based TCP algorithms, many congestion avoidance mechanisms are proposed. These works include DUAL [20], CARD [21], Tri-S [22], Packet-Pair [23], and delay-based TCPs. The so-called delay-based TCPs make congestion decisions and adjust the transmission rate based on round-trip time (RTT) variations, e.g., TCP Vegas [24, 25, 26, 27], TCP Santa Cruz [28], and FAST TCP [29].

Several studies have reported that delay-based TCPs provide better performance than loss-based TCPs with respect to overall network utilization [25, 26], packet loss, stability [10, 11], fairness [10, 11], and throughput [18, 25, 26]. However, considering overall performance over the heterogeneous networks, delay-based TCPs do not compare well with loss-based TCPs and may even have certain disadvantages [27]. This is because loss-based TCPs could "dispute" more resources than delay-based TCPs could. Therefore, in the real Internet, loss-based TCPs are still the mainstream and remain the dominant mechanisms used in practice.

1.2 Problem Statements and Motivation

With the fast growth of Internet traffic, a successful congestion control scheme needs to efficiently utilize network resources. TCP provides reliable data transmission by adjusting the sending rate according to the available bandwidth of the network. However, it has become clear that the TCP mechanisms, while necessary and powerful, are not sufficient to provide satisfactory services for all circumstances. In the following paragraphs, we will discuss the main challenges and issues currently faced by TCP.

[1]In this dissertation, the terms of 'buffer size' and 'queue occupancy' refer to 'fixed hardware space' and 'dynamic fullness of the buffer', respectively.

(1) Long-lived TCP Flows

Long-lived TCP flows may spend most life time in the congestion avoidance phase. However, due to the limitation in TCP's conservative congestion control algorithm, it may not efficiently utilize the network capacity when applications require fast data transfer over high speed and long latency networks especially [30]. This is because, during congestion avoidance, the congestion window size (CWND) grows so conservatively so that it may not fully use surplus resource promptly.

In addition, by setting the initial slow start threshold (SSTHRESH) to an arbitrary value, TCP performance may suffer from two potential problems: (a) the exponential increase of congestion window may generate packets too fast, causing multiple losses at the bottleneck nodes and coarse timeouts if the threshold is set too high relative to the per-flow available bandwidth (PAB). Hence, it leads to significant reduction of the throughput; (b) the connection will exit slow start and switches to congestion avoidance prematurely, resulting in poor startup utilization especially with a high bandwidth-delay product (BDP) network if the initial threshold is set too low (compared with PAB). For example, if TCP Reno/NewReno implementation sets initial SSTHRESH to 32 Kbytes, a TCP connection takes about 100 seconds to reach the ideal window over a path with a bottleneck bandwidth of 100 Mbps and RTT of 100 ms. The utilization in the first 10 seconds is a meager 5.97%. With the rapid advance of the Internet and ever-growing BDP, a more efficient mechanism is required to achieve better link-utilization.

(2) Short-lived TCP Connections

Recent studies in [31] reveal that a majority of the TCP connections such as Web traffic are short-lived, while a small number of long-lived connections like FTP traffic carry most Internet traffic. A short-lived connection usually terminates even before it reaches a steady state, that is, before the congestion window grows to make good bandwidth utilization of the end-to-end connection. Moreover, the probability that a packet loss will trigger the fast retransmit and fast recovery algorithm is very low.

This is because the window size will be small during the entire duration of the data transfer, and thus the sender will not be able to send enough segments to cause three duplicate ACKs. As a result, most losses trigger a retransmission timeout (RTO) which will force the TCP flow to reenter slow start phase. Accordingly, the startup stage may significantly affect the performance of short-lived TCP connections.

(3) The Queue Occupancy Made by TCP in a Router

A sender has no prior knowledge regarding the available bandwidth in the networks when a new connection starts, so this leads to the abrupt transition of congestion window with exponential growth and transmission of highly bursty traffic from the source during slow start. For loss-based TCPs, they increase CWND exponentially until reaching SSTHRESH or causing buffer overflow on the bottleneck link. Although delay-based TCPs may not cause buffer overflow, they stop increasing CWND with exponential growth when reaching their own SSTHRESH value[2]. In other words, TCP is known to send bursts of packets during its slow start phase due to the fast window increase and the ACK-clock based transmission. Furthermore, a buffer overflow may cause multiple packets from the window being dropped in one RTT, thereby causing the self-clocking mechanism of the TCP source to fail.

When staying at the congestion avoidance phase, loss-based TCPs keep increasing their window size until packet loss occurs. Therefore, the curve of queue occupancy made by loss-based TCPs is like a sawtooth. This causes the delay between a packet and its next one unexpected. Comparably, delay-based TCPs try to stabilize their sending rate and make a fixed queue occupancy in the router. Therefore, delay-based TCPs are more suitable being used to transmit real-time applications' data than loss-based TCPs are [32, 33].

[2]The definition of SSTHRESH for delay-based TCPs is different from that for loss-based TCPs. Moreover, each delay-based TCP algorithm's SSTHRESH may be not the same.

(4) Fairness and Friendliness Issues of Different TCP Variants

When a delay-based TCP user competes with other loss-based TCP users, it does not receive a fair share of bandwidth due to the conservative congestion avoidance mechanism used by the delay-based TCP. The reason is that the loss-based TCP continues to increase the window size until a packet is lost. This would occur mainly due to buffer overflow in an intermediate node (assuming the queue management algorithm is Drop Tail). This bandwidth estimation mechanism results in a periodic oscillation of window size and buffer-filling behavior of the loss-based TCP. Thus, while the delay-based TCP tries to maintain a smaller queue occupancy, the loss-based TCP keeps inserting more packets into the queue, and stealing more bandwidth [10, 34, 35, 36, 37].

(5) Unfairness of TCP Sources with Different RTTs

It is well known that there is an unfairness problem when loss-based TCP connections feature different RTTs (i.e., connections traverse networks with various latencies). The connections with smaller RTT increase their rate faster, grab more available bandwidth, and get more network resource [38]. The bias goes against longer RTT connections by a factor of RTT. Different from loss-based TCP mechanisms, delay-based TCP algorithms are not biased against the connections with longer RTT [10, 11].

(6) Random Loss

Owing to the great advancement in wireless networking technology and emerging applications, providing ubiquitous mobile Internet access becomes increasingly demanded. 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 indication 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 leads to the reduction of throughput unnecessarily. Therefore, TCP's performance drops rapidly in the presence of frequent random losses [39, 40]. More throughput deterioration problems of TCP over wireless networks have been addressed in [41, 42, 43, 44, 45, 46, 47, 48, 49, 50].

Summary

According to the above discussions, loss-based TCPs and delay-based TCPs may face same challenges and various problems. For example, how to improve the performance degradation of TCP in wireless network is a common challenge for both loss-based and delay-based TCP mechanisms. How to coexist with a loss-based TCP in the network is a tough task of delay-based TCPs. All these problems may prevent TCP from achieving a further success. In fact, loss-based and delay-based TCPs have their own advantages and shortcomings. "*Does there exist an algorithm featuring advantages and characteristics of both loss-based and delay-based TCPs?*" Our motivation of this dissertation is trying to answer this question.

Many proposed mechanisms try to solve one or some problems as mentioned before, by modifying the algorithm of one phase or changing parameter settings. For instance, [51] selects SSTHRESH dynamically to suit various kinds of BDP networks; however, it needs to estimate the available bandwidth of the network at the steady state. Setting SSTHRESH to the maximum value directly to avoid buffer overflow and limit the sending rate is suggested in [52, 53], but this may reduce the throughput of a sender. In order to reduce bursty traffic during slow start, [54] uses 200 msec timer interrupt to control data transfer. However, this way only fits some network topology. Furthermore, using timer interrupt increases the overhead of the operating system.

HighSpeed TCP [55] involves a subtle change in the congestion avoidance response function to allow connections to capture available bandwidth more readily. Similar to TCP Reno, HighSpeed TCP also causes periodic oscillations in the congestion window size, round-trip delay, and queue length of the bottleneck node [18, 19]. XCP [56] 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.

Accordingly, the way of modifying the algorithm of one phase or changing parameter settings could enhance TCP performance in some circumstances; however, this kind of method has its limitations and may not be able to solve other issues, and even may make problems critical in other environments. Hence, in order to solve aforementioned issues and improve TCP performance in all circumstances, we redesign the whole algorithm and architecture of transmit control protocol in this dissertation.

1.3 Contributions

This dissertation is divided into two areas:

(1) Investigate the core idea of some familiar TCP variants and discuss TCP performance degradation issues.

(2) Medley TCP – a new end-to-end congestion control algorithm is proposed to accommodate the obstacles of TCP for achieving a better performance. Medley TCP has also been implemented in Linux and runs on Internet successfully.

The first area takes up only a small part of this dissertation. The second part is the main contribution of this dissertation and is partially inspired by the study of the first area. We now briefly describe our proposed mechanism along with its performance and contributions as follows.

The key idea of Medley TCP is that it incorporates a scalable delay-based component into a loss-based TCP algorithm and determines congestion window based on congestion information – queueing delay and packet loss. In other words, it combines advantages and characteristics of both loss-based and delay-based TCPs. When Δ is no larger than Γ[3], Medley TCP will increase its congestion window size much quickly. Otherwise, Medley TCP will decrease its window size by one or increase the sending rate very slowly based on its estimate decision while Δ is larger than Γ. Therefore, Medley TCP is suitable for networks with high speed recourses and heterogeneous properties.

Also, in order to make it work well in a wireless network, we utilize the innate nature of Medley TCP to detect random packet losses precisely and modify the fast retransmit and fast recover phase of Medley TCP. Through the packet loss differentiation, Medley TCP reacts appropriately to the losses. Medley TCP only requires modification to the TCP sender and is therefore inter-operable with existing TCP: improvement can be achieved by having only one end of the connection employ Medley TCP. The details of Medley TCP is addressed in the dissertation.

The simulation results and Internet measurements show that Medley TCP not only has good utilization of bottleneck bandwidth, higher performance in a wireless network, better fairness, and friendliness, but also enhances the service for real-time applications such as Internet Protocol Television (IPTV), Voice over Internet Protocol (VoIP), and so on. The main reason is that Medley TCP's throughput is very stable that supplies a fixed rate to real-time applications and the variation range of queue occupancy made by Medley TCP is very small. More simulation results, explanations, Linux implementation, and Internet measurement are presented in following Chapters.

[3]The in-depth discussions of Δ and Γ are addressed in Chapter 3.

1.4 Dissertation Outline

The rest of this dissertation is organized as follows. In Chapter 2, we describe the core idea of two TCP categories, a loss-based TCP and a delay-based TCP. The design principles of some notable TCP implementations are also reviewed in this Chapter. In Chapter 3, we propose Medley TCP as a potential scheme to solve problems discussed in Section 1.2. Chapter 4 shows the analysis, performance model, examples, and fairness discussion for capturing Medley TCP's behavior. Chapter 5 contains numerous simulation results that validate the efficiency of our proposed mechanism. The implementations and measurements of Medley TCP in Fedora 8 are presented in Chapter 6. Finally, we conclude the main work of this dissertation and point out some future work in Chapter 7.

Chapter 2

Background

A best-effort network like the Internet does not have the notion of admission control or resource reservation to control the imposed network load. Also a best-effort network under very high network load is called congested. If the network is in this state, queued packets are built up at the router buffer, and finally may lead to buffer overflow due to limited buffer size. Therefore, 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. Currently, most applications use TCP to transmit data over the Internet because TCP provides reliable data transmission with embedded congestion control algorithm which effectively removes congestion collapses in the Internet by adjusting the sending rate according to the network's available bandwidth.

TCP has several implementation versions classified into two categories. One is called the loss-based approach that uses packet loss as the only indication of congestion, e.g., TCP Reno [2, 3], TCP NewReno [12], SACK [13], and Compound TCP [17]. The other is named the delay-based scheme which makes congestion decisions to adjust the transmission rate based on round-trip time (RTT) variations, e.g., TCP Vegas [24, 25, 26, 27] and FAST TCP [29]. In the following sections, we summarize and discuss the congestion control mechanism embedded in TCP Reno, TCP NewReno, SACK, TCP Vegas, FAST TCP, and Compound TCP.

2.1 TCP Reno

TCP Reno is a window-based[1] congestion control mechanism. Its window-adjustment algorithm consists of three phases; slow start, additive increase/multiplicative decrease (AIMD[2]), and fast retransmit and fast recovery. A connection begins with the slow start phase. The objective of slow start is to enable a TCP connection to discover the available bandwidth by gradually increasing the amount of data injected into the network from the initial window size. 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 increased 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.

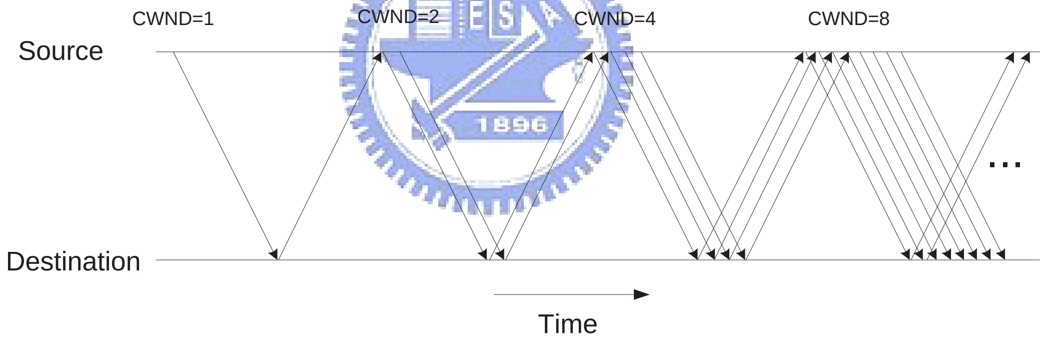


Figure 2.1: Packets in transit during slow start.

Since the CWND in the slow start phase expands exponentially, the packets sent at this increasing rate may quickly lead to network congestion. To avoid this situation, 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 CWND will be cut by half.

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

[2]Somebody also calls this phase congestion avoidance.

There are two ways for TCP Reno to detect packet loss. One is based on the reception of three duplicate ACKs, and the other is based on retransmission timeout. When a source receives three duplicate ACKs, the fast retransmit and fast 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.

The fast retransmit and fast recovery algorithm of TCP Reno allows a connection to quickly recover from isolated packet losses. However, when multiple packets are 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. Therefore, a number of enhanced loss recovery algorithms, such as TCP NewReno [12] and SACK [13], have been proposed to accommodate the above problem.

2.2 TCP NewReno

TCP NewReno has modifications to the fast retransmit and fast recovery of TCP Reno. These modifications are intended to fix the Reno problems without the addition of SACK and only have to be implemented on the sender side.

In TCP Reno, partial ACKs[3] bringing the connection out of fast recovery result in a retransmission timeout in the case of multiple packet losses. In TCP NewReno, when a source receives a partial ACK, it does not exit fast recovery [5, 19]. Instead, it assumes that the packet immediately following the most recently acknowledged packet has been lost and hence retransmits the lost packet. Thus, in the case of multiple packet losses, TCP NewReno will retransmit one lost packet per round-trip time until all the lost packets from the same window have been recovered. This avoids requiring multiple fast retransmits from a single window of data, and alleviates retransmission timeout. It remains in the fast recovery phase until all outstanding packets at the start of that phase have been acknowledged. The implementation details of TCP NewReno are as follows:

(1) Multiple packet losses: A fix for fast recovery to prevent fast retransmit and fast recovery from starting in succession when multiple segments in the same window are dropped. So far, when entering fast retransmit, the packet with the highest sequence number was sent. NewReno performs retransmission and fast recovery algorithm which is similar to Reno. However, the difference between NewReno and Reno is that when a new ACK arrives, NewReno checks whether the ACK covers the highest sequence number when fast retransmit was invoked. If not, this ACK is a partial ACK which signals that another segment was lost from the same window of data. As a result, NewReno retransmits the segment reported, as expected by the partial ACK and resets the retransmission timer, but does not exit fast recovery. On the other hand, if the new ACK covers the highest sequence number, NewReno will exit fast recovery but set CWND to the SSTHRESH and enter congestion avoidance phase.

(2) False fast recovery: NewReno records the highest sequence number ever transmitted after a retransmission timeout. Whenever three dupacks are received, NewReno performs a test to see if it should enter fast recovery. If these ACKs cover the sequence number saved at the previous timeout, then this is a new invocation

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

of the fast recovery. In this case, NewReno will enter fast recovery and perform the related actions. If these ACKs do not cover the sequence numbers and acknowledge the receipt of already queued segments at the receiver, NewReno will not enter fast recovery. A sender exits fast recovery only after all the outstanding packets (at the time of first loss) are acked.

NewReno makes a small change to a connection source: it may eliminate TCP Reno's waiting for a retransmission timeout when multiple packets from a window are lost. Although this can avoid the unnecessary window reduction, the downside is that NewReno may take many RTTs to recover a loss episode, and it must have enough new data around to keep the ACK clock running. In other words, the recovery time of NewReno is still too long.

2.3 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 successfully transmitted packets; this improves the robustness of acknowledgement when the path back to the source features a high loss rate. However, the drawback of accumulated acknowledgement is that after a packet loss, the source is unable to determine which packets have been successfully transmitted. Therefore, it is unable to recover more than one lost packet in each round-trip time.

The SACK option field contains a number of SACK blocks, where each SACK block reports a non-contiguous set of data that has been received and buffered. The destination uses ACK with the SACK option to inform the source that one contiguous block of data 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 have been missing and which have been received at the destination. A scoreboard is set up to track these transmitted and received packets according to previous information in the SACK option. For each transmitted packet, the 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 be retransmitted, but packets with the SACKed bit off and a 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.

However, there are two shortcomings of SACK. One is that a maximum of 3 or 4 SACK blocks will be allowed in a SACK option. A SACK option that specifies n blocks will have a length of $8 \times n + 2$ bytes, so the 40 bytes available for TCP options can specify a maximum of 4 blocks. In addition, it is expected that SACK will often be used in conjunction with the Timestamp option used for round-trip time measurement (RTTM), which takes an additional 10 bytes (plus two bytes of padding); thus, a maximum of 3 SACK blocks is allowed in this case. Accordingly, if there are several non-contiguous sets of data at the destination, the sender may not know the whole situation from the SACK option. The other drawback is that if the receiver runs out of buffer space, it will discard the "SACKed" packets without reporting this information to the sender. The sender will not retransmit these "SACKed" packets before it is acknowledged by the Acknowledgment Number field in the TCP header.

2.4 TCP Vegas

In this section, we first review the congestion control mechanisms of TCP Vegas in detail, the in-depth discussion of Vegas can be found in [25].

TCP Vegas [25] uses the difference in the expected and actual flow rates to estimate the available bandwidth in the network. When the network is not congested, the actual flow rate would be close to the expected flow rate. On the other hand, if the actual rate is much smaller than the expected rate, it indicates that buffer in the network is being filled up and the network is approaching congestion. This difference in flow rates can be calculated as *Diff* = *Expected* – *Actual*, where *Expected* and *Actual* are the expected and actual rates, respectively.

(i) Congestion Avoidance

In its congestion-avoidance phase, Vegas uses two threshold values, α and β (whose default values are 1 and 3, respectively), to control the adjustment of the congestion window size at the source host. Let d denote the minimum-observed packet round-trip time (also known as BaseRTT), D denote the actual RTT, and W denote the size of the congestion window size, then *Expected* = W/d and *Actual* = W/D. In addition, W is measured in segments as is normally done in any TCP version. The estimated backlog of packets in the network queues can then be computed as

$$\Delta = (Expected - Actual) \times BaseRTT = W \times \frac{(D-d)}{D}. \tag{2.1}$$

For every RTT, the congestion-avoidance algorithm adjusts W as follows:

$$W \leftarrow \begin{cases} W+1, & \text{if } \Delta < \alpha \\ W-1, & \text{if } \Delta > \beta \\ W, & \text{otherwise } (\alpha \leq \Delta \leq \beta). \end{cases}$$

Conceptually, Vegas tries to keep at least α packets but no more than β packets queued in the network. Thus, when there is only one Vegas connection, W converges to a point that lies between *window* + α and *window* + β where *window* is the maximum window size without considering the queuing in the network.

(ii) Slow Start

Like Reno, Vegas uses a slow-start mechanism that allows a connection to quickly ramp up to the available bandwidth. However, unlike Reno, to ensure that the sending rate will not increase too fast to congest the network during the slow start, Vegas doubles its congestion window size only every other RTT, and calculates the difference between the flow rates (*Diff*) and Δ given in eqn. (2.1) in every other RTT. When $\Delta > \gamma$ (whose default is 1), Vegas leaves the slow-start phase, decreases its congestion window size by 1/8 and enters the congestion-avoidance phase.

(iii) Fast Retransmit and Fast Recovery

In TCP Vegas, as in TCP Reno, a triple-duplicate ACK always results in packet retransmission. However, in order to retransmit the lost packets quickly, Vegas extends Reno's fast retransmission strategy. Vegas measures the RTT for every packet sent based on fine-grained clock values. Using these fine-grained RTT measurements, a timeout period for each packet is computed. When a duplicate ACK is received, Vegas checks whether the timeout period of the oldest unacknowledgement packet has 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 either the first or second ACK after a fast retransmission is received, Vegas again checks 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 receiving certain ACKs. This technique enables the faster detection of losses and recovery from multiple drops without restarting the slow start phase if the retransmission timer has not been expired. Hence, it allows dealing with the problem that Reno suffers from considerably, namely, multiple drops in the same data window.

After a packet retransmission is triggered by a dupack and the ACK of the lost packet is received, the congestion window size is reduced to alleviate the network congestion. There are two cases for Vegas to set the congestion window size. If the lost packet has been transmitted just once, the congestion window size will be three fourth of the previous size. Otherwise, Vegas considers the congestion as more serious, and one-half of the previous congestion window size is set as the current congestion window. Notably, in the case of multiple packet losses occurring during one RTT and triggering more than one fast retransmission, the congestion window is reduced only for the first retransmission.

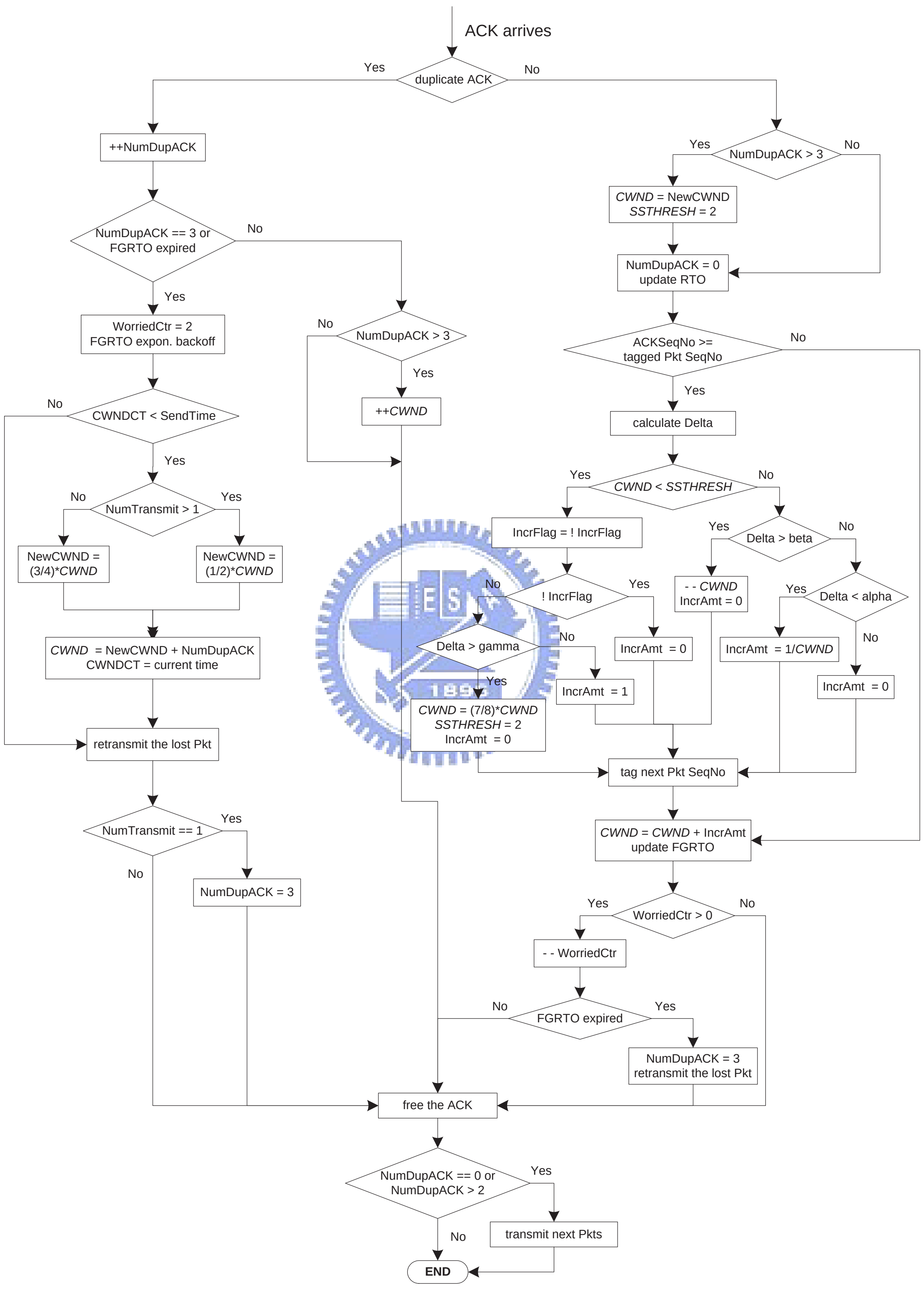


Figure 2.2: Flowchart of the procedure for TCP Vegas upon receiving an ACK.

Table 2.1: Description of variables in Fig. 2.2.

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

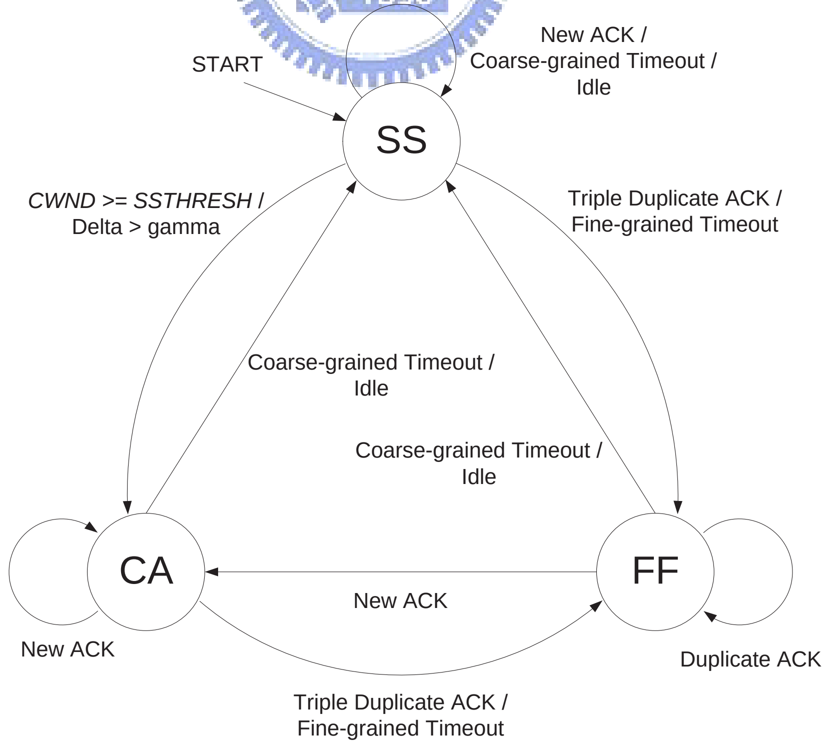

START
New ACK / Coarse-grained Timeout / Idle
SS
CWND >= SSTHRESH / Delta > gamma
Triple Duplicate ACK / Fine-grained Timeout
Coarse-grained Timeout / Idle
Coarse-grained Timeout / Idle
CA
FF
New ACK
New ACK
Duplicate ACK
Triple Duplicate ACK / Fine-grained Timeout

Figure 2.3: Phase transition diagram of TCP Vegas.

If a loss episode is severe enough that no ACKs are received to trigger the fast retransmit algorithm, the losses will be eventually identified by a Reno-style coarse-grained timeout. When this occurs, the slow start threshold is set to one-half of the congestion window size; then, the congestion window is reset to two and finally, the connection restarts from slow start. Figures 2.2 and 2.3 illustrate the detailed procedure of TCP Vegas after it received an ACK and the phase transition diagram of TCP Vegas, respectively. The description of variables used in Fig. 2.2 is listed in Table 2.1.

2.5 FAST TCP

The name FAST is a recursive acronym for Fast AQM Scalable TCP, where AQM stands for Active Queue Management. FAST TCP uses queuing delay as its main measure of congestion in its widow adjustment algorithm. In a loss-based approach, sources must periodically push buffers to overflow. However, in the absence of AQM, generating the packet loss probability may cause a jittery behavior. Delay information allows the sources to settle into a steady state when the network is static. There are two advantages to use queuing delay as the congestion measure. It provides a finer-grained measure of congestion: each measurement of packet loss (whether a packet is lost or not) provides one bit of congestion information, whereas each measurement of queuing delay provides multi-bit information, limited by clock accuracy and measurement noise. Moreover, the dynamics of delay has the right scaling with respect to link capacity that helps maintain stability as networks scale up in capacity.

Under normal network conditions, FAST periodically updated its congestion window based on the average RTT according to:

$$W \longleftarrow \min\{2W, (1-\gamma)W + \gamma(\frac{BaseRTT}{RTT}W + \alpha)\},$$

where W is the CWND, $\gamma \in (0, 1]$, and α is a positive protocol parameter that determines the total number of packets queued in routers in equilibrium along the flow's path. In fact, α is always chosen to be a constant (mostly 100). FAST TCP incorporates multiplicative increase and grows exponentially at rate α if the buffer occupied by the connection at the bottleneck is far less than α and switches to linear increase if it is near α. Then, FAST tries to maintain the buffer occupancy around α and reduces sending rate if delay is further increased. When a packet loss happens, FAST halves its window and enters loss recovery. Furthermore, although RTT measurements and the window adjustment algorithm still work during loss recovery, a source will not react to delay until it exits by loss recovery.

Theoretical analysis and experiments show that FAST TCP has better properties such as higher utilization, less self-induced packet losses, faster convergence speed, better RTT fairness and stabilization than pure loss-based approaches. However, FAST may not be able to obtain fair share when it is competing with loss-based approaches like TCP Reno. Moreover, how to set a suitable α value for a FAST source may be another possible problem since the α is a function of buffer size and the number of concurrent connections, which are generally unknown in the real Internet environment.

In summary, FAST TCP can be regarded as a scaled version of Vegas. The difference between TCP Vegas and FAST TCP lies in how the rate is adjusted when the number of packets stored is too small or too large. TCP Vegas makes fixed size adjustments to the rate, independent of how far the current rate is from the target rate. FAST TCP makes larger adjustments when the system is far from equilibrium and smaller adjustments when it is near equilibrium. This improves the speed of convergence and the stability.

2.6 Compound TCP

Compound TCP (CTCP) is a Microsoft algorithm that is part of the Windows Vista and Window Server 2008 TCP stack. The core idea of CTCP is to contain both loss-based and delay-based approaches. Like TCP Vegas and FAST TCP, CTCP uses estimates of queueing delay as a measure of congestion. If the queueing delay is small, it assumes that no link along its path is congested, and rapidly increases its rate. However, unlike Vegas and FAST, it does not intend to maintain a constant number of queued packets. The details are described in following paragraphs.

CTCP keeps the same slow start behavior of TCP Reno at the start-up of a new connection. In the congestion avoidance phase, CTCP contains two components that jointly control the sending rate of a TCP sender. So, it maintains two congestion windows: a regular AIMD window (*cwnd*) and a Delay Window (*dwnd*). The size of the actual sliding window (*win*) used is the sum of these two windows.

$$win = min(cwnd + dwnd, awnd), \tag{2.2}$$

where *awnd* is the advertised window from the receiver.

Cwnd is increased the same way as that in TCP Reno (i.e., *cwnd* is increased by one every RTT and halved upon a packet loss event). Specifically, *cwnd* is updated as follows:

$$cwnd \leftarrow \begin{cases} cwnd + \frac{1}{win}, & \text{on receiving an ACK} \\ \frac{cwnd}{2}, & \text{if loss detected.} \end{cases}$$

Dwnd is updated based on the delay information. Similar to TCP Vegas, the algorithm used in CTCP to estimate the number of backlogged packets of a connection is:

$$\begin{aligned} &Expected = win/BaseRTT \\ &Actual = win/RTT \\ &Diff = (Expected - Actual) \times BaseRTT, \end{aligned} \tag{2.3}$$

where the definitions of BasRTT, *Expected*, *Actual*, and *Diff* are same as those in Chapter 2.4. If *Diff* is smaller than a threshold γ with default value 30, the network path will be regarded as under-utilized and the *dwnd* will increase rapidly to improve the utilization of the network. Once congestion is experienced (i.e., *Diff* $\geq \gamma$), the *dwnd* gradually decreases to compensate for the increase in the AIMD window. CTCP updates its *dwnd* based on the following rules:

$$dwnd \leftarrow \begin{cases} dwnd + (\alpha \times win^k - 1)^+, & \text{if } Diff < \gamma \\ (dwnd - \xi \times Diff)^+, & \text{if } Diff \geq \gamma \\ (wind \times (1-\beta) - \frac{cwnd}{2})^+, & \text{if loss detected,} \end{cases}$$

where $(.)^+$ is defined as max(., 0). Parameters of α, β, ξ, and k are tuned to have comparable scalability to HSTCP [57] when there is free of congestion. In fact, CTCP can be regarded as a scaled version of SACK.

2.7 Chapter Summary

In this chapter, we review the design principles of several loss-based TCPs and delay-based TCPs. A loss-based TCP uses packet loss to signal that network is congested so that the source can reduce its window size accordingly. Therefore, a loss-based TCP can be concluded as a reactive congestion control mechanism. On the other hand, a delay-based TCP uses a sophisticated bandwidth estimation scheme to keep a proper amount of extra data in the network. As a result, it may prevent the system from congestion loss before it actually occurs. Thus, a delay-based TCP is a proactive congestion control mechanism.

Chapter 3

Medley TCP

3.1 Overview

TCP is a reliable connection-oriented protocol that implements flow control by means of a sliding window algorithm. TCP has experienced much success by making use of the slow start and congestion avoidance algorithms to adjust the window size, and using the fast retransmit and fast recover algorithms to save lost packets. TCP has several implementation versions classified into two categories. One is the loss-based approach that uses packet loss as the only indication of congestion. The other is the delay-based scheme which makes congestion decisions that adjust the transmission rate based on RTT variations. Both loss-based and delay-based TCPs have their own advantages and disadvantages. Also, these schemes are increasingly being stretched by the high speed networks and emergence of heterogeneity on the Internet.

We propose an end-to-end congestion control mechanism suitable for networks with high speed recourses and heterogeneous properties, called Medley TCP. It combines advantages and characteristics of both loss-based and delay-based TCPs. Also, Medley TCP only requires modification to the TCP sender and is therefore inter-operable with existing TCP: improvement can be achieved by having only sender side of the connection to employ Medley TCP.

In Section 3.2, a new congestion control algorithm for Medley TCP is addressed. Section 3.3 explains the design concept of Medley TCP. The proof is presented in Section 3.4.

3.2 Medley TCP

To ease the definition and understanding of our proposed mechanism, Medley TCP, we consider a simple case in which a single connection tries to fill up an empty network consisting of N links that connect the source and the destination as shown in Figure 3.1. We denote the transmission rate of N links (in packets/s) as χ_i, i = 1,...,N, and the total round-trip propagation delay of the end-to-end path (in seconds) as τ. We also assume that there is one bottleneck (link j, where $j \in \{1, ..., N\}$) in the route path. Because χ_j is the smallest transmission rate (i.e., link j behaves as the bottleneck link), we let μ be equal to χ_j. The un-congested bandwidth-delay product (BDP) of this network is then given by μd where

$$d = \tau + (1 + a) \sum_{i=1}^{N} \frac{1}{\chi_i}, \tag{3.1}$$

with a being the ACK size relative to the data packet size.

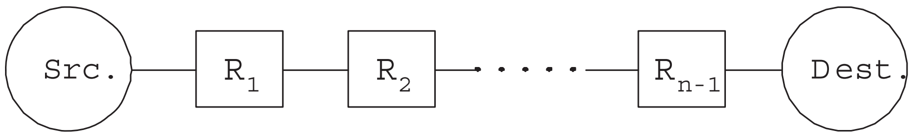


Figure 3.1: Network topology for a simple case.

A fluid model and a backlogged source are assumed. Moreover, the i^{th} RTT starts with the transmission of W_i packets where W_i is the size of congestion window in this RTT. The i^{th} RTT ends when the source receives the ACK of the first packet in this RTT, then the source starts transmitting a new packet of the next RTT. Suppose that there is no congestion in ACK path.

3.2.1 Notation

Here we collect the definitions of all the variables used in Medley TCP and some of their obvious properties.

d	minimum-observed packet round-trip time.
D	actual RTT.
W	the congestion window size.
E	expected throughput, W/d.
A	actual throughput, W/D.
I	amount W to be increased (decreased) in the next RTT. Its default is 0.
Δ	the difference between *Expect* and *Actual* (as given in eqn. (2.1)).
Δ_{last}	the Δ in the last RTT.
Γ	the largest influence of increasing I packets on the bottleneck when $I > 0$. The equation of Γ is similar to that of Δ. When $I > 0$, update $\Gamma = \frac{W \times I}{2 \times A \times d + I}$. We will describe it in the next paragraph.
R	a record for the reason of a packet loss. If it is caused by congestion loss, set R 'True'. Otherwise, R is 'False'.

3.2.2 The Detail of Γ

Medley TCP increases (or decreases) its congestion window size by I. When Medley TCP increases its window in this RTT, the last packet will see at most I packets queued in the bottleneck. Without loss of generality, we consider the worst case that the last packet will see I packets queued in the bottleneck when the throughput of Medley TCP approaches the available bandwidth. Therefore, this last packet experiences the highest RTT, D, and $D = d + \frac{I}{\mu}$. In an actual Medley TCP implementation, D is the smoothed RTT rather than the RTT of a specific packet. Thus, D for the last packet is the average of the actual RTTs of all packets in the same RTT period, i.e., $D = d + \frac{I}{2 \times \mu}$, rather than $D = d + \frac{I}{\mu}$, as given above.

From W, I, D, and d, we can get Γ as below.

$$\Gamma = (\frac{W}{d} - \frac{W}{D})d = W\frac{D-d}{D} = \frac{W \times I}{2 \times \mu \times d + I}. \tag{3.2}$$

In the steady state, $\mu \simeq W/D$, so we replace μ to W/D in the eqn. (3.2). Note that W/D is also actual throughput A. Hence, the equation (3.2) can be written to

$$\Gamma = \frac{W \times I}{2 \times A \times d + I}. \tag{3.3}$$

If Medley TCP decreases its window size, Δ will be larger than Γ. Accordingly, there is no reason to re-calculate Γ in the next RTT[1].

3.2.3 Algorithm

Medley TCP determines congestion window based on congestion information – queueing delay and packet loss, provided by above equations. Medley TCP periodically updates its congestion window size according to:

$$W \leftarrow \begin{cases} \min\{W+I, 2W\}, & \text{if there is no loss} \\ 3W/4, & \text{if loss is detected,} \end{cases}$$

where

$$I \leftarrow \begin{cases} 2, & \text{if } ((\Delta \le \Gamma) \wedge (I < 0)) \vee ((\Delta > \Gamma) \wedge (I = -1) \wedge (\Delta < \Delta_{last})) \\ I+2, & \text{if } (\Delta \le \Gamma) \wedge (I \ge 0) \\ \frac{1}{2}, & (\Delta > \Gamma) \wedge (R = True) \\ -1, & otherwise. \end{cases}$$

In the next section, we will explain why we design such a formula for I. As for the fast retransmit and fast recovery part of Medley TCP, it is quite similar to that of TCP Vegas, as described in Chapter 2.4. However, there are two differences between Medley TCP's and TCP Vegas' fast retransmit and fast recovery algorithms. One is that Medley TCP tries to distinguish between congestion loss and random loss. If Medley TCP enters fast retransmit and fast recovery with $\Delta > \Gamma$, it is very

[1]In fact, Γ will be negative if re-calculating it. Moreover, it is illegitimate for queuing a negative number of packets.

likely that congestion occurs in the bottleneck. Therefore, Medley TCP will set R to 'True'. Otherwise, it is most likely a random loss. The other difference is that I is set to 0 when Medley TCP enters fast retransmit and fast recovery. This is because the fast recovery algorithm will compensate a user for lost packets in this RTT. Accordingly, extra window size increment may not be needed.

3.3 Exposition for I

The key idea of varying I is that if a link is under-utilized, Medley TCP should be aggressive in increasing its sending rate to obtain available bandwidth more quickly. However, once the link is fully utilized, being aggressive is no longer good, as it may cause problems such as TCP unfairness. We note that a delay-based approach has a nice property of adjusting its throughput when the link is fully utilized. However, the major weakness of delay-based approaches is that they could not compete with loss-based approaches when delay-based TCPs and loss-based TCPs coexist in a network. Also, this weakness is difficult to be remedied by delay-based approaches themselves. Therefore, in order to adjust the congestion window size more effectively, Medley TCP incorporates an adaptive delay-based component into a loss-based TCP algorithm. This adaptive delay-based component has a rapid window increase rule when the network is sensed of under-utilization and gracefully reduces the sending rate once the bottleneck queue is built.

If the estimated backlog of packets is no larger than the largest influence of increasing I packets in the network (i.e., $\Delta \leq \Gamma$,) I could be increased more and more rapidly since the network may be under-utilized. For this reason, I will be added by 2, which is contributed with 1 from a loss-based component and 1 from a delay-based part, when $\Delta \leq \Gamma$ and $I \geq 0$. If $I < 0$, I will be set to 2 directly. On the other hand, while $\Delta > \Gamma$, the congestion window size should be decreased in order to avoid a network congestion. Accordingly, Medley TCP will decrease its window size by 1, which is contributed with 1 from a loss-based component and -2 from a delay-based part (i.e., $I = -2 + 1 = -1$.) However, the TCP fairness and friendliness issues should be considered. Hence, after Medley TCP decreased its window size in the last RTT, if the estimated backlog of packets in this RTT is smaller than that in the last RTT (i.e., $(\Delta > \Gamma) \wedge (I = -1) \wedge (\Delta < \Delta_{last})$), Medley TCP will increase the congestion window size instead of decreasing it. More detailed reasons and mathematical proof are described in Section 3.4.

Suppose a TCP SACK's[2] congestion window size (W) equals a Medley TCP's when congestion occurs in the bottleneck. Then, the window sizes of SACK and Medley TCP are reduced to $W/2$ and $3W/4$, respectively. Because the SACK increases its window by 1 every RTT, it will need $W/2$ RTTs to recover its window size, W. If we also want Medley TCP to spend $W/2$ RTTs on recovering, the increase rate should be 1/2 (i.e., $(W - \frac{3W}{4})/(W - \frac{W}{2}) = 1/2$). Accordingly, the value of I is 1/2 when R is 'True'.

3.4 Proofs

Consider that a loss-based TCP and a Medley TCP coexist in a network to share a bottleneck link. If Medley TCP only reduces its congestion window when $\Delta > \Gamma$, it may not be able to obtain its fair share. Since the Medley TCP flow tries to maintain a small number of packets in the bottleneck queue (i.e., Γ,) it will stop increasing its sending rate until $\Delta \leq \Gamma$. However, the loss-based flow will not react to the increase of delay, and continues to increase the sending rate. This, observed by the Medley TCP flow, is considered as congestion indication and therefore the sending rate of the Medley TCP flow is further reduced. In this way, the Medley TCP flow may obtain far less bandwidth than its fair share. Therefore, in some condition with $\Delta > \Gamma$, Medley TCP can increase its sending rate.

Proposition 1: After decreasing sending rate, Medley TCP can increase its congestion window size if $\Delta < \Delta_{last}$ and $\Delta > \Gamma$.

Proof: Suppose that there are m loss-based TCP and n Medley TCP flows in a dumbbell network topology, $\forall\ m, n > 0$. At the i^{th} RTT, Q packets are queued in

[2]This TCP version is the most used in the current Internet.

the bottleneck buffer and each Medley TCP gets $\Delta_i > \Gamma$. Therefore, m loss-based TCP flows increase m packets and n Medley TCP connections decrease n packets in the $(i+1)^{th}$ RTT. Note that $\Delta_{i+1} = (W-1)(Q+m-n)/(2\mu d+Q+m-n)$ and $\Delta_i = WQ/(2\mu d+Q)$, where W is the window size in the i^{th} RTT.

(a) $m = n$:

$$\begin{aligned}\Delta = \Delta_{i+1} &= (W-1)\frac{Q+m-n}{2\mu d+Q+m-n} = (W-1)\frac{Q}{2\mu d+Q} \\ &< W\frac{Q}{2\mu d+Q} = \Delta_i = \Delta_{last}\end{aligned}$$

(b) $m < n$:

$$\begin{aligned}\Delta = \Delta_{i+1} &= (W-1)\frac{Q+m-n}{2\mu d+Q+m-n} \\ &< W\frac{Q}{2\mu d+Q} = \Delta_i = \Delta_{last} \\ \because \frac{b}{a} &> \frac{b-c}{a-c} \quad \forall\, a,b,c > 0\end{aligned}$$

(c) $m > n$: The value of Q is getting larger until a packet loss occurs.

For case (a) and (b), since packets increased by loss-based flows are no more than those decreased by Medley TCP connections, the number of queuing packets will not increase (no packet loss occur) and the loss-based flows will continue to increase the sending rate. It is conceivable that if Medley TCPs do not increase their window size in this RTT, they may obtain far less bandwidth than their fair share. On the other hand, for the last case (c), if Medley TCP also increases its congestion window size, it may suffer a serious packet loss when buffer overflows. Therefore, we think Medley TCP can increase its sending rate in cases (a) and (b) (i.e., $\Delta < \Delta_{last}$ and $\Delta > \Gamma$.)

$\square$

According to the above reason and mathematical proof, Medley TCP will increase the congestion window size when $(\Delta > \Gamma) \wedge (I = -1) \wedge (\Delta < \Delta_{last})$. Furthermore, the increase rule is $I = 2$ because Medley TCP wants to recover its window size, which was decreased by 1 in the last RTT, and increases it by 1 in this RTT.

Chapter 4

Analysis and Model

In this Chapter, we analyze one Medley TCP's basic behavior, establish a model for its performance evaluation and use examples to verify the correctness of the model (compared with simulation results). After knowing Medley TCP's behavior, we demonstrate that two (and more) Medley TCP flows with same RTT converge to their fair shares, finally we study its RTT fairness in the last section.

4.1 Basic Behavior Analysis

Similar to Chapter 3.2, we consider a simple case when a single connection tries to fill up an empty network with N links connecting the source and the destination, as shown in Figure 3.1. We denote the transmission rate of N links (in packets/s) as χ_i, i = 1,...,N, and the total round-trip propagation delay of the end-to-end path (in seconds) as τ. We also assume that there is one bottleneck (link j, where $j \in \{1, ..., N\}$) in the end-to-end path. Because χ_j is the smallest transmission rate (i.e., link j behaves as the bottleneck link), we let μ be equal to χ_j. The un-congested BDP of this network is then given by μd where

$$d = \tau + (1 + a) \sum_{i=1}^{N} \frac{1}{\chi_i}, \tag{4.1}$$

with a being the ACK size relative to the data packet size.

A fluid model is assumed and the source always has a packet to transmit. Moreover, the i^{th} RTT starts with the transmission of W_i packets where W_i is the size of congestion window in this RTT. The i^{th} RTT ends when the source receives the ACK of the first packet in this RTT, then the source starts transmitting a new packet of the next RTT. Suppose that there is no congestion in ACK path and $\Delta > \Gamma$ at ${t_1}^{th}$ RTT, ${t_2}^{th}$ RTT, ${t_3}^{th}$ RTT, ..., respectively.

Before calculating t_1, t_2, and so on correctly, we need to introduce an important parameter, average queue occupancy (Q). This is because the Q value affects the results of Δ and Γ. The Q_i is the average of W_i packets' queue occupancies in the i^{th} RTT. The equation of Q_i is as follows,

$$Q_i = \frac{I_{i-1} \times (W_{i-1} + \frac{I_{i-1}}{2})}{W_i} = \frac{I_{i-1} \times (2W_{i-1} + I_{i-1} - 1)}{2W_i}. \tag{4.2}$$

4.1.1 Calculate t_1

From our proposed algorithm in Chapter 3.2.3, we can get

$$W_i = i^2 - i + 2 \quad \text{and} \quad I_i = 2 \times i - 2, \tag{4.3}$$

where $i \in \mathrm{N}$, $W_0 = 2$, and $I_0 = 0$, before Medley TCP first decreases its congestion window size. Table 4.1 shows an example of all values of W_i and I_i at the i^{th} RTT before Δ_{t_1} becomes larger than Γ_{t_1} (i.e., $i \le t_1$).

Table 4.1: The window size and the increase amount at the i^{th} RTT with $i \le t_1$.

i^{th}	0	1	2	3	4	5	6	7	8	9	10	11	...
W_i	2	2	4	8	14	22	32	44	58	74	92	112	...
I_i	0	0	2	4	6	8	10	12	14	16	18	20	...

The process of calculating t_1:

$$\Delta_{t_1} = \frac{W_{t_1}(D_{t_1} - d)}{D_{t_1}} > \frac{W_{t_1} \times I_{t_1 - 1}}{2 \times A \times d + I_{t_1 - 1}} = \Gamma_{t_1}$$

$$\Rightarrow \quad \frac{Q_{t_1}}{\mu d + Q_{t_1}} > \frac{(\mu d + Q_{t_1})I_{t_1-1}}{2W_{t_1}\mu d + (\mu d + Q_{t_1})I_{t_1-1}}$$

$$\Rightarrow \quad 2W_{t_1}Q_{t_1} - (\mu d + Q_{t_1})I_{t_1-1} > 0$$

$$(\because W_i = W_{i-1} + I_{i-1}, \quad Q_i = \frac{I_{i-1}(2W_{i-1} + I_{i-1} - 1)}{2W_i})$$

$$\Rightarrow \quad 2I_{t_1-1}W_{t_1-1} + I_{t_1-1}(I_{t_1-1}) - (\mu d + Q_{t_1})I_{t_1-1} > 0$$

$$\Rightarrow \quad 4W_{t_1}W_{t_1-1} + 3I^2_{t_1-1} - 2W_{t_1}(\mu d + 1) - I_{t_1-1} > 0$$

$$(\because W_i = i^2 - i + 2, \quad I_i = 2i - 2)$$

$$\Rightarrow \quad 4(t_1^2 - t_1 + 2)((t_1 - 1)^2 - (t_1 - 1) + 2) + 3(2t_1 - 4)^2$$
$$-2(t_1^2 - t_1 + 2)(\mu d + 1) - (2t_1 - 4) > 0$$

$$\Rightarrow \quad 2t_1^4 - 8t_1^3 + (23 - \mu d)t_1^2 - (44 - \mu d)t_1 + (28 - 2\mu d) > 0 \tag{4.4}$$

Because equation (4.4) is a quartic equation, we could use Matlab to solve t_1. Therefore, when Δ_{t_1} is larger than Γ_{t_1}, the BaseRTT, actual RTT, increment, and window size are given by d, $d + Q_{t_1}/\mu$, I_{t_1-1}, and W_{t_1}, respectively.

4.1.2 Calculate t_j

After Medley TCP decreases its window size in the t_j^{th} and $(t_j + 1)^{th}$ RTTs, it will increases its sending rate until $\Delta_{t_{j+1}}$ is larger than $\Gamma_{t_{j+1}}$, where $j = 1, 2, \ldots$. Moreover, due to the decrement before, the equation of Q_i needs a slight modification as follows,

$$Q_i = \frac{I_{i-2} \times (W_{i-2} + \frac{I_{i-2}}{2})}{W_{i-2}} = \frac{I_{i-2} \times (2W_{i-2} + I_{i-2} - 1)}{2W_{i-2}}, \tag{4.5}$$

where $i > t_1$.

Based on Medley TCP's algorithm in Chapter 3.2.3, we can get

$$W_{t_j} = W_{t_{j-1}} + (t_j - t_{j-1})^2 - 3(t_j - t_{j-1}) \quad \text{and} \quad I_{t_j} = 2(t_j - t_{j-1} - 1), \tag{4.6}$$

where $j = 2, 3, ...,$ until Medley TCP finishes its work and terminates. An example of the congestion window size and increment amount with some t_j is shown in Table 4.2, where $j > 1$.

Table 4.2: The window size and the increase amount with some t_j, where $j > 1$.

i^{th}	t_{j-1}	$t_{j-1}+1$	$t_{j-1}+2$	$t_{j-1}+3$	$t_{j-1}+4$	$t_{j-1}+5$	t_j
W_i	$W_{t_{j-1}}$	$W_{t_{j-1}}-1$	$W_{t_{j-1}}-2$	$W_{t_{j-1}}$	$W_{t_{j-1}}+4$	$W_{t_{j-1}}+10$	W_{t_j}
I_i	-1	-1	2	4	6	8	-1

The process of calculating t_j:

$$\Delta_{t_j} = \frac{W_{t_j}(D_{t_j} - d)}{D_{t_j}} > \frac{W_{t_j} \times I_{t_j-1}}{2 \times A \times d + I_{t_j-1}} = \Gamma_{t_j}$$

$$\Rightarrow \quad \frac{Q_{t_j}}{\mu d + Q_{t_j}} > \frac{(\mu d + Q_{t_j})I_{t_j-1}}{2W_{t_j}\mu d + (\mu d + Q_{t_j})I_{t_j-1}}$$

$$\Rightarrow \quad 2W_{t_j}Q_{t_j} - (\mu d + Q_{t_j})I_{t_j-1} > 0$$

(by eqn. (4.5))

$$\Rightarrow \quad 2W_{t_j}\frac{I_{t_j-2}(2W_{t_j-2} + I_{t_j-2} - 1)}{2W_{t_j-2}} - (\mu d + \frac{I_{t_j-2}(2W_{t_j-2} + I_{t_j-2} - 1)}{2W_{t_j-2}})I_{t_j-1} > 0$$

$$\Rightarrow \quad 2W_{t_j}I_{t_j-2}(2W_{t_j-2} + I_{t_j-2} - 1)$$
$$-(2\mu d W_{t_j-2} + I_{t_j-2}(2W_{t_j-2} + I_{t_j-2} - 1))I_{t_j-1} > 0$$

$$\Rightarrow \quad 4W_{t_j}W_{t_j-2}I_{t_j-2} + 2W_{t_j}I_{t_j-2}(I_{t_j-2} - 1) - 2\mu d W_{t_j-2}I_{t_j-1}$$
$$-2W_{t_j-2}I_{t_j-1}I_{t_j-2} - I_{t_j-1}I_{t_j-2}(I_{t_j-2} - 1) > 0$$

(by eqn. (4.6) and let $a = t_j - t_{j-1},\ b = W_{t_{j-1}}$)

$$\Rightarrow \quad 4(b + a^2 - 3a)(b + a^2 - 7a + 10)(2a - 6) + 2(b + a^2 - 3a)(2a - 6)(2a - 7)$$
$$-2(b + a^2 - 7a + 10)(2a - 4)\mu d - 2(b + a^2 - 7a + 10)(2a - 4)(2a - 6)$$

$$-(2a-6)(2a-7)(2a-4) > 0$$

$$\Rightarrow \quad 2a^5 - 26a^4 + (125+4b-\mu d)a^3 - (271+32b-5\mu d)a^2$$
$$+(2b^2+77b+254+(4-b)\mu d)a - (6b^2+51b+78+(20-2b)\mu d) > 0 \quad (4.7)$$

Similar to solving a quartic equation, we also could use Matlab to find the value of a because equation (4.7) is a quintic equation. Therefore, when Δ_{t_j} is larger than Γ_{t_j}, the BaseRTT, actual RTT, increment, and window size are given by d, $d + Q_{t_j}/\mu$, I_{t_j-1}, and W_{t_j}, respectively, where $j = 2, 3,$ and so on.

$\square$

4.2 Performance Model

From above discussions, we have a series of $\{t_i, W_{t_i}\}$ sets, where $i \in \mathrm{N}$. By using these sets, the throughput of Medley TCP can be calculated. Let Λ and $\Lambda_{(t_{i-1},t_i)}$ denote the whole throughput and the throughput from t_{i-1}^{th} RTT to t_i^{th} RTT, respectively, where $i \in \mathrm{N}$ and $t_0 = 0$. Moreover,

$$\Lambda = \sum_{i=1}^{\infty} \Lambda_{(t_{i-1},t_i)}. \quad (4.8)$$

When getting all $\Lambda_{(t_{i-1},t_i)}$, the whole throughput is also known. However, it is hard to calculate each throughput from t_{i-1}^{th} RTT to t_i^{th} RTT by a linear function. Therefore, we compute the lower bound and the upper bound for each $\Lambda_{(t_{i-1},t_i)}$.

4.2.1 The Lower Bound and Upper Bound of $\Lambda_{(t_0,t_1)}$:

Lower bound:

$$\begin{aligned}\sum_{i=0}^{t_1} W_i &= \sum_{i=0}^{t_1}(i^2 - i + 2) = \sum_{i=0}^{t_1} i^2 - \sum_{i=0}^{t_1} i + \sum_{i=0}^{t_1} 2 \\ &= \frac{1}{6}t_1(t_1+1)(2t_1+1) - \frac{1}{2}t_1(t_1+1) + 2(t_1+1) \\ &= \frac{1}{3}(t_1^3 + 5t_1 + 6)\end{aligned}$$

Upper bound:

$$\frac{1}{2}(W_{t_1}+2)t_1 = \frac{1}{2}(t_1-t_0)(W_{t_1}+W_{t_0})$$

From above, we can get

$$\frac{1}{3}(t_1^3+5t_1+6) \le \Lambda_{(t_0,t_1)} \le \frac{1}{2}(t_1-t_0)(W_{t_1}+W_{t_0}) \tag{4.9}$$

4.2.2 The Lower Bound and Upper Bound of $\Lambda_{(t_1,\infty)}$:

Lower bound:

$$\begin{aligned} \because\ & (t_i-t_{i-1})W_{t_{i-1}} \le \Lambda_{(t_{i-1},t_i)} \\ \therefore\ & \sum_{i=2}^{\infty}(t_i-t_{i-1})W_{t_{i-1}} \le \Lambda_{(t_1,\infty)} \end{aligned}$$

Upper bound:

$$\begin{aligned} \because\ & \Lambda_{(t_{i-1},t_i)} \le \frac{1}{2}(t_i-t_{i-1})(W_{t_{i-1}}+W_{t_i}) \\ \therefore\ & \Lambda_{(t_1,\infty)} \le \sum_{i=2}^{\infty}\frac{1}{2}(t_i-t_{i-1})(W_{t_{i-1}}+W_{t_i}) \end{aligned}$$

From above, we can get

$$\sum_{i=2}^{\infty}(t_i-t_{i-1})W_{t_{i-1}} \le \Lambda_{(t_1,\infty)} \le \sum_{i=2}^{\infty}\frac{1}{2}(t_i-t_{i-1})(W_{t_{i-1}}+W_{t_i}) \tag{4.10}$$

4.2.3 The Lower Bound and Upper Bound of Λ:

According to equations (4.9) and (4.10), the relation of Λ's lower bound and upper bound is as follows.

$$\frac{1}{3}(t_1^3+5t_1+6) + \sum_{i=2}^{\infty}(t_i-t_{i-1})W_{t_{i-1}} \le \Lambda \le \sum_{i=1}^{\infty}\frac{1}{2}(t_i-t_{i-1})(W_{t_{i-1}}+W_{t_i}) \tag{4.11}$$

□

4.3 Examples

The network topology of one single link is shown in Fig. 4.1, where S1 represents a sender host using Medley TCP. The type of service used here is FTP. The receiver sends an ACK for every data packet received. For the convenience of presentation, we assume that all window sizes are measured in number of fixed-size packets, which are 1000 bytes. R1 and R2 represent two finite-buffer gateways. The buffer size in each gateway is set to 100 packets. For the constant-load experiment, drop-tail gateways with first in first out (FIFO) service are assumed. The bandwidth of access links are 1Gbps, and propagation delays are 1ms. The bandwidth of connection link is X-Mbps and propagation delay is Y-ms. By varying X and Y, this communication network can represent networks with different characteristics such as small-bandwidth, large-bandwidth, short-delay, and long-delay networks.

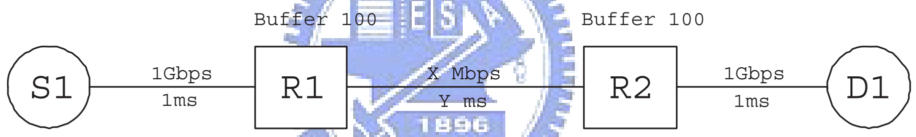


Figure 4.1: A network topology for analysis and simulation.

In the first example, we set bottleneck bandwidth to 50 Mbps and BaseRTT to 0.1001664 second (i.e., $X = 50$ and $Y = 48$). From equation (4.4), we can get $t_1 > 19.0866$. Because t_1 is a positive integer, t_1 must be 20 and W_{t_1} is 382 (by equation (4.3)). By using equations (4.7) and (4.6), t_2, W_{t_2}, t_3, W_{t_3}, etc. can be calculated as 28, 422, 34, 440, and so on, respectively. Furthermore, with above values, we also get the lower bound and upper bound of Medley TCP's throughput. In order to verify the correctness of the model, we compare model results with simulation results. The simulation experiments are conducted using *ns2* [58], version 2.29, developed at Lawrence Berkeley National Laboratories (LBNL). The results of model and simulation, and the outcome of model, lower bound, and upper bound are shown in Fig. 4.2 and Fig. 4.3, respectively. From these two figures, we could find that the simulation results match closely with our analytical model.

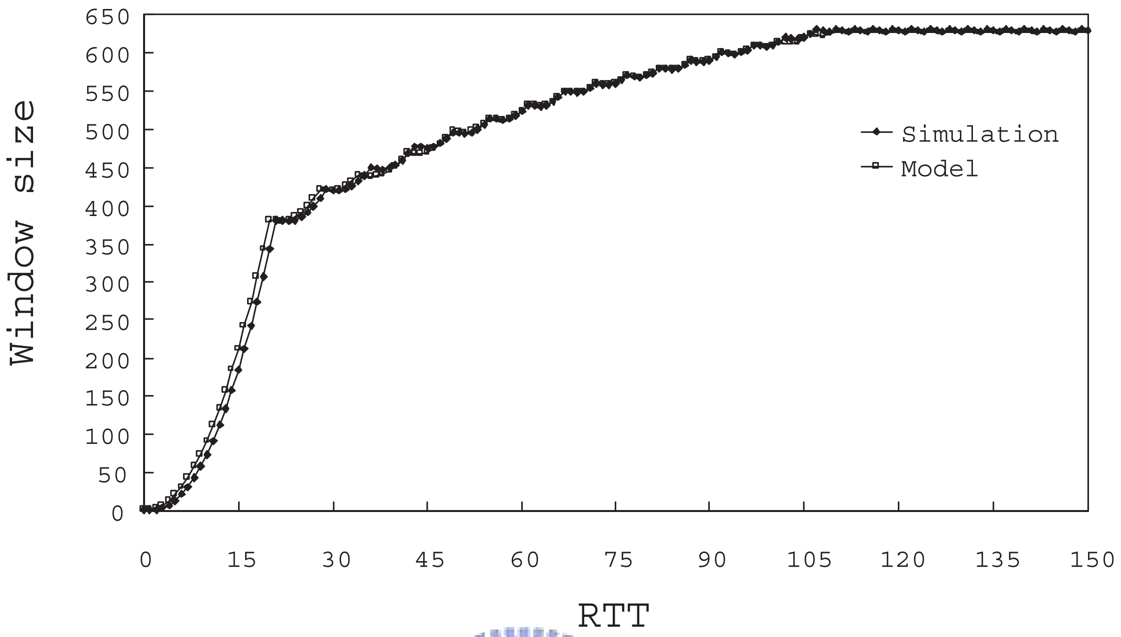


Figure 4.2: The model and simulation results.

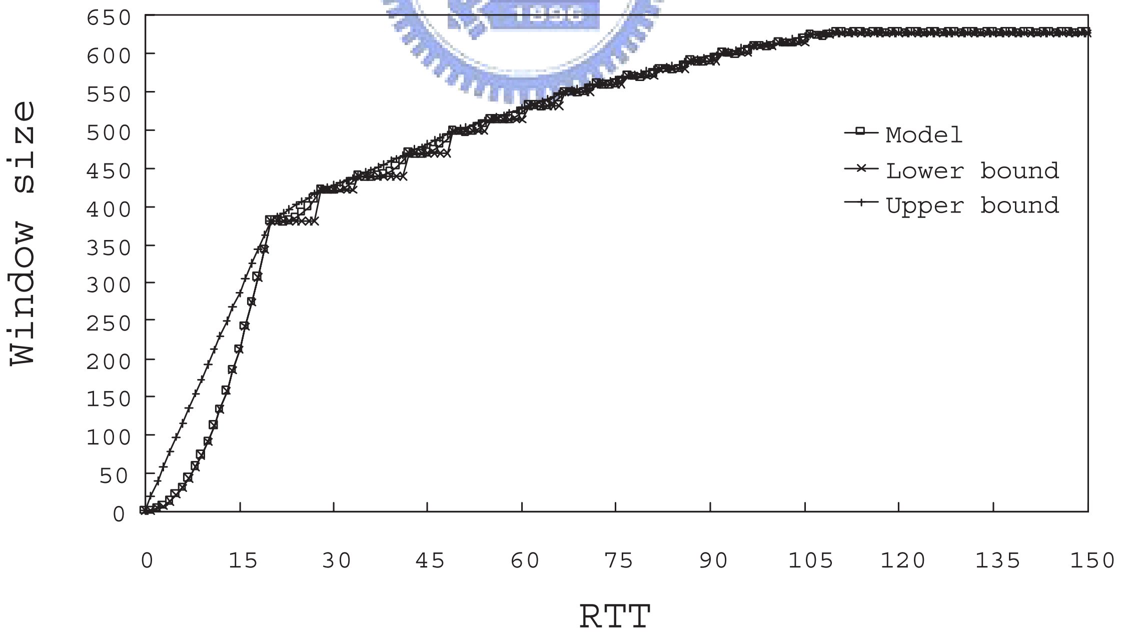


Figure 4.3: The outcome of model, lower bound, and upper bound.

With high BDP networks, the transient period of TCP can greatly affect overall performance. We use a metric “convergence time” [51] to capture the transient performance of our proposed mechanism. Convergence time indicates how many BaseRTTs are required to reach a new stable state or period. Figure 4.4 shows the convergence time of both analytical model and simulation in networks with different BDPs. The difference between simulation and model is very small. According to these experiments, we may say that our model captures Medley TCP’s basic behavior.

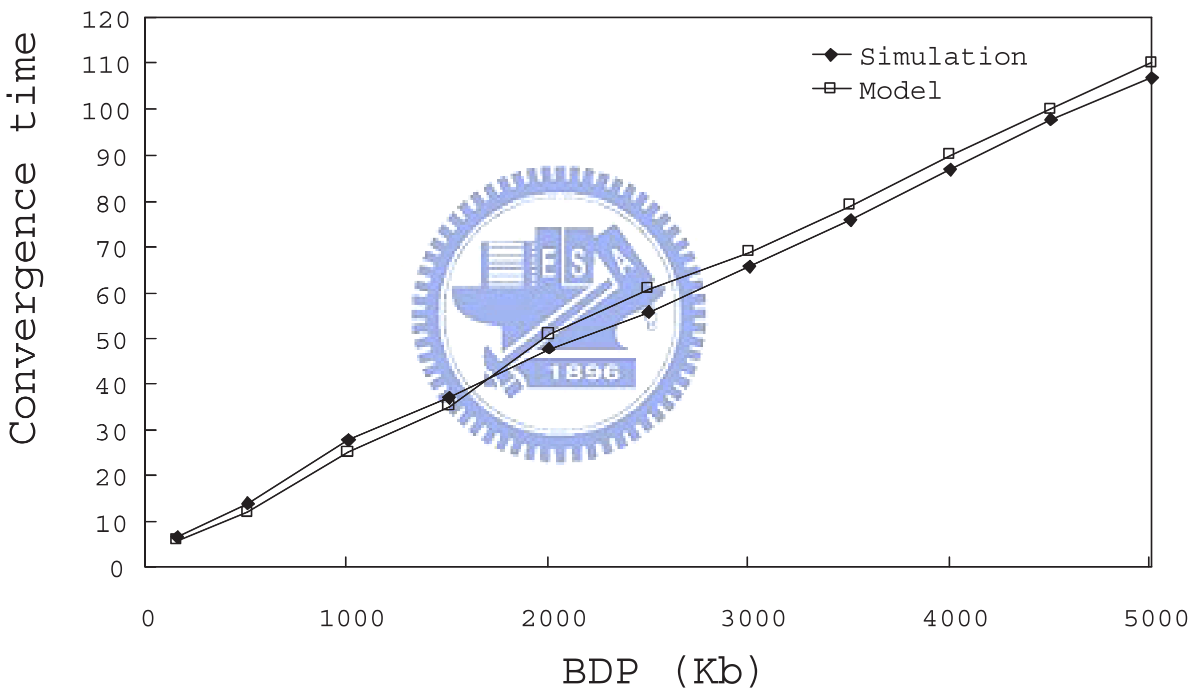


Figure 4.4: The convergence time of model and simulation in networks with different BDPs.

4.4 Fair Shares

The property of fair shares and fairness is that a new TCP protocol has good intra-protocol fairness, even when the competing flows have different RTTs.

4.4.1 Same RTT Convergence

We first demonstrate two Medley TCP flows' fair shares and then extend to a common case (with N flows).

Theorem 2: Two Medley TCP flows converge to their fair share under the network topology as shown in Figure 3.1 with same round trip delay.

Proof: We use Jain's fairness index [59], which is defined as follows,

$$F(x) = \frac{(\sum x_i)^2}{n(\sum x_i^2)}. \tag{4.12}$$

Let's consider two Medley TCP flows, whose window size are x_1 and x_2, respectively. We assume $x_1 < x_2$. We see $\triangle F \geq 0$, if and only if $\triangle x_1/x_1 \geq \triangle x_2/x_2$ [17]. In increasing period ($\Delta \leq \Gamma$), $\triangle x = I$ for some $I \geq 2$, it is easy to see that $I/x_1 > I/x_2$, and therefore, $\triangle F > 0$. When they reach the steady state or queuing delay is detected, the increase/decrease amount I of two flows will be the same (i.e., -1, -1, and 2). During this period, the fairness index keeps the same value because $-1 + (-1) + 2 = 0$. While the packet loss is detected[1], both rates are decreased by three-fourths. So, the fairness index still keeps the same value. In summary, in each period of the control law, the fairness index is strictly none decreasing. Therefore, eventually, the two Medley TCP flows converge to their fair share.

$\square$

Corollary 3: N Medley TCP flows converge to their fair share under the network topology as shown in Figure 3.1 with same round trip delay.

Proof: Similar to the proof of Theorem 2, Jain's fairness index is used. Let's consider N Medley TCP flows, whose window size are x_1, x_2, ..., x_N, respectively. We assume $x_i < x_j$, $\forall\ i < j$. We see $\triangle F \geq 0$, if and only if $\triangle x_i/x_i \geq \triangle x_j/x_j$, $\forall\ i < j$. In increasing period ($\Delta \leq \Gamma$), $\triangle x = I$ for some $I \geq 2$, it is easy to see that $I/x_i > I/x_j$, and therefore, $\triangle F > 0$, $\forall\ i < j$. When they reach the steady state or queuing delay is detected, the increase/decrease amount I of all flows will

[1]When all sources use Medley TCP algorithm in a wired network, the probability for the packet loss is very small.

be the same. During this period, the fairness index keeps the same value because $-1 + (-1) + 2 = 0$. While the packet loss is detected, all rates are decreased by three-fourths. So, the fairness index still keeps the same value. In summary, in each period of the control law, the fairness index is strictly none decreasing. Therefore, eventually, the N Medley TCP flows converge to their fair share.

$\square$

4.4.2 Fairness with Different RTTs

Here, we consider the case where different Medley TCP flows may have different delays. Similar to the above paragraph, we study two flows' RTT fairness and then go over a common case.

Theorem 4: Let Th_1 and Th_2 represent the throughput of two Medley TCP flows with minimum round trip time as d_1 and d_2, respectively. Then, the following equality is satisfied

$$\frac{Th_1}{Th_2} \approx 1.$$

Proof: The delay-based TCP reacts to the increase of round trip time and tries to maintain certain number of packets backlogged in the bottleneck queue [17, 11]. We assume, in the steady state, $\Gamma_1 = \Gamma_2$, where Γ is given in equation (3.2). According to [29], the throughput of a delay-based TCP is approximately proportional to increment and queue occupancy. So, $Th \approx I/\Gamma$, where I is the increase/decrease amount.

$$\begin{aligned}
\frac{Th_1}{Th_2} &\approx (\frac{I_1}{\Gamma_1})/(\frac{I_2}{\Gamma_2}) = \frac{2\mu d_1 + I_1}{W_1} \times \frac{W_2}{2\mu d_2 + I_2} \\
& \quad (\because\ \Gamma = \frac{WI}{2\mu d + I},\ \text{where } W \text{ is the window size}) \\
&= \frac{2\mu d_1 + I_1}{W_1} \times \frac{W_1}{2\mu d_1 + I_1} \times \frac{I_1}{I_2} \\
& \quad (\because\ \Gamma_1 = \Gamma_2) \\
&= \frac{I_1}{I_2} \approx 1 \\
& \quad (\because\ \text{in the steady state, } I_1 \approx I_2.)
\end{aligned}$$

Hence, each connection could get roughly the same throughput.

□

Corollary 5: N Medley TCP connections could get roughly the same throughput, even when their round trip times are different.

Proof: In the steady state, we assume, like above proof does, $\Gamma_1 = \Gamma_2 = ... = \Gamma_N$ and $Th \approx I/\Gamma$. For all i and j,

$$\begin{aligned}
\frac{Th_i}{Th_j} &\approx (\frac{I_i}{\Gamma_i})/(\frac{I_j}{\Gamma_j}) = \frac{2\mu d_i + I_i}{W_i} \times \frac{W_j}{2\mu d_j + I_j} \\
& \quad (\because\ \Gamma = \frac{WI}{2\mu d + I},\ \text{where } W \text{ is the window size}) \\
&= \frac{2\mu d_i + I_i}{W_i} \times \frac{W_i}{2\mu d_i + I_i} \times \frac{I_i}{I_j} \\
& \quad (\because\ \Gamma_i = \Gamma_j) \\
&= \frac{I_i}{I_j} \approx 1 \\
& \quad (\because\ \text{in the steady state, } I_i \approx I_j.)
\end{aligned}$$

Hence, each connection could get roughly the same throughput.

□

Chapter 5

Experiment Evaluation

5.1 Overview

Throughput, fairness, and friendliness are three main metrics for TCP evaluation. This chapter reports simulation results by using *ns2* [58] simulator. The results show that Medley TCP not only achieves significant improvements on throughput both in large BDP networks and wireless networks, but also experiences smaller queue occupancy at the bottleneck than other TCP versions do. When all sources use same TCP version to share network resources, the Medley TCP's fairness is no worse than other TCP versions'.

We also study Medley TCP's behavior when competing with SACK's[1] connections for the bandwidth on the bottleneck link in the network experiments. Many studies [10, 34, 35, 36, 37] and live Internet measurement have demonstrated that when a delay-based TCP user competes with a loss-based TCP user, it does not receive a fair share of bandwidth due to the conservative congestion avoidance algorithm used by a delay-based TCP. However, simulation results give an indication that Medley TCP flows can co-exist harmoniously with SACK flows. It can be explained that Medley TCP's better throughput is attributed to better utilization of the available bandwidth rather than "stealing" bandwidth from SACK's connec-

[1]SACK TCP is commonly used by operating systems in the current Internet.

tions.

The rest of this chapter is organized as follows. Section 5.2 shows the simulation results of throughput, convergence time, and queue length demand for one SACK, CTCP, FAST TCP[2], and Medley TCP connection. The bandwidth utilization and fairness of same TCP version flows are included in Section 5.3. In Section 5.4, we evaluate four TCP versions performance in wireless networks. The competition results (or so-called TCP friendliness) of CTCP with SACK, FAST TCP with SACK, and Medley TCP with SACK are presented in Section 5.5.

5.2 Performance and Queue Length

In this section, the basic behavior, convergence time, and adaptation time of SACK, CTCP, FAST TCP, and Medley TCP are discussed. Convergence time indicates how many BaseRTTs are required to reach a new stable state or period. Adaptation time means when the available bandwidth is changed, like halved or doubled, how many round-trip times are needed by a TCP source to adjust its congestion window size.

Figure 5.1 shows the network topology of one single link used in subsections 5.2.1 and 5.2.2, where S1 represents a sender host using SACK, CTCP, FAST TCP, or Medley TCP. The slow start thresholds of SACK and CTCP are 64 KB[3]. The type of service used here is FTP. The receiver sends an ACK for every data packet received. For the convenience of presentation, we assume that all window sizes are measured in number of fixed-size packets, which are 1000 bytes. R1 and R2 represent two finite-buffer gateways. For the constant-load experiment, drop-tail gateways with FIFO service are assumed. The bandwidth of access links are 1Gbps, and propagation delays are 1ms. The bandwidth of connection link is X-Mbps and propagation delay is Y-ms. By varying X and Y, this communication network can represent networks with different characteristics such as small-bandwidth, large-bandwidth, short-delay,

[2]The FAST TCP module for *ns2* has been developed by CUBIN lab at the University of Melbourne's Centre. http://www.cubinlab.ee.unimelb.edu.au/ns2fasttcp/

[3]This value is commonly used in current operating systems' TCP algorithms.

and long-delay networks. The simulation time for each experiment is 600 seconds.

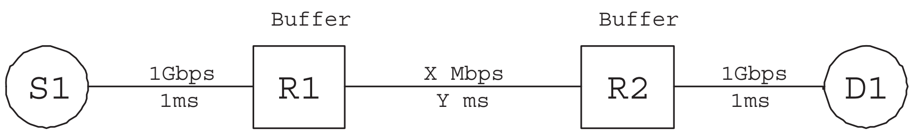


Figure 5.1: Network configuration for the simulations in subsections 5.2.1 and 5.2.2.

5.2.1 Basic Behavior

Four TCP versions' basic behaviors such as throughput and queue length requirement are presented in the following paragraphs when the router has either large or small buffer size in the bottleneck. The bottleneck capacity is set at 50 Mbps and link delay is 48 ms. A TCP connection starts sending data at 0 second. The objective of the simulation scenario is to explore how fast for a new connection to ramp up to equilibrium and how many packets in average are queued in the bottleneck.

(1) Large Buffer Size

Figures 5.2 and 5.3 exhibit the throughput and queue occupancy of SACK, CTCP, FAST TCP, and Medley TCP respectively when the buffer size of the bottleneck is set to 250 packets.

By observing the throughput evolution shown in Fig. 5.2, we can find that Medley TCP can achieve higher throughput than the others and also can stabilize at 50 Mbps until the simulation ends. SACK spends a lot of time to reach the available bandwidth because its slow start threshold value is too small (compared with BDP) and the increase rate in its congestion avoidance phase is very slow. The time for other three TCP versions to ramp up to equilibrium are very short. Although FAST TCP uses the shortest time among four TCP versions, it needs a constant buffer size (about 100 packets) to save its packets, as presented in Fig. 5.3. Comparatively, at most five packets sent from Medley TCP are queued in the bottleneck. As for SACK and CTCP, because they are loss-based algorithms, they increase their congestion

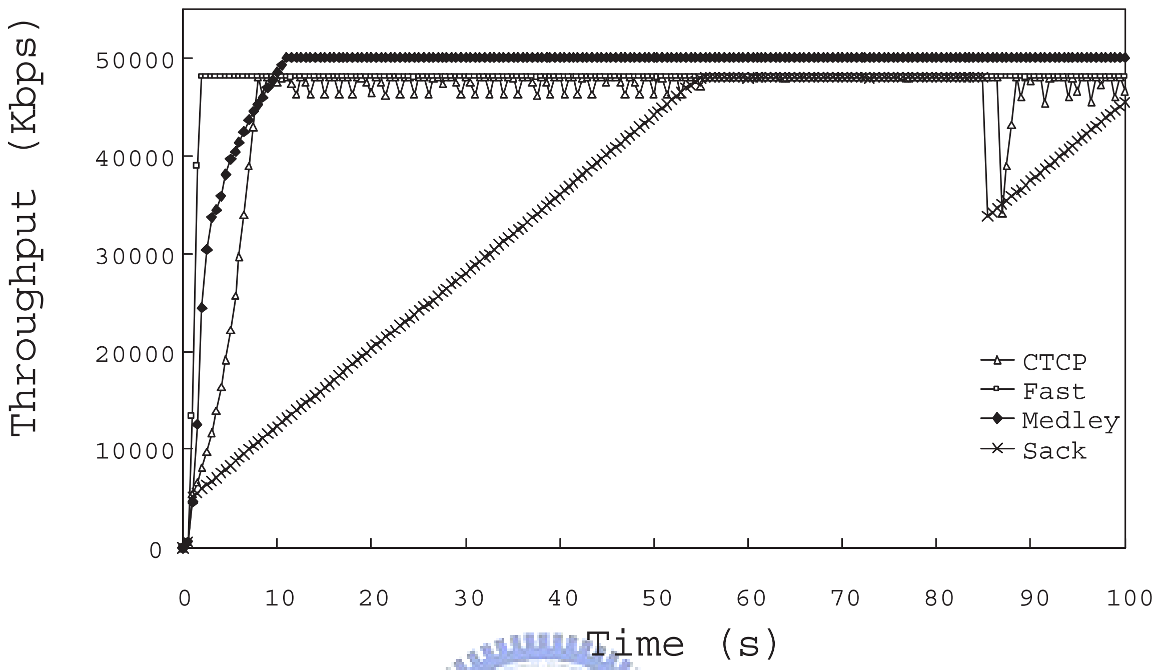


Figure 5.2: The throughput of SACK, CTCP, FAST TCP, and Medley TCP.

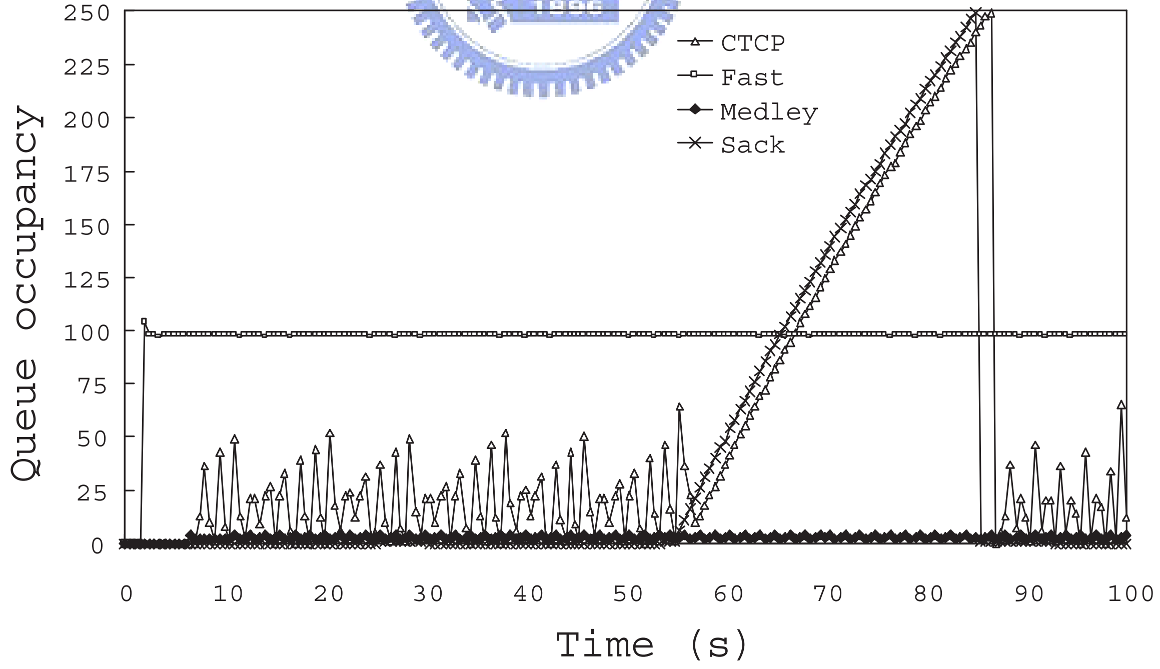


Figure 5.3: The queue occupancy made by four TCP versions.

window sizes until the router drops a packet.

(2) Small Buffer Size

In this simulation, we want to know whether the small buffer size in the router influences TCP's throughput, so the buffer size is set to 50 packets[4]. From Fig. 5.4, we can find that Medley TCP still achieves higher throughput than the others and also stabilizes at 50 Mbps until the simulation ends because Medley TCP just needs a small buffer size. In other words, when the router's buffer size is larger than Medley TCP's buffer size requirement, Medley TCP will achieves and stabilize at the available bandwidth very quickly. Moreover, SACK's behavior in Fig. 5.4 is similar to that in Fig. 5.2. The difference between two behaviors is that SACK suffers the packet loss here earlier than the above simulation. However, the behaviors of FAST TCP and CTCP in Fig. 5.4 are quite different from those in Fig. 5.2. The buffer size is a half of FAST TCP's requirement, so many Fast TCP's packets are dropped

[4]This value is a half of FAST TCP's need.

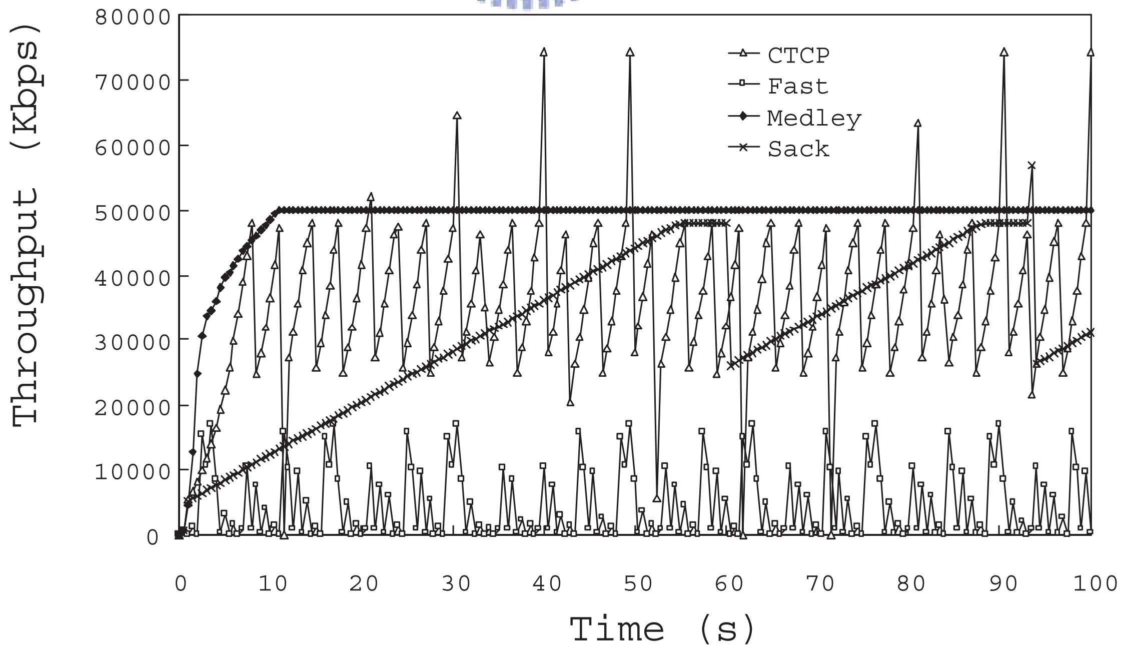


Figure 5.4: The throughput of SACK, CTCP, FAST TCP, and Medley TCP.

by the router. Accordingly, FAST TCP can not reach the available bandwidth and does not stay in a steady state. Because the delay-based component of CTCP increases by too many packets, the router can not handle these packets within a short time so that it drops part of them. Therefore, CTCP frequently changes its congestion avoidance phase to fast retransmit and fast recovery phase, and vice versa. Consequently, the shape of CTCP's throughput curve is like a sawtooth.

5.2.2 Convergence Time

With high BDP networks, the transient period of TCP can greatly affect overall performance. Here, we use convergence time to capture the transient performance of TCP. The range of BDP values for this experiment is from 150 Kb to 5000 Kb. In the following paragraphs, we show the convergence time, average throughput, and average queue occupancy of four TCP versions when the router in the bottleneck has large and small buffer size, respectively.

(1) Large Buffer Size

Figures 5.5, 5.6, and 5.7 present the convergence time, average throughput, and average queue length for a new connection of SACK, CTCP, FAST TCP, and Medley TCP to reach equilibrium respectively when the buffer size of the bottleneck is set to 250 packets.

Theoretically, SACK increases its congestion window size linearly after entering the congestion avoidance phase. So, a new SACK connection spends the longest time to ramp up to the available bandwidth when the BDP is large. In order to make a constant number of packets be queued in the bottleneck router (as shown in Fig. 5.7), FAST TCP sends more packets than other TCP versions do no matter how large buffer size in the bottleneck is, so that it uses the shortest time to reach equilibrium. The convergence times of Medley TCP and CTCP increase linearly because their algorithms adjust the congestion window sizes based on RTT variations. Moreover, the convergence time of Medley TCP is less than a fourth of that of SACK as the

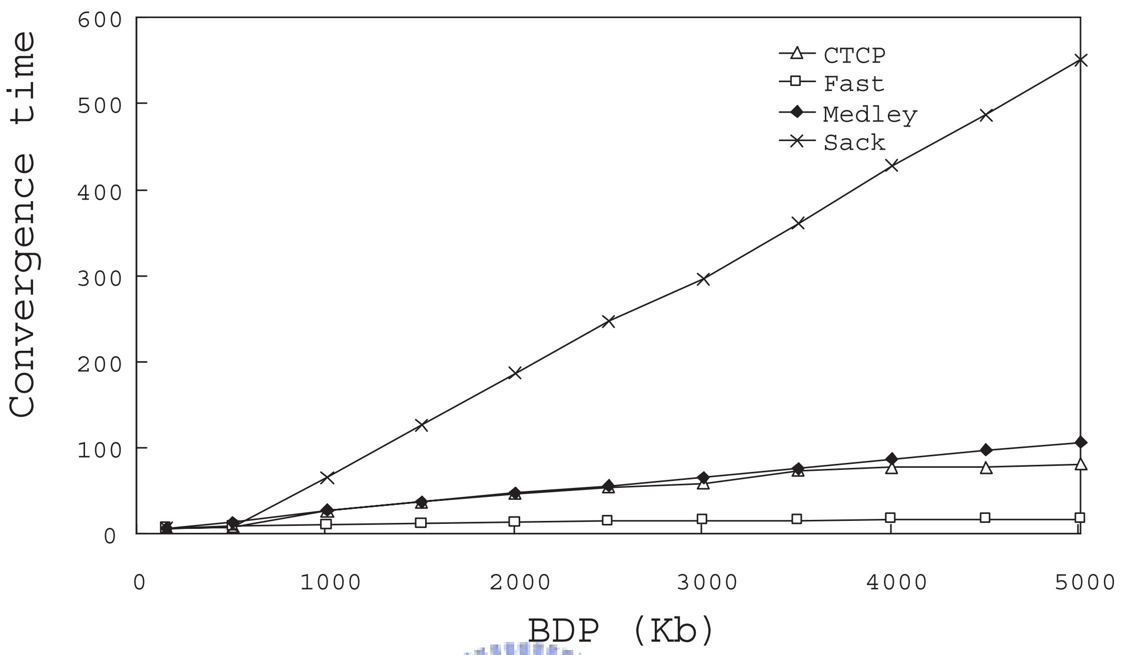


Figure 5.5: Convergence time of new connections with four TCP versions.

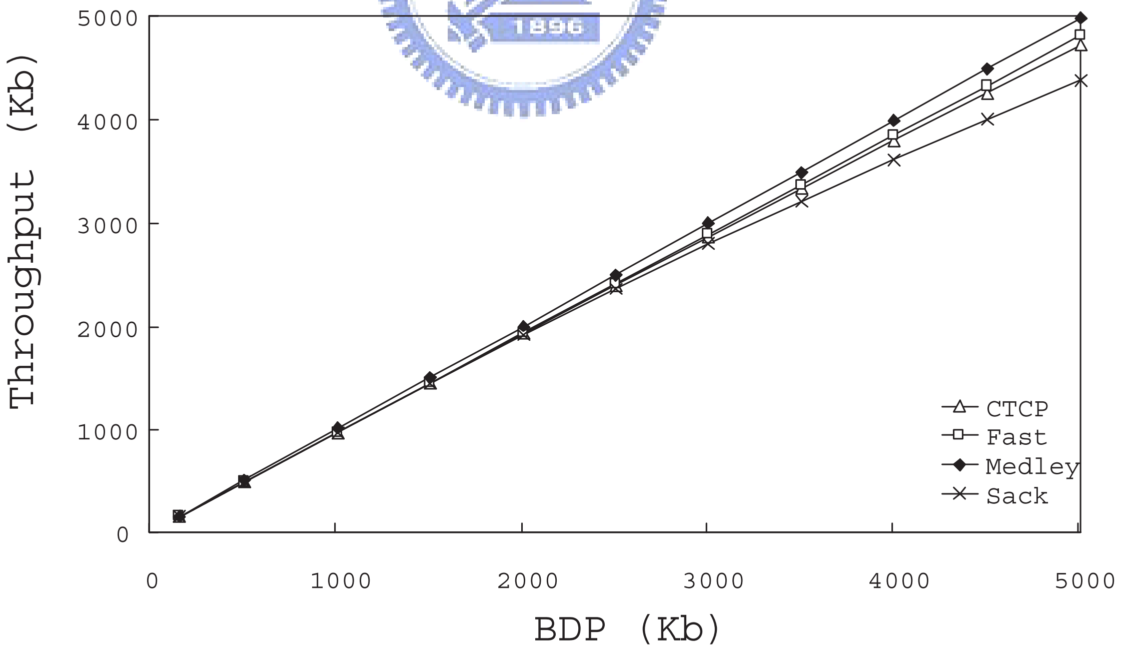


Figure 5.6: The average throughput of SACK, CTCP, FAST TCP, and Medley TCP.

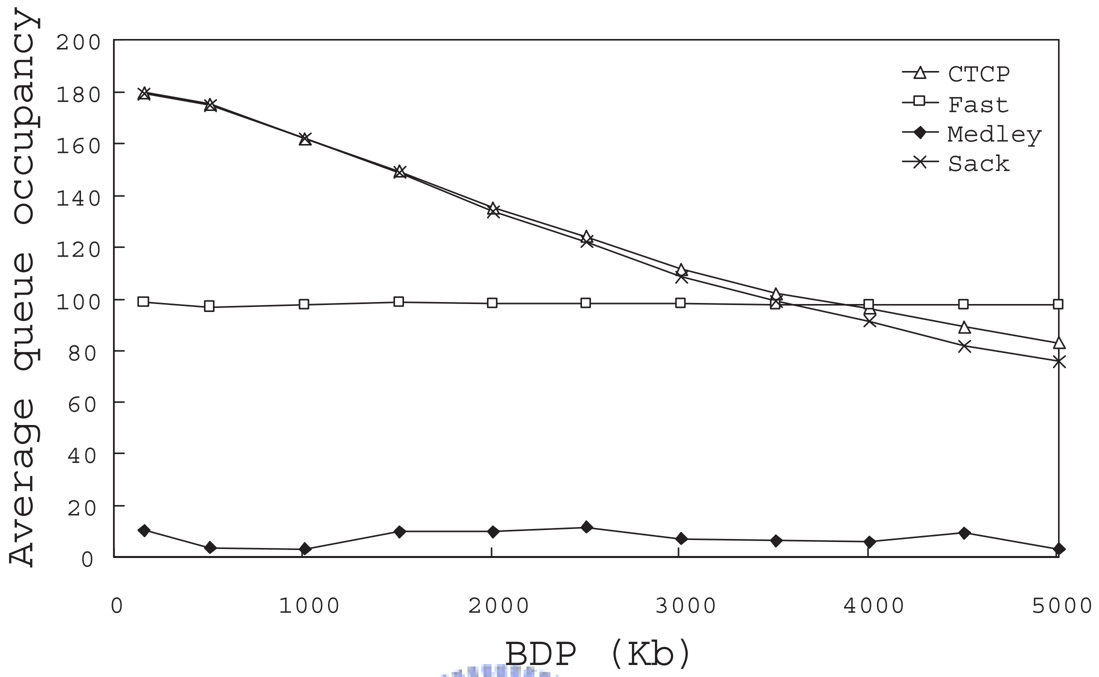


Figure 5.7: The average queue occupancy made by four TCP versions.

BDP is greater than 2 Mb.

Compared with CTCP and FAST TCP, the speed for a new Medley TCP flow climbing to the available bandwidth may be a little slow. However, Medley TCP has the highest average throughput in all TCP versions as shown in Fig. 5.6. Also, from Fig. 5.7, we can find that, no matter how high BDP is, the average queue occupancy made by Medley is no larger than 12 packets which is at most a ninth, a tenth, and an eighth of that of SACK, CTCP, and FAST TCP, respectively.

(2) Small Buffer Size

In this subsection, we only observe the performance of SACK, CTCP, and Medley TCP. This is because when the router's buffer size is smaller than FAST TCP's requirement, the performance of FAST TCP is very low as demonstrated in section 5.2.1. Here, the buffer size is changed to 50 packets, and other parameters are same as those in the above experiment.

Similar to Fig. 5.5, in Fig. 5.8, a new SACK connection spends the longest

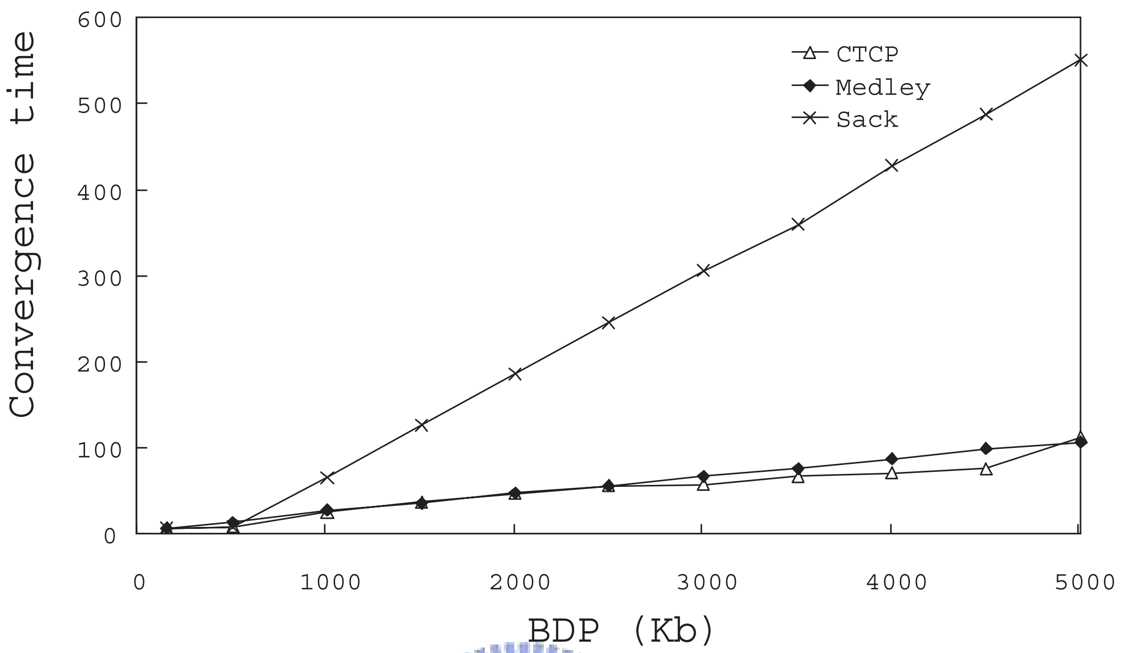


Figure 5.8: Convergence time of new connections with three TCP versions.

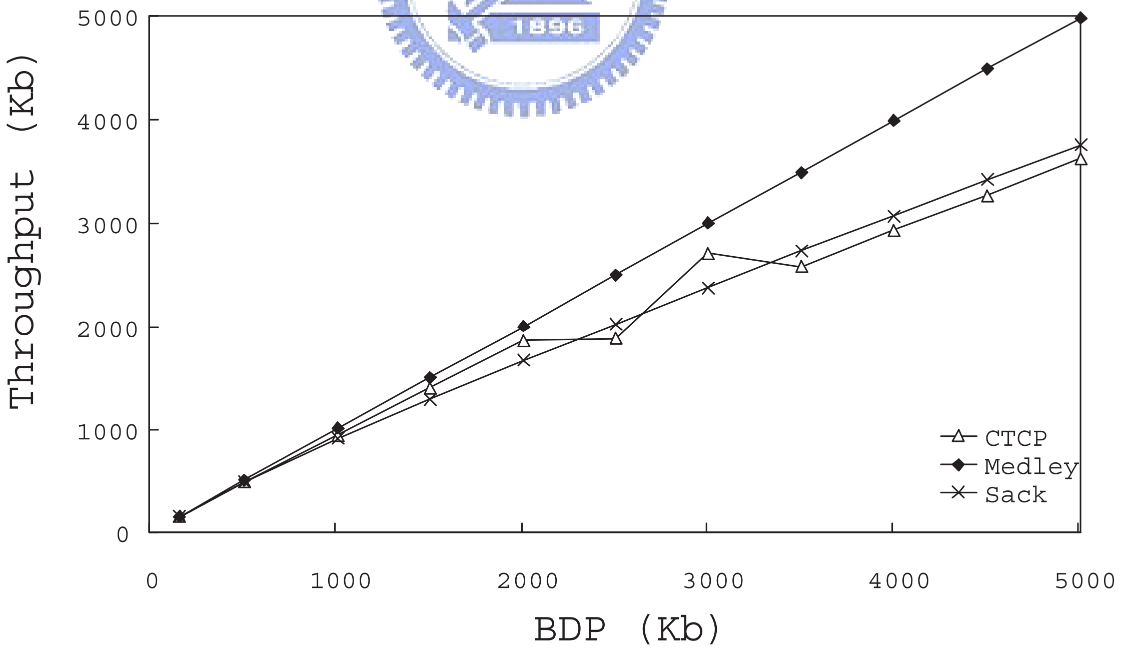


Figure 5.9: The average throughput of SACK, CTCP, and Medley TCP.

time to ramp up to equilibrium when the BDP is large and the convergence times of Medley TCP and CTCP increase linearly. However, from Fig. 5.9, the average throughput of SACK is higher than that of CTCP when the BDP is greater than 3.5 Mb. The reason may be that CTCP's delay component makes an error decision and increases more packets than the rest of router's buffer size. Accordingly, the whole performance of CTCP is low although the recovery speed is fast. Furthermore, the average throughput of Medley TCP is the highest among three TCP versions. For example, Medley TCP achieves a 1.32 and a 1.37 times higher average throughput in comparison with that of SACK and CTPC, respectively.

By observing Figures 5.7 and 5.10, the range of Medley TCP's packets averagely queued in the bottleneck is from 3 to 12 packets no matter how large BDP and buffer size are[5]. The average queue occupancies made by SACK and CTCP are getting smaller when the BDP is getting higher. Moreover, when the bottleneck router's buffer size is getting smaller, this situation is getting clearer. This is because when

[5]Maybe saying that the buffer size in the bottleneck must be larger than 12 packets is correct.

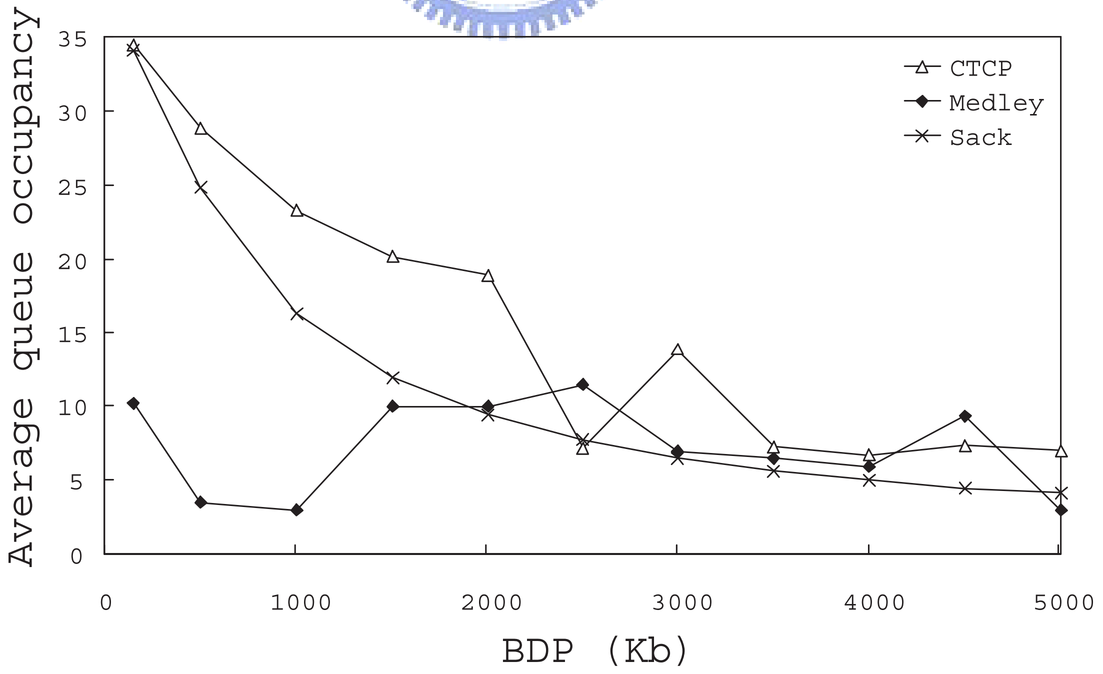


Figure 5.10: The average queue occupancy made by these three TCP versions.

the BDP becomes higher, SACK and CTCP need more time to make packet loss occur. On the other hand, packet dropping in the bottleneck router will be more likely to happen if the buffer size becomes smaller.

From simulation results of subsections 5.2.1 and 5.2.2, we think Medley TCP can provide a good service for real-time applications such as Internet Protocol Television (IPTV), Voice over Internet Protocol (VoIP), and so on. There are three reasons. First, Medley TCP's throughput is very stable that supplies a fixed rate to real-time applications. Second, the range of queue occupancy made by Medley TCP is very small (from 3 to 12 packets). This could make the delay between a packet and its next packet be expected and estimated. Finally, Medley TCP works well no matter how small buffer size is and how high BDP is.

5.2.3 Adaptation Time

In order to change the available bandwidth, we add a constant bit rate (CBR) traffic flow in this simulation and other traffic sources are same as that in the previous subsections. A CBR traffic flow starts sending data with a half of bottleneck bandwidth rate at the 200^{th} second and stops at the 400^{th} second. For example, if the bottleneck bandwidth is 50 Mbps, the rate of CBR traffic flow is 25 Mbps. When this CBR traffic starts, the available bandwidth is halved. Similarly, the bottleneck bandwidth is doubled as the CBR traffic stops sending data.

Figures 5.11 and 5.12 show the adaptation time as the available bandwidth is halved and doubled, respectively. From Fig. 5.11, we can see that the time for SACK adjusting its window size to the new available bandwidth is the longest. Other TCP versions can change their throughput to suit the network in a short time. By observing Fig. 5.12, we can find that the adaptation time of Medley TCP is the shortest no matter how high BDP is when the available bandwidth is doubled. Moreover, Medley TCP's adaptation time increases linearly while the BDP becomes large. In summary, Medley TCP improves the transient performance of connection in both scenarios as compared to SACK, CTCP, and FAST TCP.

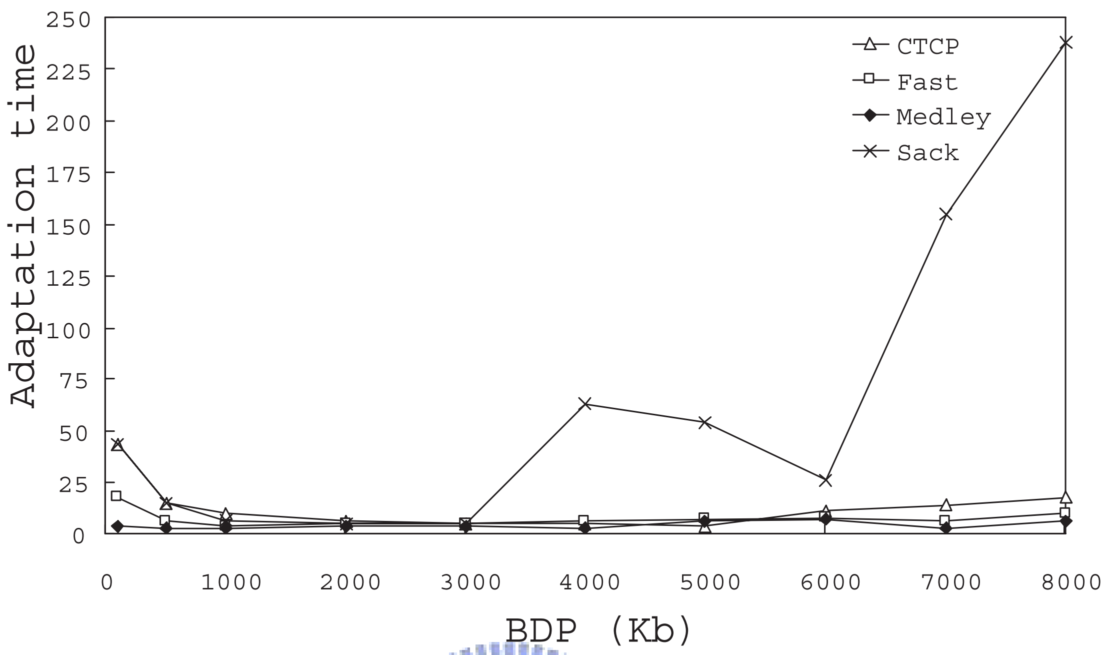


Figure 5.11: Adaption time of connections when available bandwidth is halved.

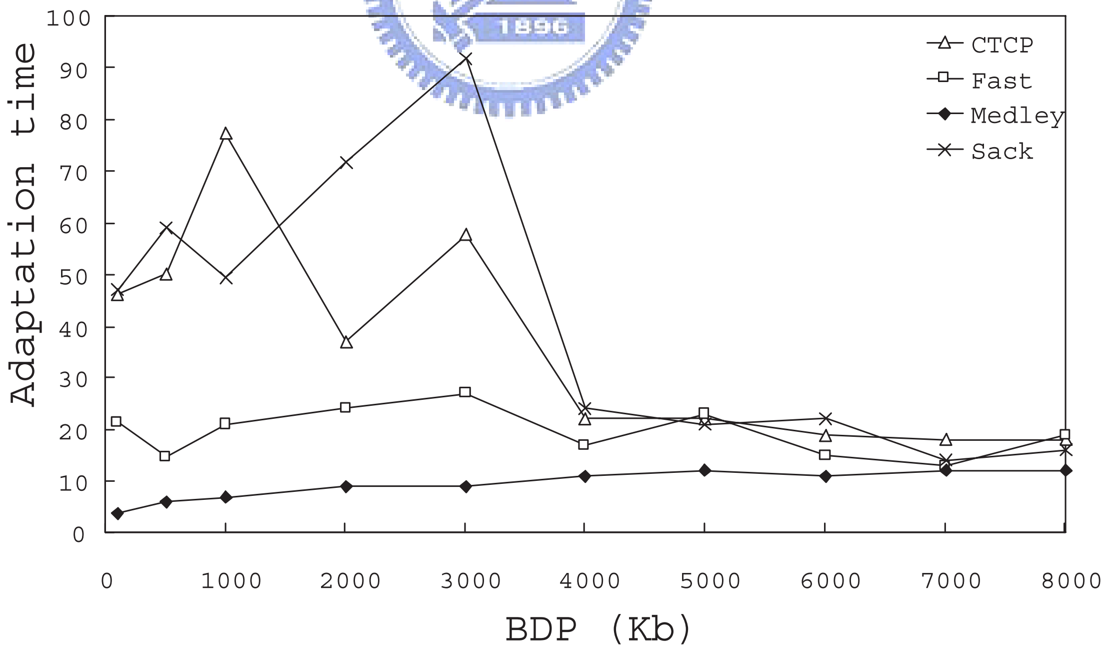


Figure 5.12: Adaption time of four TCPs when available bandwidth is doubled.

5.3 Utilization and Fairness

Utilization of bottleneck bandwidth and fairness are important issues of transport protocol. The simulations presented in this section intend to demonstrate link utilization of bottleneck and fairness between the connections when sources use same TCP version. There are two cases for discussion. One is that all users have same RTT, and sources with different RTTs is the other. Note that the latter happens frequently in real Internet, and parameters of the former can be controlled easily in the simulation. In Figures 5.13 and 5.14, parameter settings are as follows. The simulation time is 900 seconds. All sources start sending data at the same time. The link capacity and propagation delay between two routers are 100 Mbps and 23 ms, respectively. The link capacity and propagation delay between end host and router are 1 Gbps and 1 ms, respectively. The buffer size of each router is k times the number of connections. For example, if k and the number of connections equal to 50 and 20 respectively, the buffer size in each router will be 1000 packets.

5.3.1 Same RTT

The first network topology for the simulation is shown in Fig. 5.13, in which all traffic pairs features the same propagation delay. There are two, five, ten, fifteen, and twenty connections using same TCP version from S_1 to S_n in each simulation.

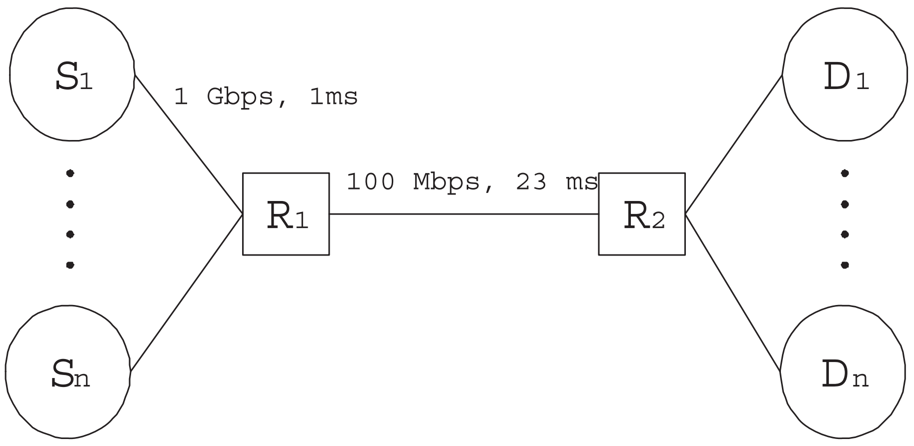


Figure 5.13: The network topology for sources with same RTT.

(1) Large Buffer Size

Here, we let k equal 200 so that the buffer sizes of the router are 400, 1000, 2000, 3000, and 4000 packets when the numbers of connections are 2, 5, 10, 15, and 20, respectively. Note that in the ideal case, each connection can have 200 packets queued in the router. The utilization and fairness index of each simulation are presented in Table 5.1 and Table 5.2, respectively.

From these two Tables, we can find that the utilizations of bandwidth in all simulations are higher than 90% and those of Medley TCP are the highest and almost close to 100%. Furthermore, the fairness indexes of Medley TCP are just

Table 5.1: Utilization of bandwidth with SACK, CTCP, FAST, and Medley TCP for large buffer size.

No. of flows	SACK	CTCP	FAST	Medley
2	94.72%	95.65%	96.11%	99.92%
5	96.08%	96.08%	96.12%	99.95%
10	96.10%	96.10%	96.12%	99.96%
15	96.11%	96.11%	96.12%	99.97%
20	96.10%	96.10%	96.13%	99.97%

Table 5.2: Fairness index for SACK, CTCP, FAST TCP, and Medley TCP for large buffer size.

No. of flows	SACK	CTCP	FAST	Medley
2	99.99%	99.99%	99.99%	92.47%
5	99.99%	99.99%	99.99%	99.72%
10	99.99%	99.99%	99.99%	98.79%
15	99.95%	99.96%	99.99%	99.30%
20	99.92%	99.94%	99.99%	99.22%

little lower than those of other TCP versions. For instant, when there are two connections in the network, the fairness indexes of SACK, CTCP, FAST TCP, and Medley TCP are 99.99%, 99.99%, 99.99%, and 92.47%, respectively. The simulation results suggest that Medley TCP provides the better bandwidth utilization than the others do and keeps the good characteristic of fairness as those of other TCP algorithms when the router's buffer size is large.

(2) Small Buffer Size

In order to investigate whether small buffer size in the router influences TCP's fairness, we set k to 50 in the following simulations. Other parameters are same as those in previous experiments. Table 5.3 and 5.4 show the utilization and fairness index of each simulation, respectively.

By observing Table 5.3, we can see that when the buffer size is small, the utilizations of bottleneck bandwidth of SACK, CTCP, and FAST TCP are affected. Particularly, FAST TCP's performance is very poor, and falls down to less than 30% of available bandwidth, because it can not keep a constant queue occupancy in the router's buffer. On the contrary, the utilization of bandwidth and the fairness index of Medley TCP are still maintained as high as those in previous simulations.

Table 5.3: Utilization of bandwidth with SACK, CTCP, FAST, and Medley TCP for small buffer size.

No. of flows	SACK	CTCP	FAST	Medley
2	78.11%	76.53%	23.96%	99.92%
5	85.51%	85.69%	34.65%	99.95%
10	92.26%	95.03%	32.86%	99.96%
15	95.14%	95.86%	37.47%	99.97%
20	96.10%	96.10%	78.47%	99.97%

Table 5.4: Fairness index for SACK, CTCP, FAST TCP, and Medley TCP for small buffer size.

No. of flows	SACK	CTCP	FAST	Medley
2	99.99%	99.92%	99.95%	92.47%
5	99.99%	99.99%	99.84%	99.72%
10	99.99%	99.94%	99.98%	98.79%
15	99.99%	99.84%	99.99%	99.30%
20	99.99%	99.93%	97.83%	99.22%

5.3.2 Different RTTs

The case of sources having different RTTs happens commonly in the real Internet. Therefore, in following simulations, we evaluate the sources with different RTTs, and the network topology is shown in Fig. 5.14. The bandwidth and propagation delay are 1 Gbps and 1 ms, and 100 Mbps and 23 ms for the full-duplex access link and the full-duplex trunk link, respectively. The sources are SACK, CTCP, FAST TCP, and Medley TCP. There are two, three, four, and five connections from S_1 to S_n in each simulation.

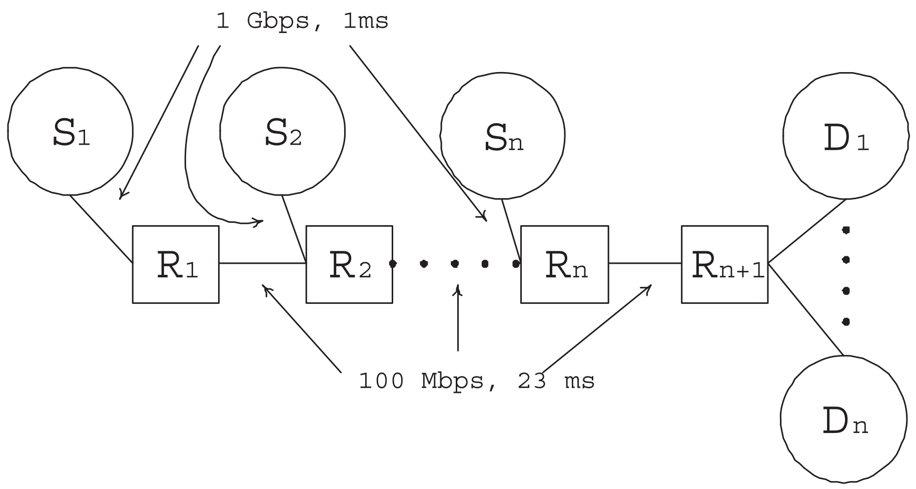


Figure 5.14: The network topology for sources with different RTTs.

(1) Large Buffer Size

Similar to above experiments, we first discuss a case of routers having large buffer size. K equals 200 so that the buffer sizes of the router are 400, 600, 800, and 1000 packets when there are 2, 3, 4, and 5 connections, respectively. Note that in the ideal case, each connection can have 200 packets queued in the bottleneck router. The utilization and fairness index of each simulation are depicted in Table 5.5 and 5.6, respectively.

From Table 5.5, we observe that Medley TCP's utilization of bottleneck bandwidth is the highest among all TCP variants and the value is very close to 100% although other TCPs' utilizations of bandwidth are over 90%. Table 5.6 shows that the fairness index of Medley TCP is much higher than those of SACK and CTCP

Table 5.5: Utilization of bottleneck bandwidth with same TCP version for large buffer size.

No. of flows	SACK	CTCP	FAST	Medley
2	94.34%	95.71%	96.09%	99.83%
3	95.97%	95.98%	96.09%	99.86%
4	95.91%	95.99%	96.09%	99.86%
5	96.02%	96.01%	96.09%	99.87%

Table 5.6: Fairness index for SACK, CTCP, FAST TCP, and Medley TCP for large buffer size.

No. of flows	SACK	CTCP	FAST	Medley
2	73.18%	74.23%	99.99%	95.20%
3	60.57%	59.92%	99.98%	98.55%
4	89.44%	90.75%	99.98%	96.00%
5	81.07%	83.64%	99.99%	97.35%

because the main cores of SACK and CTCP are loss-based algorithms, which are demonstrated and proved that they do not have a good fairness when the sources have different RTTs. Also, the difference between Medley TCP's and FAST TCP's fairness index is very small. This is because Medley TCP preserves the characteristic and advantage of delay-based mechanisms. For example, when there are three flows in the network, the differences between SACK's and Medley TCP's, between CTCP's and Medley TCP's, and between FAST TCP's and Medley TCP's fairness index are 37.98%, 38.63%, and 1.43% respectively.

(2) Small Buffer Size

We set k to 50 in the following simulations in order to investigate whether small buffer size in the router influences TCP's fairness. Other parameters are same as those of previous experiments. Table 5.7 presents the utilization of bottleneck bandwidth and Table 5.8 shows fairness index of each TCP mechanism.

By observing Table 5.7, we can find that when the buffer size is small, the utilizations of bottleneck bandwidth of SACK, CTCP, and FAST TCP are decreased. Particularly, FAST TCP's performance is very poor, which falls down to about forty percent of bottleneck bandwidth, because it is unable to keep a constant queue occupancy in the router's buffer. On the contrary, although the utilization of bottleneck bandwidth of Medley TCP is a little lower than the previous simulation with large buffer size, the utilization of bandwidth and the fairness index of Medley TCP are the highest among all TCP mechanisms. For instant, the utilization of bottleneck bandwidth of Medley is about 1.09 times of SACK, about 1.08 times of CTCP, and about 2.17 times of FAST TCP, and the fairness index of Medley TCP is about 1.36 times of SACK, more than 1.09 times of CTCP, and about 1.30 times of FAST TCP when five connections with different RTTs share the same bottleneck bandwidth, respectively.

Table 5.7: Utilization of bottleneck bandwidth with same TCP variant for small buffer size.

No. of flows	SACK	CTCP	FAST	Medley
2	89.94%	93.22%	23.81%	99.83%
3	92.18%	94.98%	35.15%	99.68%
4	90.52%	93.02%	40.92%	99.73%
5	91.06%	92.17%	45.88%	99.34%

Table 5.8: Fairness index for SACK, CTCP, FAST TCP, and Medley TCP for small buffer size.

No. of flows	SACK	CTCP	FAST	Medley
2	72.02%	64.52%	99.99%	95.20%
3	75.71%	74.85%	88.56%	95.92%
4	83.71%	93.36%	79.85%	95.23%
5	71.09%	88.37%	74.56%	96.56%

By observing simulations in Section 5.3, Medley TCP can provide a good RTT fairness whether the sources have same RTT or different RTTs regardless of number of connections or buffer size in the bottleneck router. Therefore, we think this characteristic of Medley TCP may be useful to provide good services for ordinary data transmission like File Transfer Protocol (FTP), and for real-time applications such as IPTV, VoIP, etc.

5.4 Impact of Random Loss

The wireless network configuration for the simulations is shown in Fig. 5.15, in which the bandwidth and delay of each full duplex link and a router's buffer size are

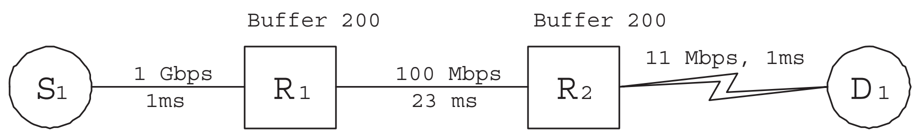


Figure 5.15: Wireless network configuration for the simulations.

depicted. S_1, D_1, and R_i express as a source, a destination, and routers respectively. A TCP connection is established from S_1 to D_1. 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 Medley TCP to recognize them. Because Medley TCP takes every packet loss as random loss when Δ is no larger than Γ. Therefore, we use a more complicated case, uniformly distributed loss model, to evaluate Medley TCP. In wireless environments, if the bit error rate is uniformly distributed, the larger a packet is, the more likely it will be corrupted [60, 61]. Keeping this in mind, when we apply a random loss rate to data packets, we always set the proportional random loss rate to ACKs.

We compare the average throughput among the four TCP variants with different random loss rates. By observing the results shown in Fig. 5.16, Medley TCP can fully utilize the bottleneck link when the random loss rate is zero. However, other TCPs can not maintain such high throughput with the same condition. This is because SACK and CTCP need to create packet losses by themselves to probe the available bandwidth along the path. So, certain amount of throughput will be lost.

With the increasing random loss rate, the throughput improvement of Medley TCP becomes obvious. When random loss rate is between 0.5% and 2%, the throughput of Medley TCP is about 2.76, 2.74, and 1.66 times higher than that of SACK, CTCP, and FAST TCP respectively. When the random loss rate is over 2%, Medley TCP always keeps a throughput improvement of larger than 79%, 77%, and 15% in comparison with SACK, CTCP, and FAST TCP. Notably, with slight random loss rate (from 3% to 5%), the throughput improvement is up to 116%, 115%, and

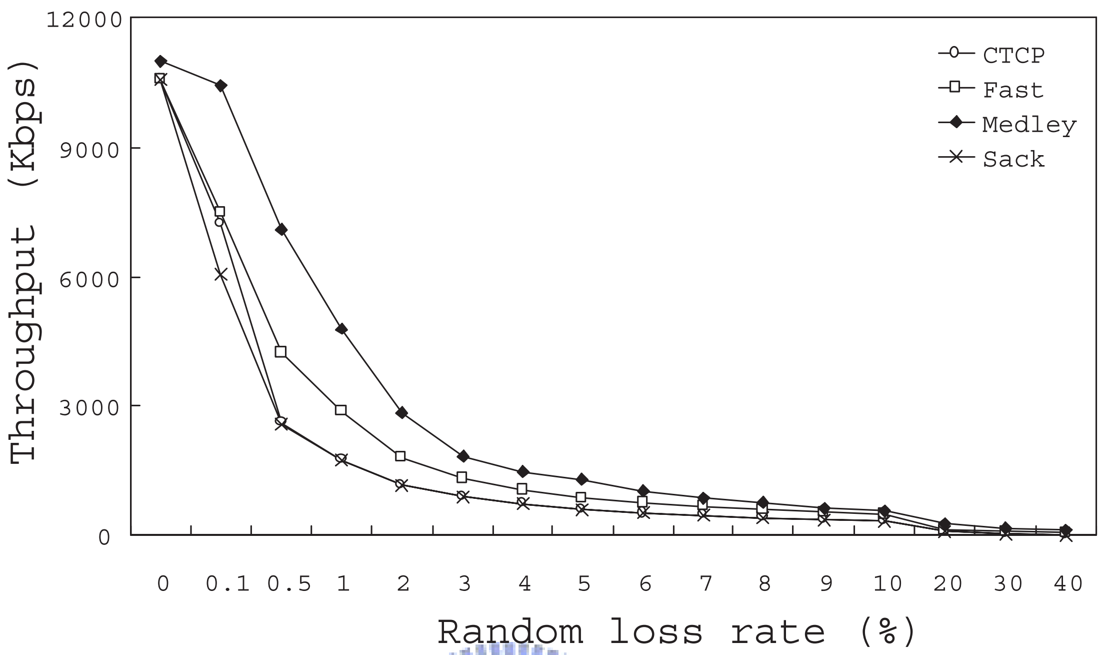


Figure 5.16: Average throughput versus random loss rate for four TCP variants.

49.7%. Moreover, the throughput improvements are 79.35%, 79.35%, and 15.07% as compared with SACK, CTCP, and FAST TCP respectively in a severe random loss rate (10%).

5.5 TCP Friendliness

For a newly proposed TCP algorithm, when it shares resources with current TCP mechanisms in the same network, it should be compatible with today's TCP algorithm. In the ideal situation, a new TCP will not cause any detrimental effects to the current TCP and vice versa. This is so-called TCP friendliness. Accordingly, in this section, we discuss the TCP friendliness of CTCP, FAST TCP, and Medley TCP when they compare with SACK which is currently the dominating TCP version deployed in the Internet.

The simulation topology is presented in Fig. 5.17. S_i, D_i, and R_i denote sources, destinations, and routers respectively. A source and a destination with the same

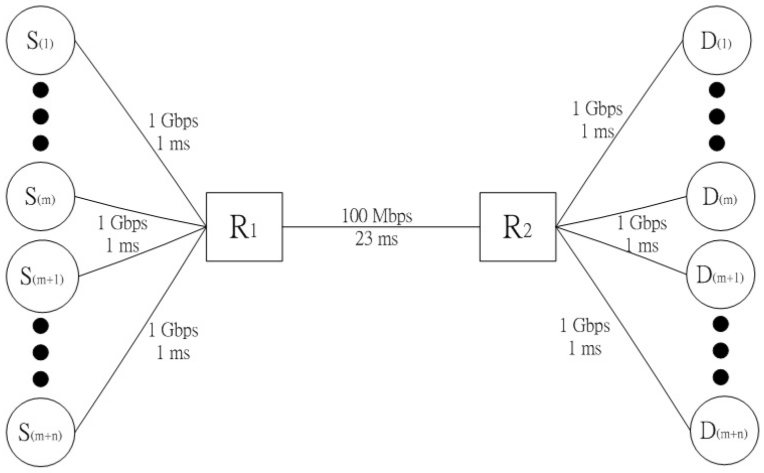
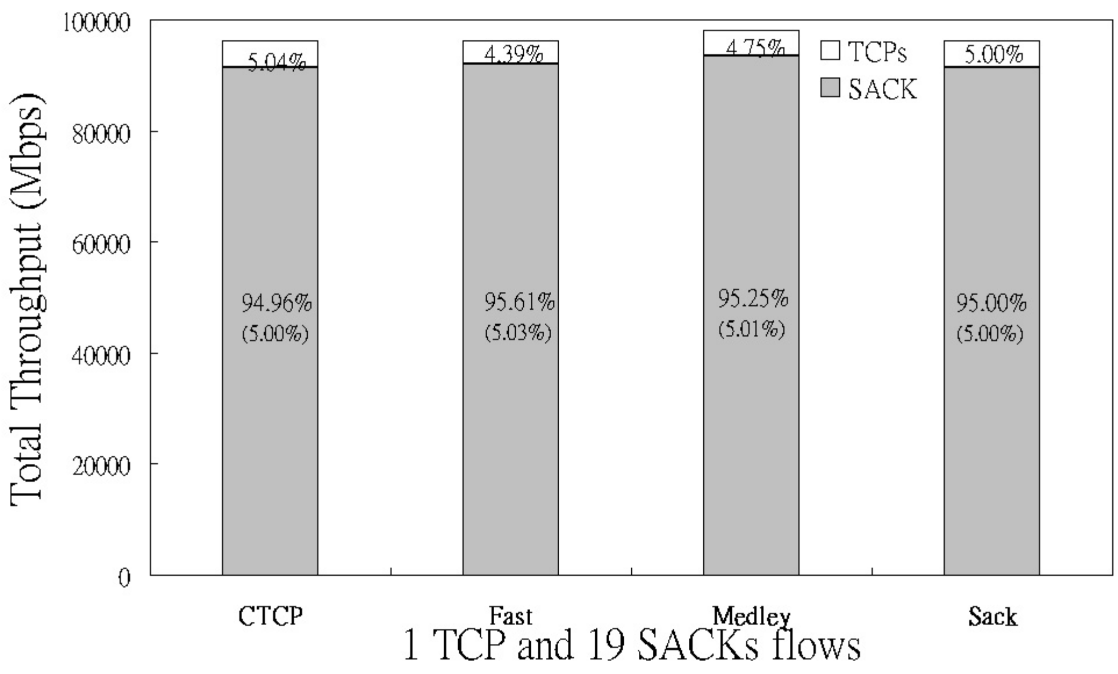


Figure 5.17: The network topology for evaluating TCP friendliness.

suffix value represent a traffic pair. M sources (from S_1 to S_m) use one TCP algorithm (i.e., CTCP, SACK, FAST TCP, or Medley TCP), and SACK is executed by other sources (from $S_{(m+1)}$ to $S_{(m+n)}$). Note that all connections using SACK is a control group in order to see whether other TCP variants cause any effects to SACK and vice versa. The bandwidth and propagation delay are 1 Gbps and 1 ms for the full-duplex access link, and 100 Mbps and 23 ms for the full-duplex trunk link respectively. The simulation results of 20 flows sharing same bottleneck bandwidth are shown in the following figures (from Fig. 5.18 to Fig. 5.25). In these figures, a word TCP presents CTCP, SACK, FAST TCP, or Medley TCP, and the results with 3000 and 200 packets for the buffer size of routers are shown in the top and bottom of each bar chart, respectively. The percentages of the sum of m connections' and $(20-m)$ flows' throughput are exhibited in the figures. Also, the average percentage of each flow is presented in brackets.

By observing simulation results, we can find that when FAST TCP and SACK coexist in the network, the performance of SACK is much higher than that of FAST TCP no matter how large buffer size is. This is because a loss-based TCP connection uses most of the buffer space in the bottleneck node when competing with a delay-based TCP connection. A delay-based TCP may be therefore not competitive with a loss-based TCP. This outcome is very close to the situation in the present Internet. From figures, we can see that the utilization of bottleneck bandwidth of CTCP and SACK connections is as high as that of all SACK flows. Moreover, the percentages of CTCP and SACK are alike. The reason is that CTCP is regarded as a scaled version of SACK, and therefore it is friendly to SACK when they share same bottleneck bandwidth. Comparatively, Medley TCP coexists peacefully with SACK without stealing bandwidth from it. Furthermore, the utilization of bottleneck bandwidth is also improved to about 99%. The improvement can be attributed to Medley TCP's efficient utilization of the available bandwidth. We may say Medley TCP provides a friendly congestion control mechanism to the current transport protocol.



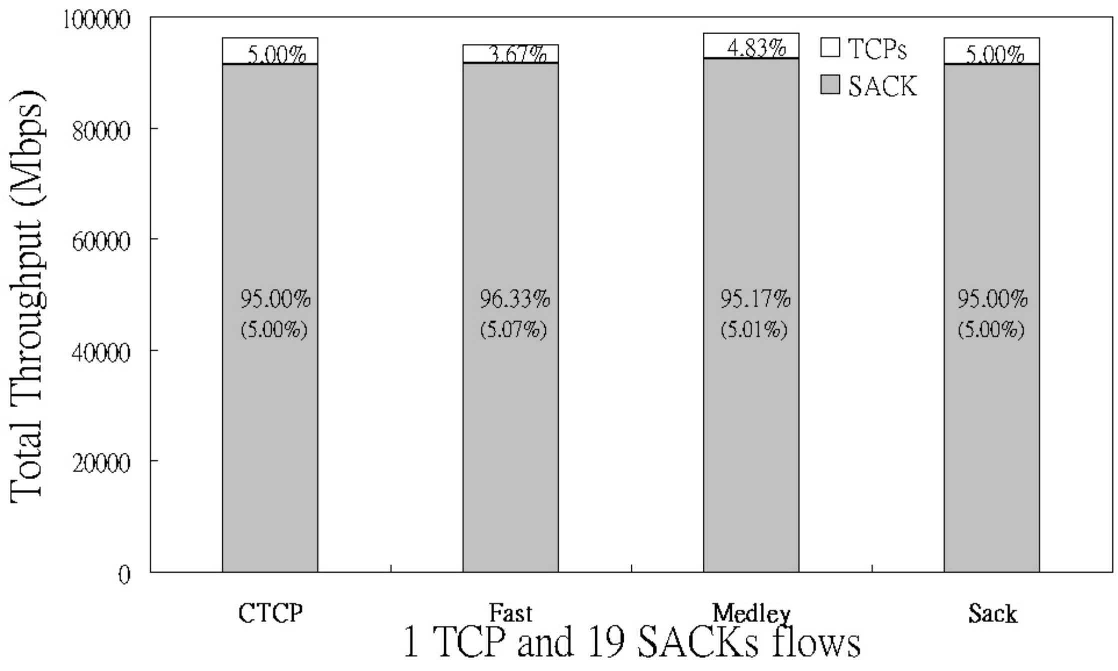
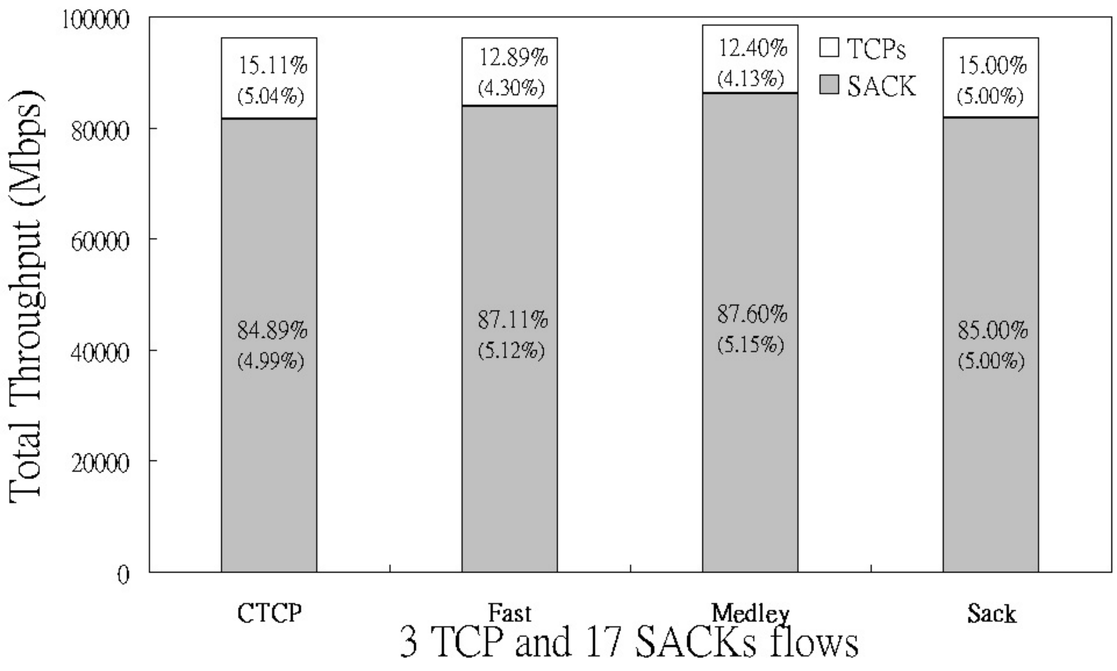


Figure 5.18: The percentage and throughput for one flow of one TCP variant vs. nineteen SACK flows.



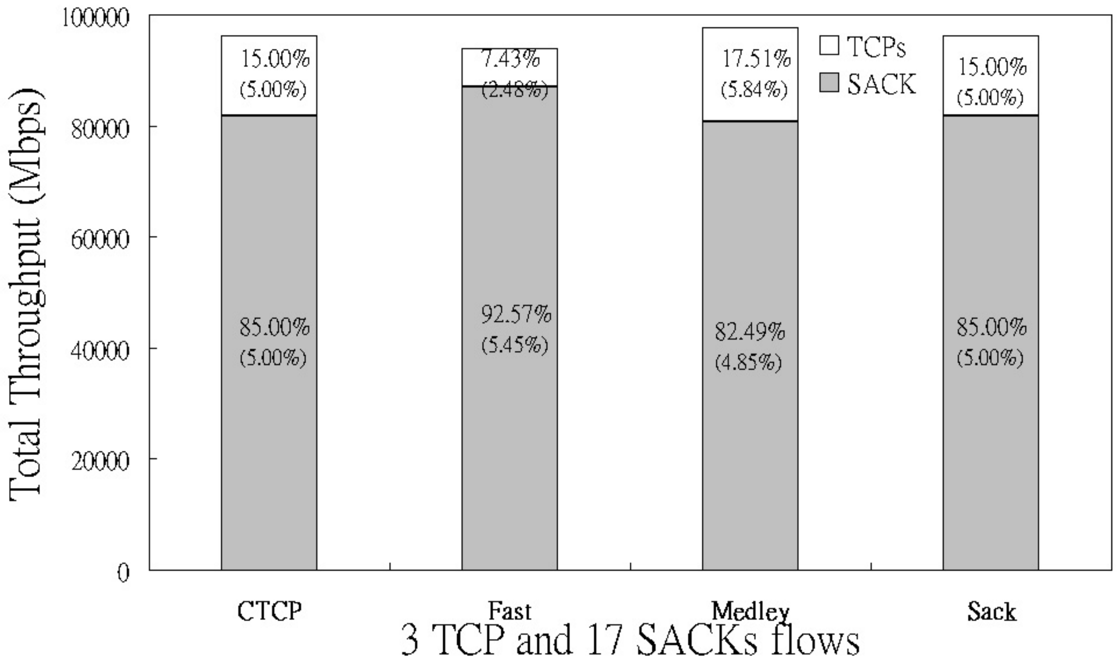
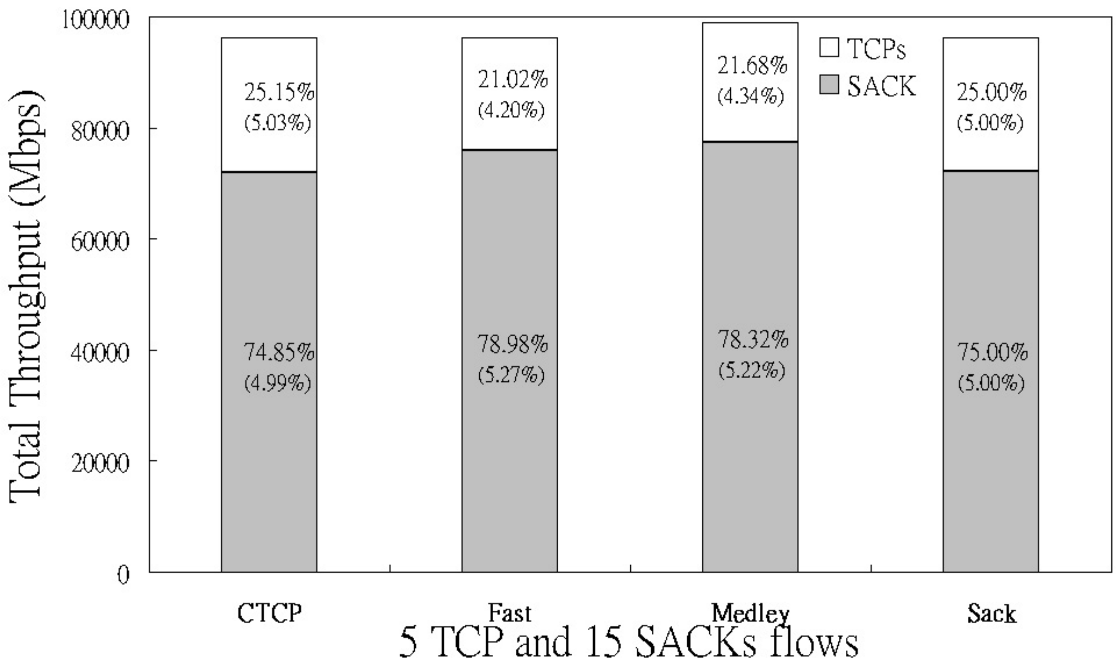


Figure 5.19: The percentage and throughput for three flows of one TCP variant vs. seventeen SACK flows.



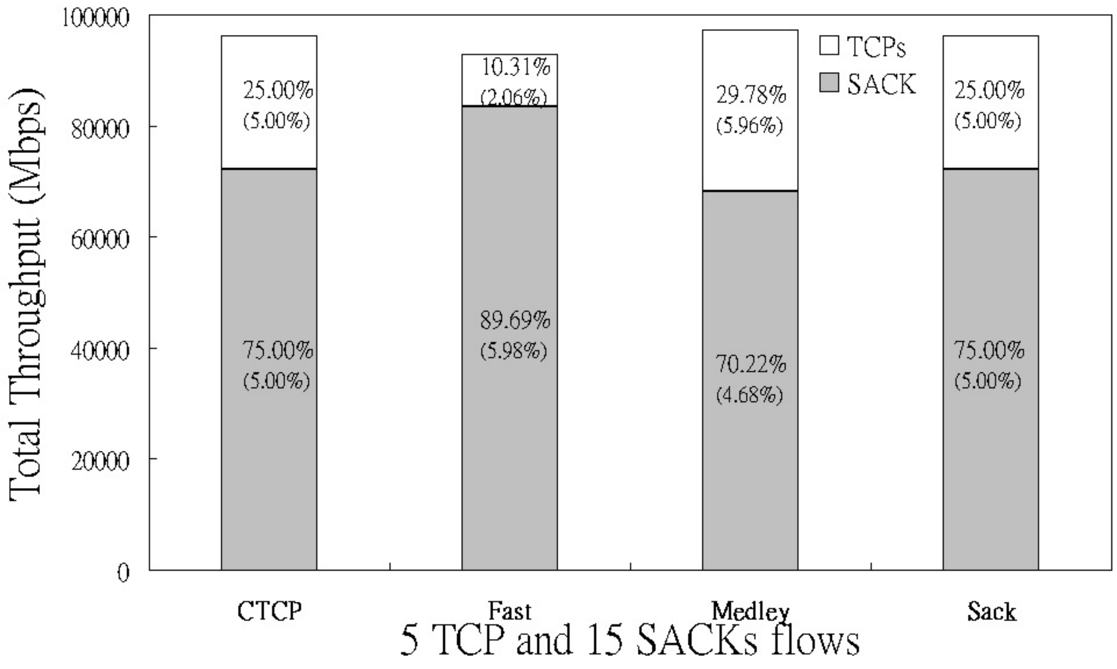
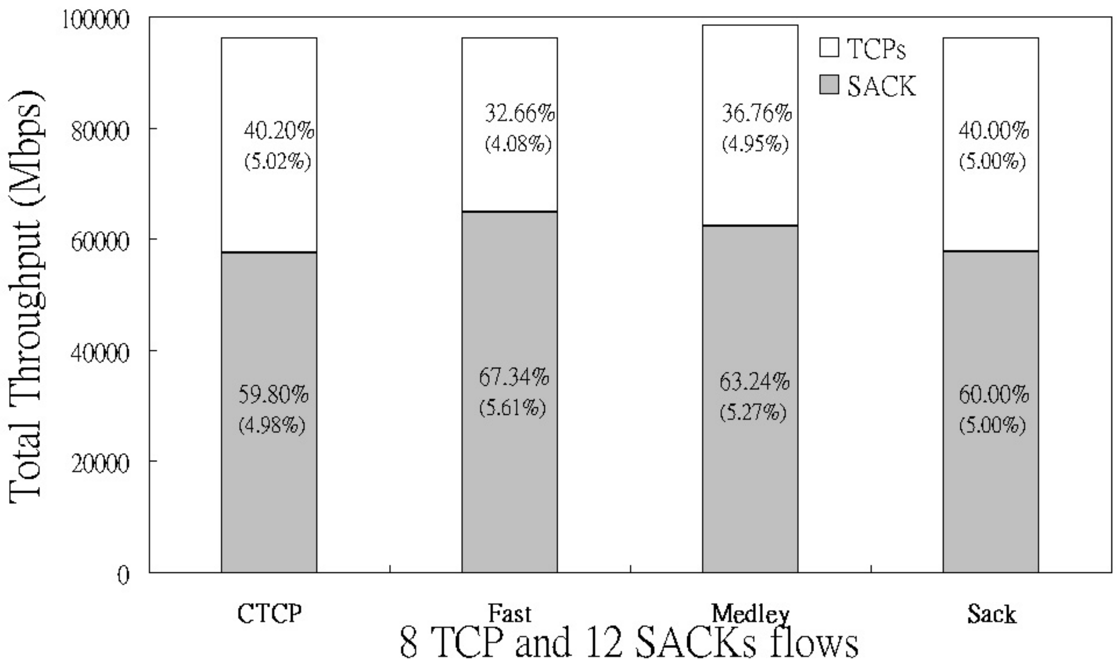


Figure 5.20: The percentage and throughput for five flows of one TCP variant vs. fifteen SACK flows.



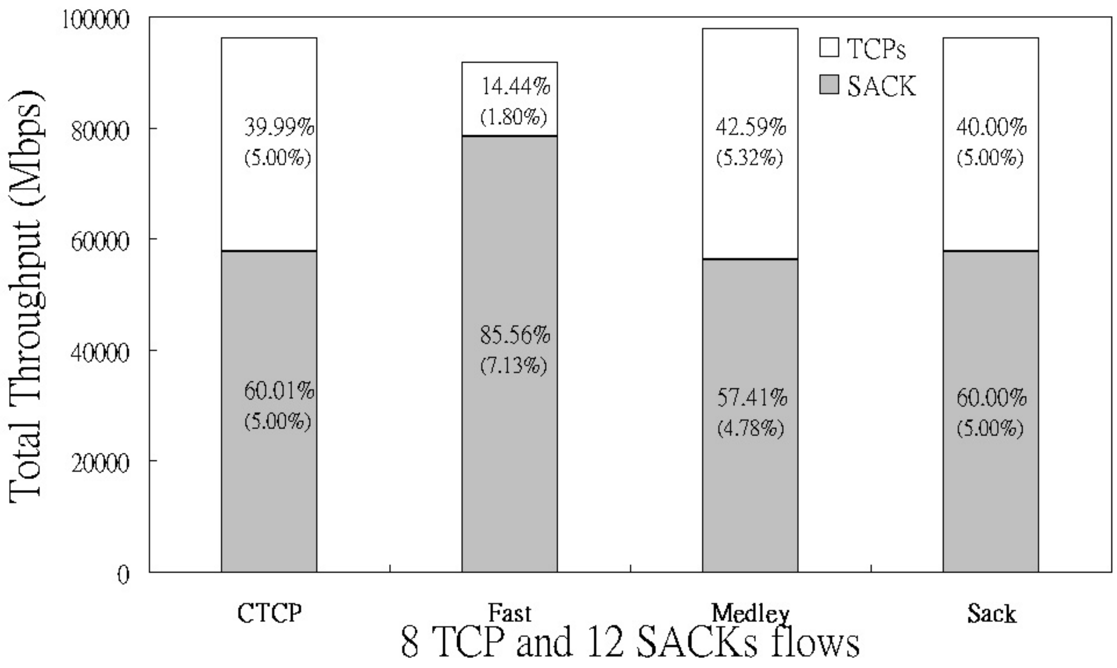
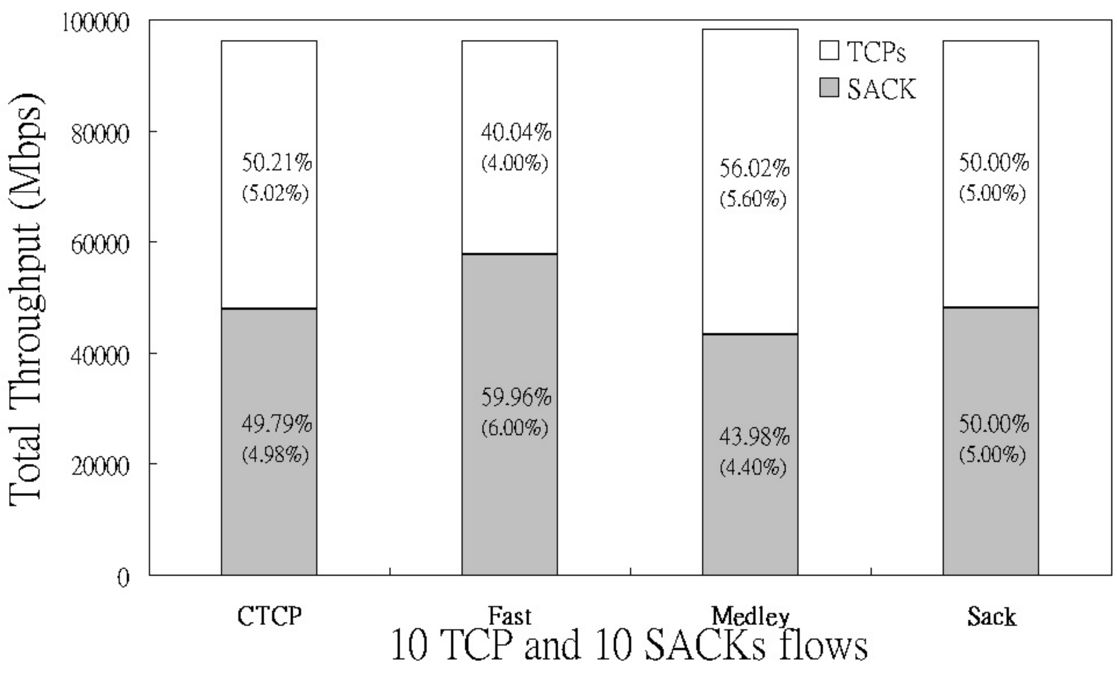


Figure 5.21: The percentage and throughput for eight flows of one TCP variant vs. twelve SACK flows.



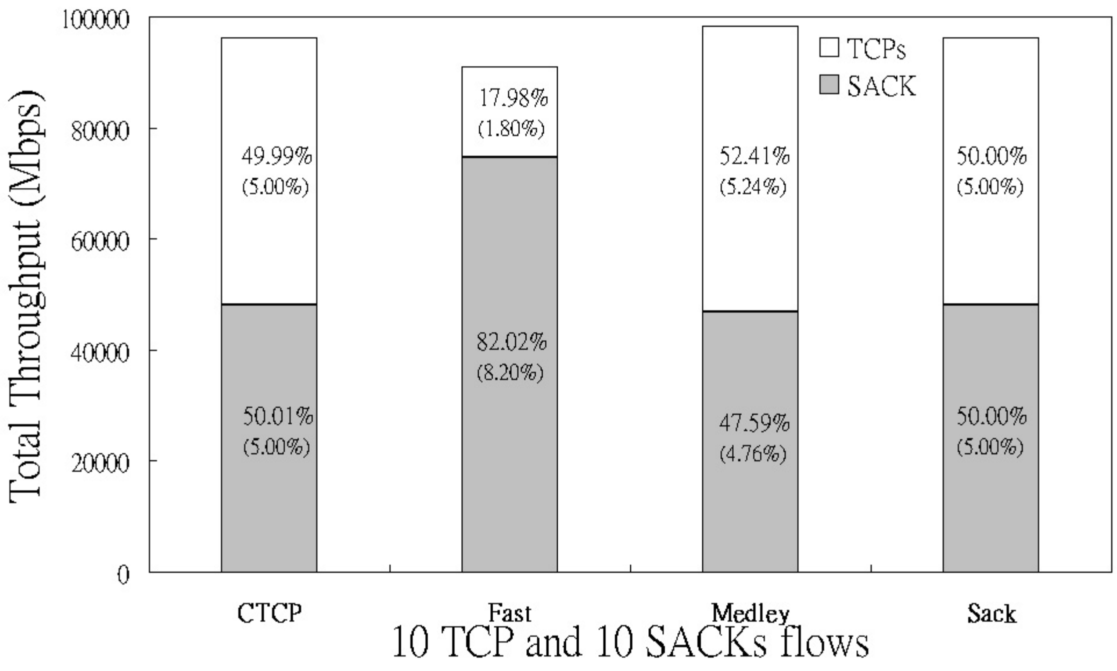
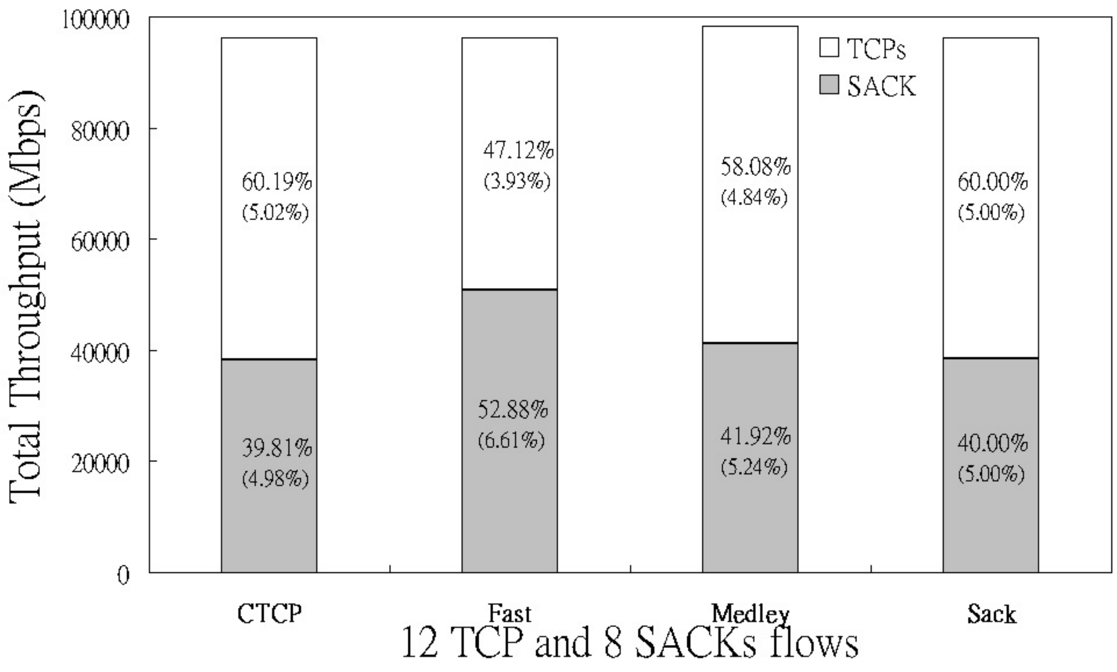


Figure 5.22: The percentage and throughput for ten flows of one TCP variant vs. ten SACK flows.



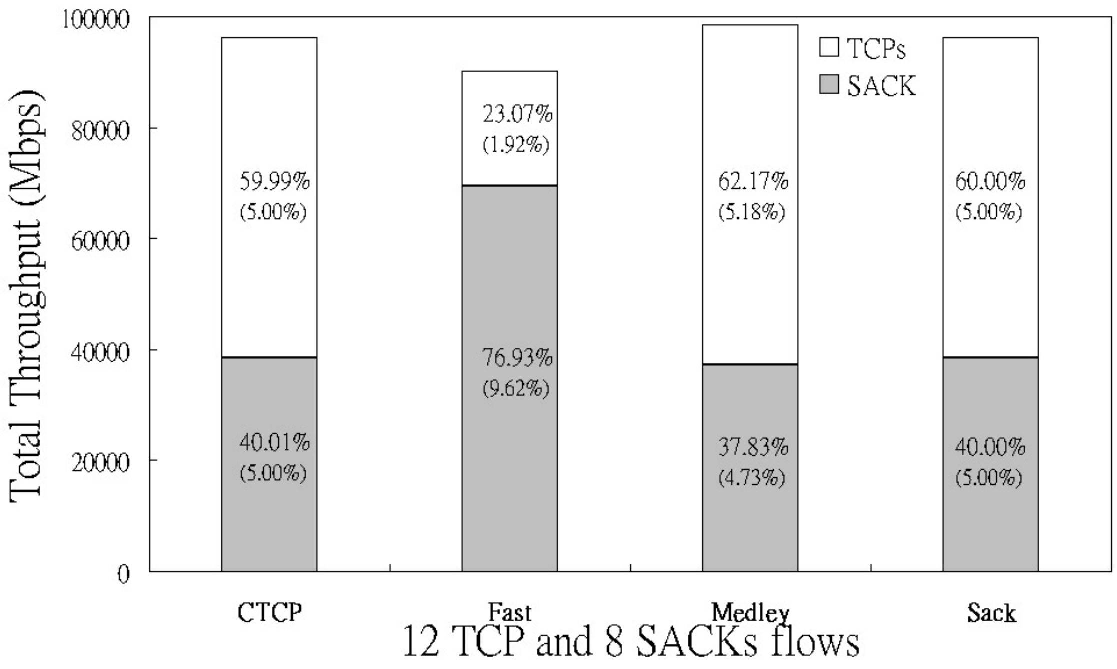
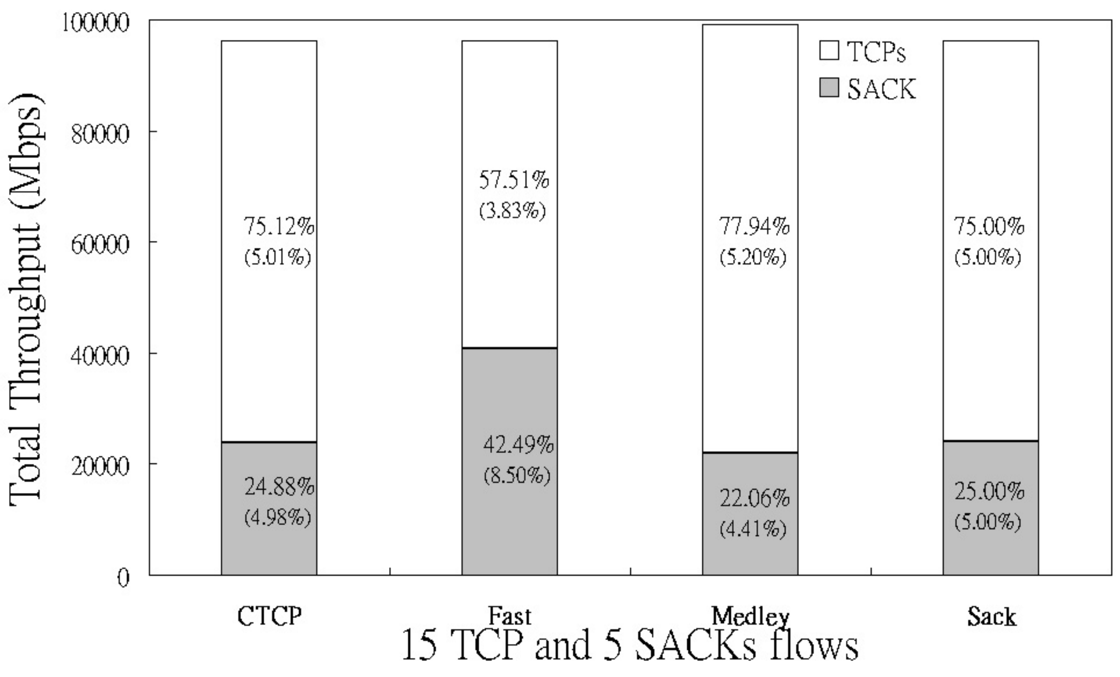


Figure 5.23: The percentage and throughput for twelve flows of one TCP variant vs. eight SACK flows.



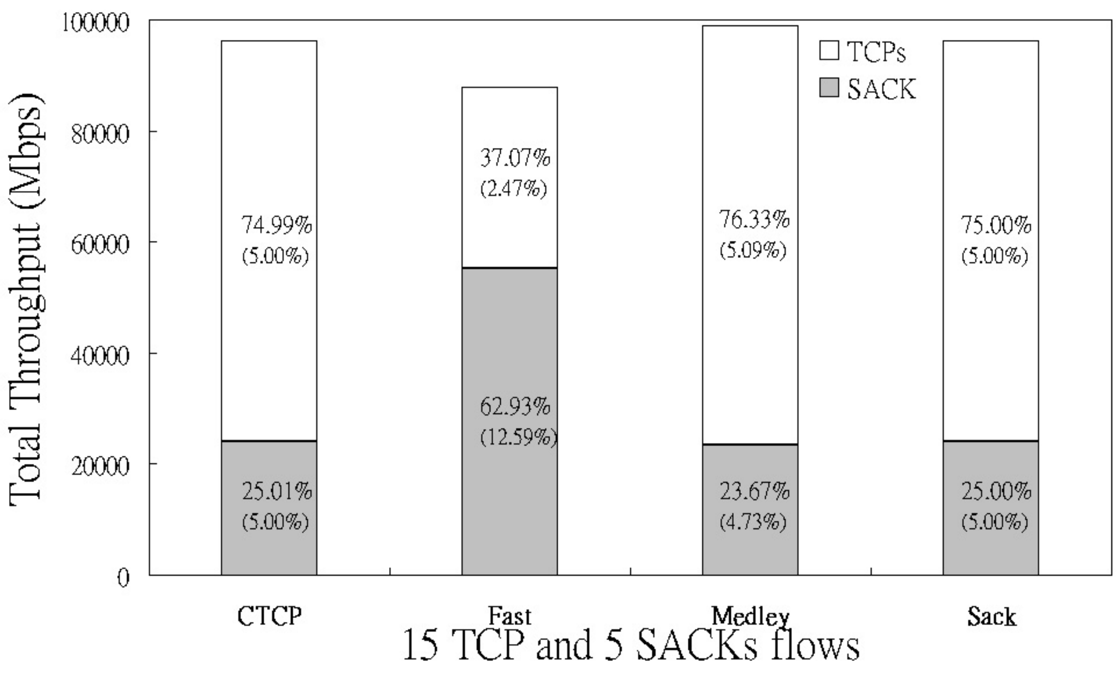
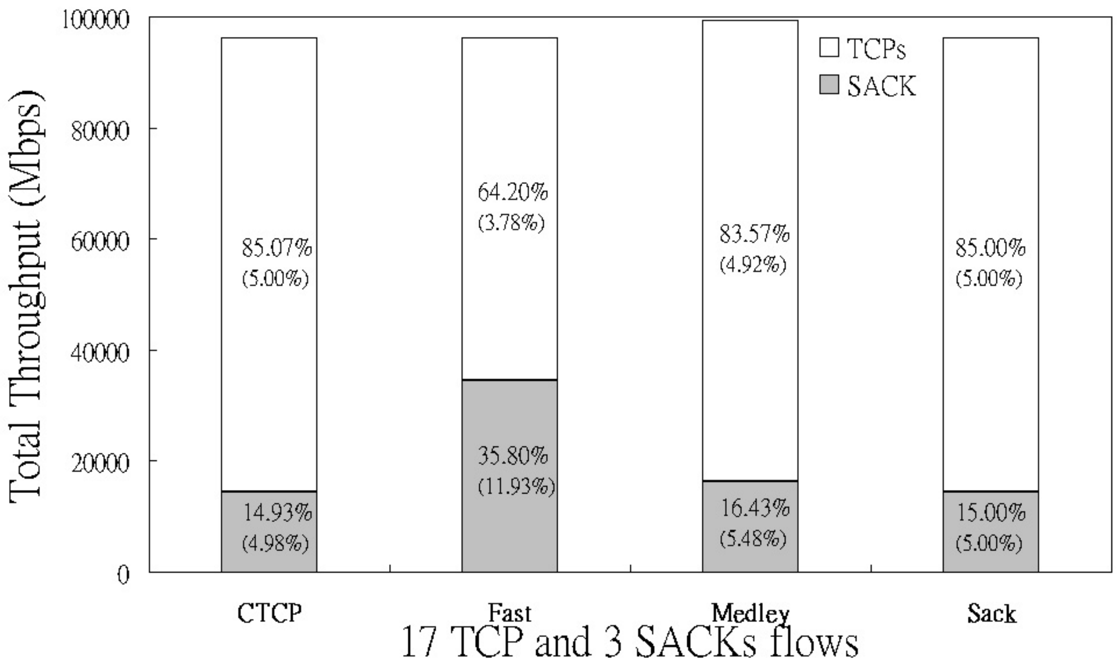


Figure 5.24: The percentage and throughput for fifteen flows of one TCP variant vs. five SACK flows.



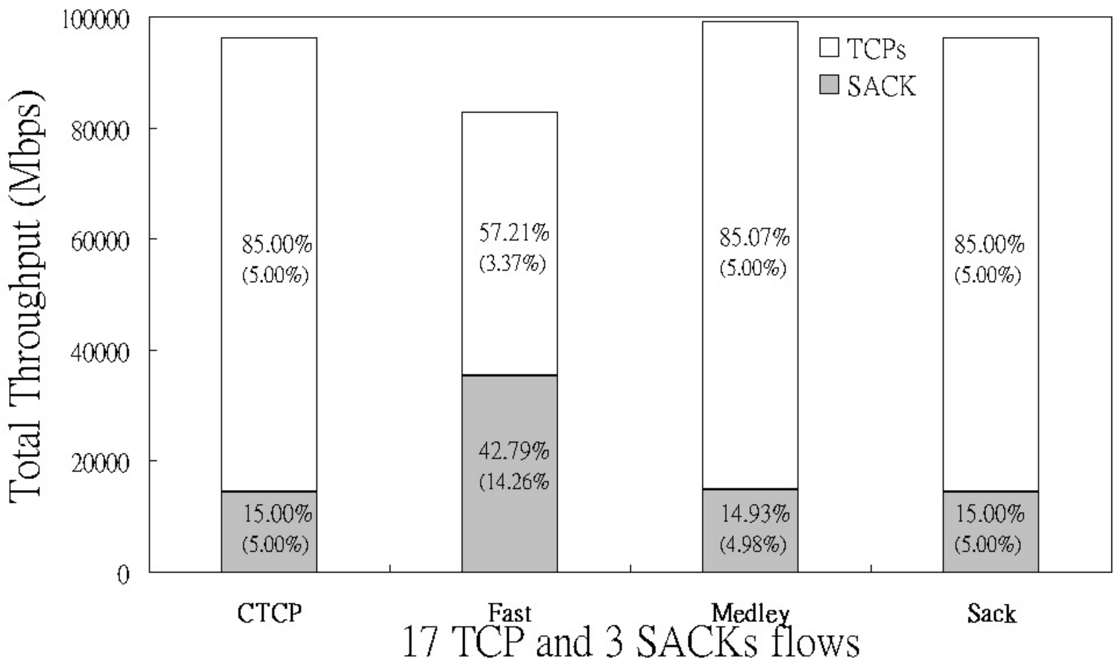
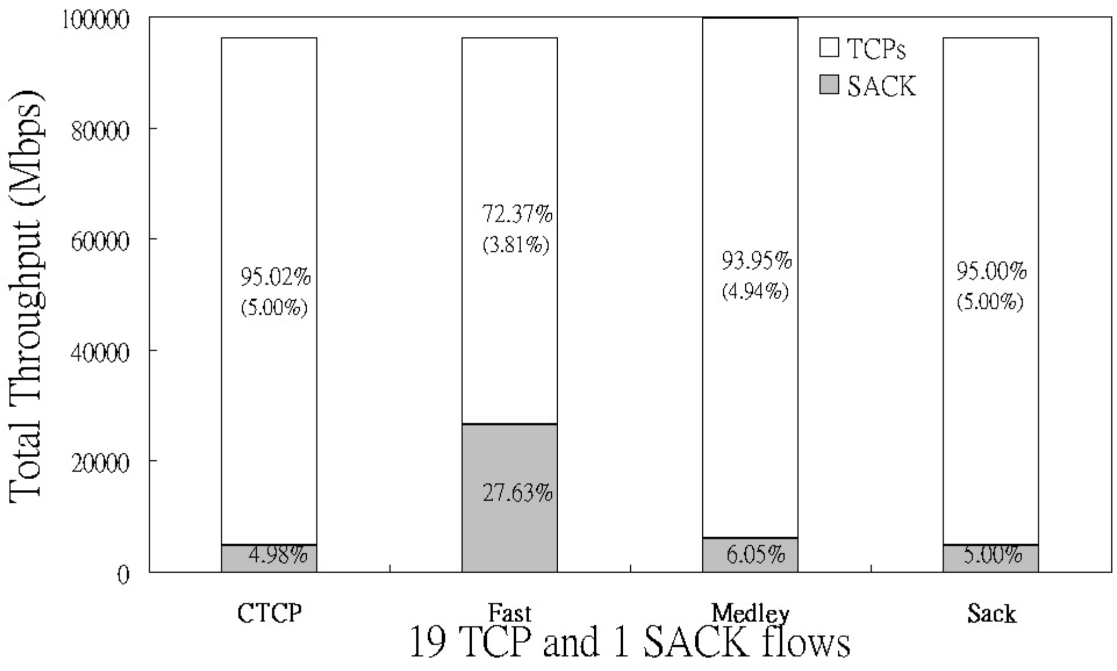


Figure 5.25: The percentage and throughput for seventeen flows of one TCP variant vs. three SACK flows.



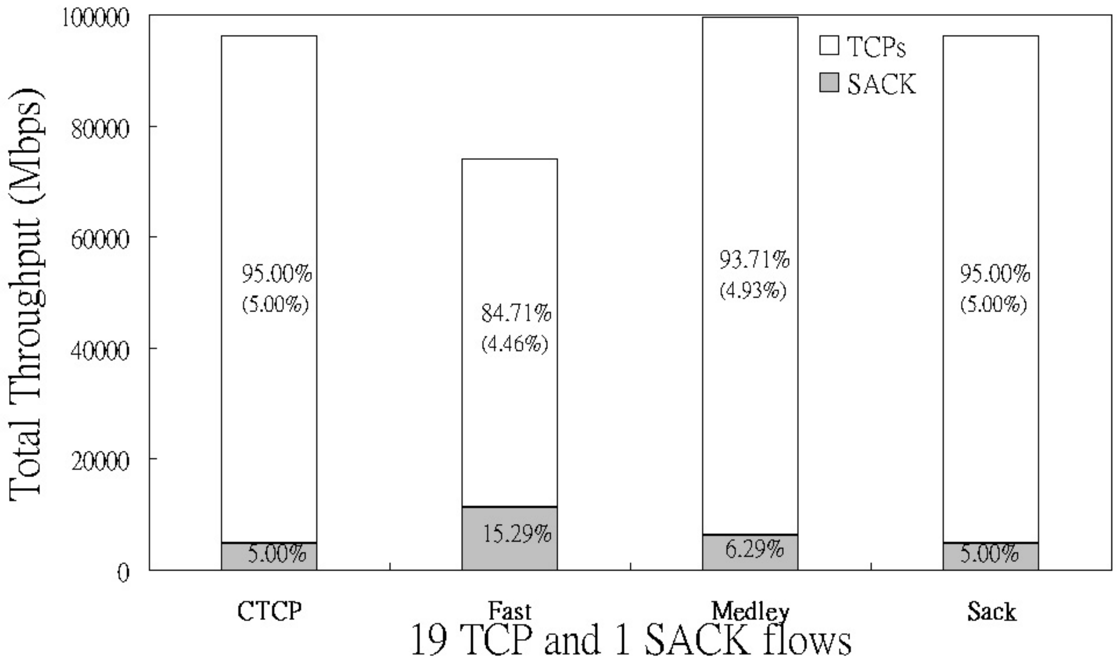


Figure 5.26: The percentage and throughput for nineteen flows of one TCP variant vs. one SACK flow.

Chapter 6

Medley TCP Implementation

6.1 TCP and Medley TCP Stacks of Linux

We briefly describe the current TCP stack of Linux and the implementation of Medley TCP in Linux. Figure 6.1 shows the basic architecture of all layers in Linux. The implementation of TCP is probably the most complex part of the networking code in the Linux kernel [62]. Although both ends of a TCP connection can be sender and receiver simultaneously, we separate explanations for the receiver behavior (the

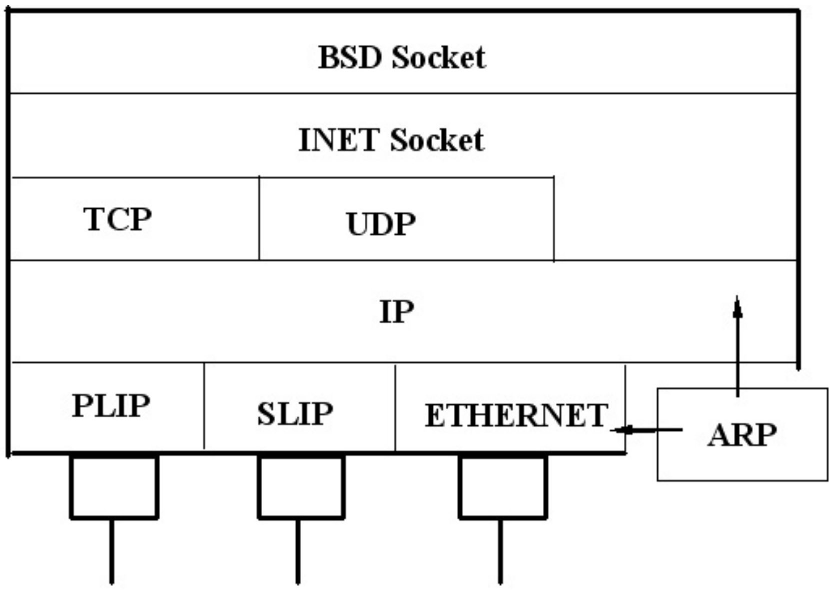


Figure 6.1: The basic architecture.

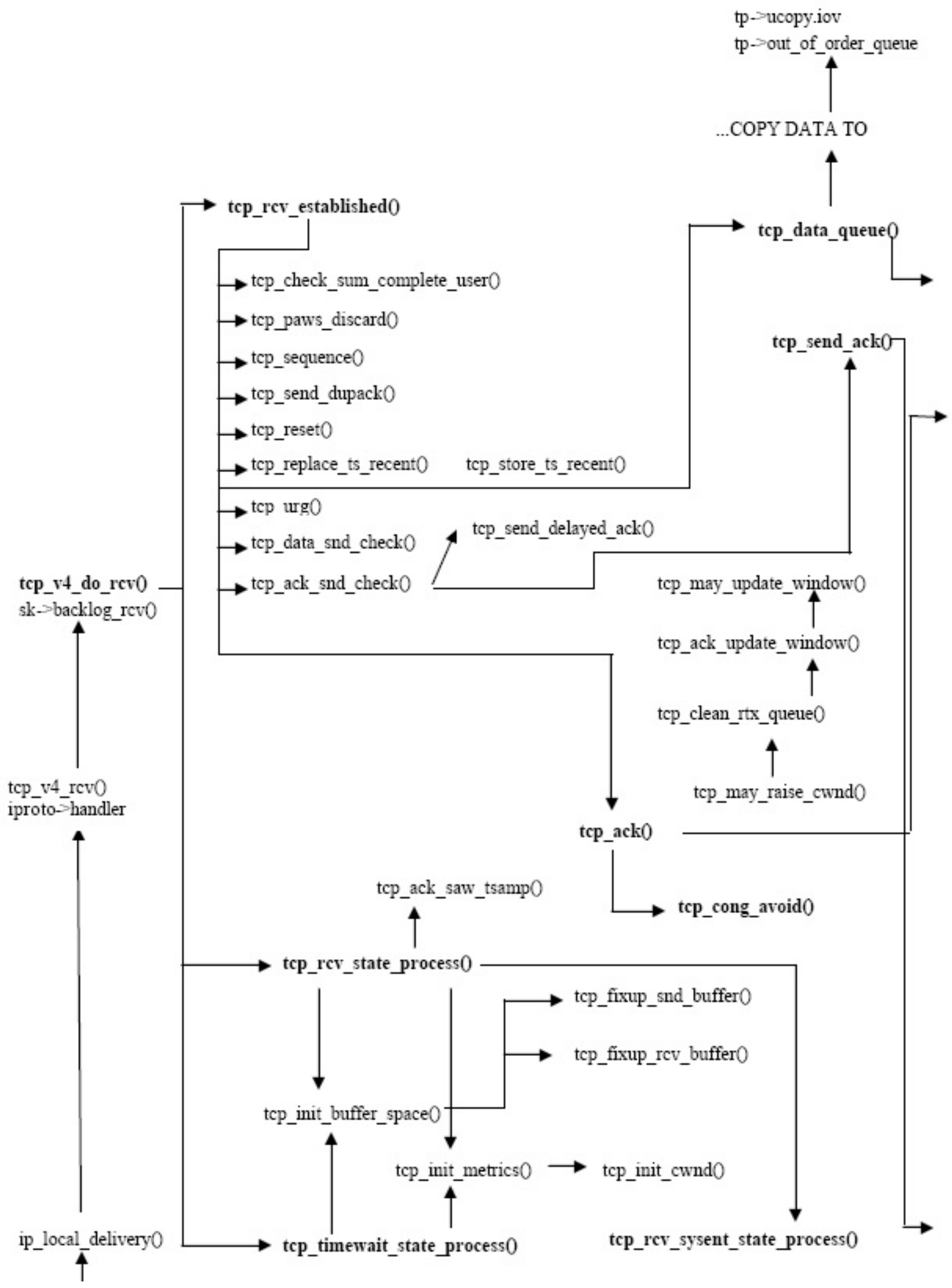


Figure 6.2: Input processing of TCP.

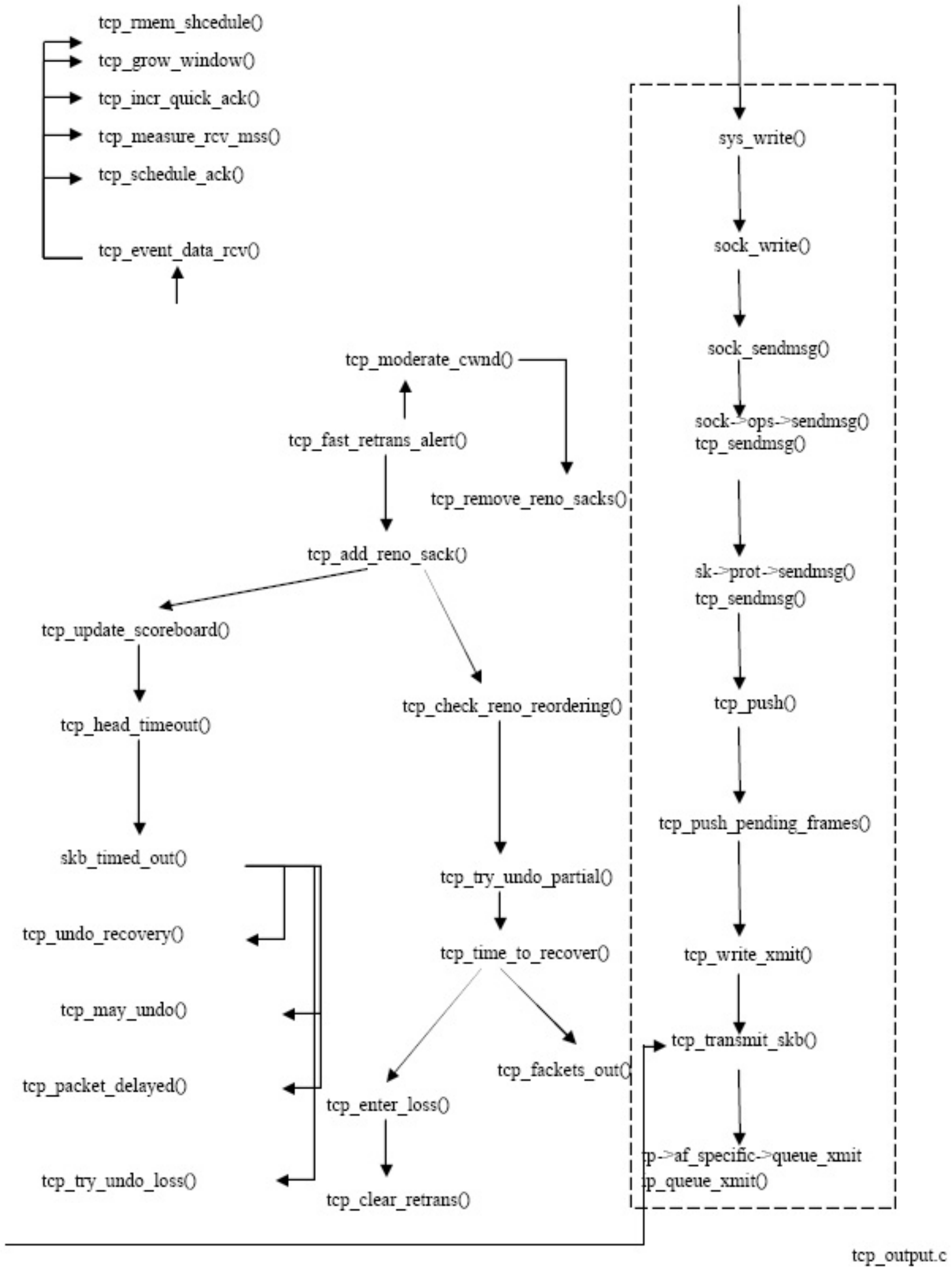


Figure 6.3: Output processing of TCP.

host receives data and sends ACKs) and the sender behavior (the host sends data, receives ACKs, retransmits lost packets and adjusts congestion window size). The complexity of the latter is significantly higher. The main files of TCP code are located in `net/ipv4`, except header files which are in `include/net`.

Fig. 6.2 and Fig. 6.3 depict the TCP data path and are meant to be viewed side by side [62]. TCP input processing is described in Fig. 6.2 and output processing of TCP is illustrated by Fig. 6.3. TCP input is mainly implemented in `net/ipv4/tcp_input.c`. This is the largest part of the TCP code. It deals with the reception of a TCP packet. The sender and receiver code is tightly coupled as an entity can be both at the same time. On the other hand, TCP output code is mainly implemented in `net/ipv4/tcp_output.c`. It deals with packets going out of the host and includes both data packets from the sender and ACKs from the receiver. For more details, please see [63].

Two extra files (tcp_medley.c and tcp_medley.h) are added to the TCP part of Linux with kernel 2.6.23 for Medley TCP implementation. Similar to TCP Vegas, timestamp structure is defined to record each sent packet length and its sending time with microsecond resolution[1]. By using this timestamp, Medley TCP can perform more accurate RTT computations for a more precise estimation of the network state.

6.2 Internet Measurements

Now, we present measurements of SACK, CTCP[2], and Medley TCP over the Internet[3]. Specifically, we measured three TCP variants transfers between the National Chiao Tung University (NCTU) and NCTU, between NCTU and the National Changhua University of Education (NCUE), and between NCTU and Goosean Media Inc.

[1]Note that the clock resolution in conventional TCP is 500 ms, which gives coarse-grained for the measurements of RTT.

[2]A Linux patch derived from this has been developed by Netlab at Caltech. http://netlab.caltech.edu/lachlan/ctcp/index.php

[3]It is not free to use FAST TCP because FastSoft Inc. was founded by Caltech professor Dr. Steven Low who, with his research team, developed FAST TCP.

which is located in San Mateo, California, USA (CA). Figure 6.4 shows the test-bed network. There are four computers in this test: computer A, B, C, and D. Computer A, whose IP address is 140.113.215.238, is a sender with SACK, CTCP, or Medley TCP algorithm in NCTU. Computer B with IP 140.113.252.129 in NCTU computer C with IP 163.23.224.36 in NCUE, and computer D with IP 66.79.166.158 in CA are receivers. In addition, the operating systems of three computers are all Linux Fedora with kernel 2.6.23 and all computers are equipped with a 2.8 GHz Intel Celeron processor, 512 MB RAM, and 100/10M Ethernet card. Moreover, when testing the performance of Medley TCP, we only changed TCP stack to Medley TCP scheme in computer A and did not modify any mechanism running in computer B, C, and D.

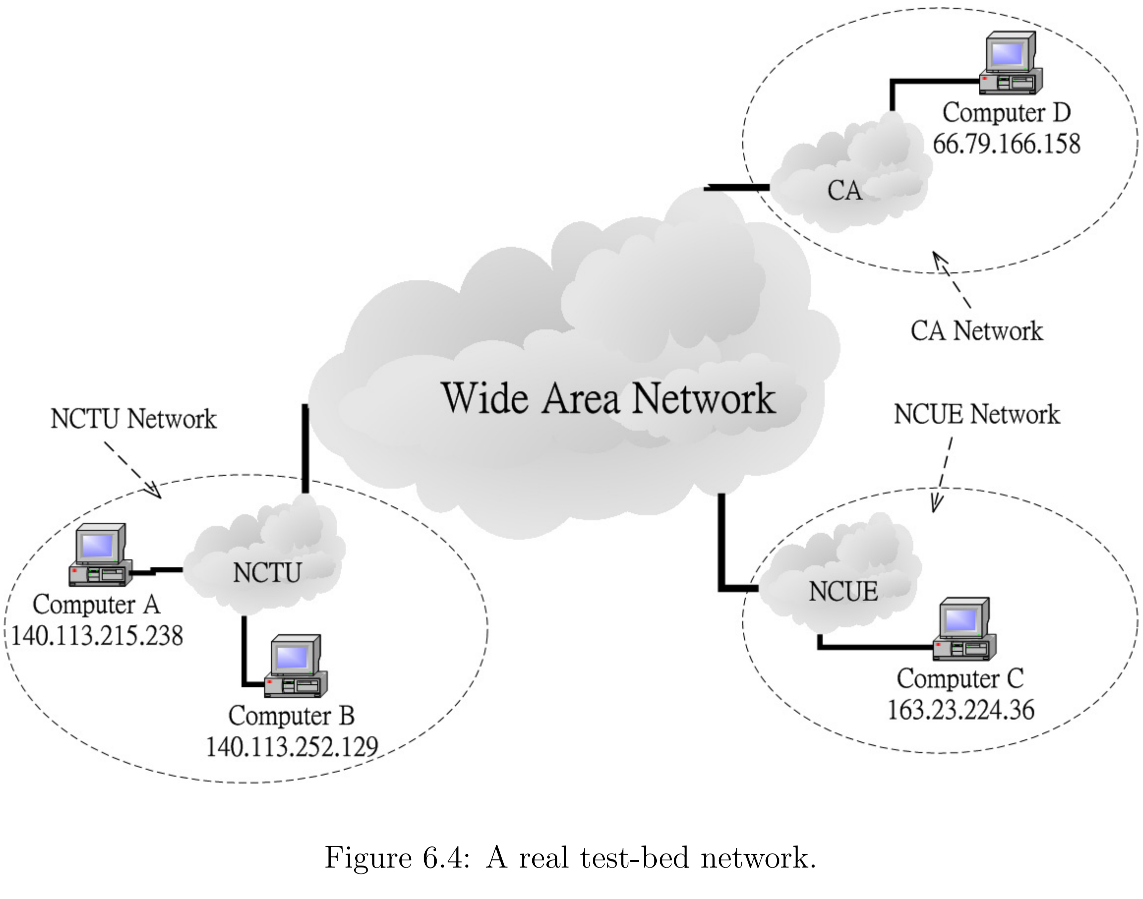


Figure 6.4: A real test-bed network.

The information of connections such as the number of hops between computer A and computer B, between computer A and computer C, and and between computer A and computer D is as shown in Table 6.1. The results are derived from a set

Table 6.1: The information of connections

	Computer A – Computer B		Computer A – Computer C		Computer A – Computer D	
hop	IP address	delay(ms)	IP address	delay(ms)	IP address	delay(ms)
1	140.113.215.238	0	140.113.215.238	0	140.113.215.238	0
2	140.113.215.254	0.30	140.113.215.254	0.33	140.113.215.254	0.32
3	140.113.0.166	0.32	140.113.0.166	0.28	140.113.0.166	0.30
4	140.113.0.205	0.37	140.113.255.111	1.74	211.76.255.194	1.53
5	140.113.252.129	0.41	203.72.43.121	1.91	203.160.226.137	1.77
6	X	X	192.83.196.120	5.39	203.78.186.26	1.82
7	X	X	140.128.251.85	6.33	203.160.226.190	148.52
8	X	X	192.168.202.254	7.12	203.160.225.6	152.83
9	X	X	163.23.224.254	9.56	203.160.225.154	144.09
10	X	X	163.23.224.36	10.28	154.54.7.173	155.37
11	X	X	X	X	154.54.2.138	144.08
12	X	X	X	X	154.54.6.74	140.12
13	X	X	X	X	38.99.208.14	141.38
14	X	X	X	X	66.79.166.158	196.86

of runs over a seven day period from May 8-16, 2008. Each run consists of a set of twenty one transfers from computer A to computer B (from NCTU to NCTU), twenty one transfers from computer A to computer C (from NCTU to NCUE), and twenty one transfers from computer A to computer D (from NCTU to CA) — SACK, CTCP, and Medley TCP send files with size 106 KB, 4.33 MB, 11.1 MB, 34 MB, 100 MB, 583 MB, and 930 MB, respectively. We inserted an 8 minute delay between each transfer in a run to give the network a chance to settle down, a run started approximately once every hour, and we shuffled the order of the transfers within each run.

Table 6.2, 6.3 and 6.4 show the results for all transfers, where the units of 'Transmission Time' and 'Average Throughput' are second and Mbps, respectively. We could see that Medley TCP spends less time to complete the data transmission than other TCP variants do whether the data size is large or small, and the RTT is long or short. For example, the average throughput of Medley TCP (19.42 Mbps) to transmit 930 MB data from NCTU to NCTU is about 1.24 times higher than that of SACK (15.70 Mbps). Our proposed mechanism spends 0.51 seconds, which is about eight tenths of the time SACK used, to finish 4.33 MB data transmission from NCTU to NCUE. Similarly, compared with CTCP, the throughput improvement of Medley TCP is kept between 5% and 16%, and between 4.5% and 14% when sending data from computer A to computer B and from computer A to computer C, respectively. Furthermore, compared with SACK and CTCP, the performance improvement of Medley TCP is kept between 79% and 164%, and between 1% and 26% when sending data from NCTU to CA, respectively.

Table 6.2: Internet measurements between NCTU and NCTU

	NCTU – NCTU					
	Transmission Time			Average Throughput		
data size	SACK	CTCP	Medley	SACK	CTCP	Medley
106 KB	0.0183	0.0171	0.0163	5.7	6.20	6.50
4.33 MB	0.58	0.55	0.47	7.6	8.20	9.20
11.1 MB	1.19	1.09	0.95	9.6	10.44	11.68
34 MB	3.32	3.05	2.63	10.5	11.24	12.94
100 MB	9.05	8.73	7.08	11.3	12.29	14.13
583 MB	44.06	41.33	35.33	13.3	14.47	16.50
930 MB	60.45	55.21	47.89	15.7	17.08	19.42

Table 6.3: Internet measurements between NCTU and NCUE

	NCTU – NCUE					
	Transmission Time			Average Throughput		
data size	SACK	CTCP	Medley	SACK	CTCP	Medley
106 KB	0.0488	0.0418	0.0394	2.2	2.50	2.70
4.33 MB	0.65	0.57	0.51	6.8	7.78	8.50
11.1 MB	1.46	1.29	1.14	7.8	8.86	9.76
34 MB	4.25	3.75	3.36	8.2	9.32	10.13
100 MB	11.22	9.91	8.55	9.1	10.30	11.69
583 MB	55.35	47.25	43.93	10.8	12.70	13.27
930 MB	83.44	73.65	65.77	11.4	12.96	14.14

Table 6.4: Internet measurements between NCTU and CA

	NCTU – CA					
	Transmission Time			Average Throughput		
data size	SACK	CTCP	Medley	SACK	CTCP	Medley
106 KB	1.44	0.687	0.546	0.074	0.154	0.194
4.33 MB	4.57	2.33	1.94	0.947	1.858	2.232
11.1 MB	5.68	3.25	2.99	1.954	3.415	3.712
34 MB	11	6.25	6.15	3.091	5.440	5.528
100 MB	31.2	15.7	15.6	3.205	6.369	6.410
583 MB	158	84.8	84.6	3.690	6.875	6.891
930 MB	259	135	134	3.591	6.889	6.940

Chapter 7

Conclusions and Future Work

TCP is the main congestion control method for the Internet, it would remain effective and efficient as the Internet evolves. To conclude this dissertation, in Section 7.1, we argue for desirability of Medley TCP from various practical standpoints and believe that Medley TCP would be useful in the Internet. A summary of the contributions of this dissertation is given in Section 7.2, and some directions for future work are discussed in Section 7.3.

7.1 Desirability of Medley TCP

When we design Medley TCP, three basic design principles are considered, compatibility, deployability, and flexibility.

7.1.1 Compatibility

A newly proposed TCP is compatible with today's TCP when they are running concurrently in the same network smoothly. Note that the best situation is that a new TCP does not cause any detrimental effects to the current TCP and vice versa. However, in present Internet, a loss-based TCP connection uses most of the buffer space in the bottleneck node when competing with a delay-based TCP connection. A delay-based TCP may be therefore not compatible with a loss-based

TCP. Comparatively, Medley TCP coexists peacefully with a loss-based TCP like SACK without stealing bandwidth from it. Its improvement is attributed to its efficient utilization of the available bandwidth.

7.1.2 Deployability

A new proposed TCP must be feasible to deploy over the existing Internet. To realize this, there should be little or no changes required at intermediate routers. For the two end hosts, only modifying one side is better than two sides. Generally speaking, modification in senders rather than in receivers is preferred. Accordingly, we try to modify a TCP sender stack which may make Medley TCP easily to deploy in real networks. Any one interested in Medley TCP can single-handedly install Medley TCP stack without cooperating with the other side.

7.1.3 Flexibility

Flexibility refers to whether a new TCP can accommodate different environments effectively such as a network with long propagation delay, high bandwidth, and/or random packet loss. Some TCP algorithms work well in certain environments but not in others. For example, TCP Reno and SACK work well in the wired networks, but suffer performance degradation in wireless networks and heterogeneous networks. We think that Medley TCP is a more flexible algorithm than SACK, CTCP, and FAST TCP. Medley TCP not only obtains significant improvement on throughput performance in large BDP networks, but also obtains higher throughput in wireless networks.

7.2 Summary of Contributions

In this dissertation, we have proposed and studied a novel end-to-end congestion control mechanism called Medley TCP. There are three main contributions as described below.

(1) The core idea of Medley TCP. Medley TCP differs from the conventional TCP in three ways.

(a) it does not use slow start threshold anymore because unsuitable slow start threshold value may cause TCP's performance down.

(b) it combines advantages and characteristics of both loss-based and delay-based TCPs by incorporating an adaptive delay-based component into a loss-based TCP algorithm.

(c) it may distinguish between congestion loss and random loss in the source side and does not need a help from the base station.

(2) Demonstration of Medley TCP's advantages. Both simulation results and Internet measurements indicate that Medley TCP can achieve significant improvement not only in utilization, fairness, friendliness, and buffer size requirement, but also in wireless networks and high bandwidth networks.

(3) Implementation in Fedora 8 with Kernel 2.6.23. Medley TCP has been implemented in the Linux kernel 2.6.23 of Fedora 8. Because we refer to TCP Vegas' source code, some functions are similar to Vegas. For example, Medley TCP maintains a timestamp queue for all outstanding packets to perform more accurate RTT calculation and obtain more accurate estimation of network.

7.3 Future Work

We think some interesting directions for future work may be considered.

Medley TCP works with routers embedded in other AQM schemes - The drop tail queue management scheme drops packets only when the buffer is full, other AQM algorithms use their distinctive rules to drop packets before buffer overflows. The goal is to give the TCP connections an early indication of impending congestion. In fact, current routers in Internet only use drop tail queue management because implementing drop tail is easier than other AQM schemes. However, RFC2309 strongly recommends that Internet routers should implement some AQM mechanism to manage queue lengths, reduce end-to-end latency, reduce packet dropping, and so on. Therefore, it may be worthwhile to study the performance of Medley TCP with some AQM scheme installed at the routers.

Applications over Medley TCP - Currently SACK is widely used to support many applications and services. If we expect that our proposed mechanism could be popularly used in the Internet, further study should be done to see whether Medley TCP works well when actual applications run on it.

More real Internet measurements - It seems that the measurements only in Taiwan is not enough. In order to get more real Internet measurements, a large scale work in the future may be to build a network measurement environment to make anyone interesting in Medley TCP be able to test it.

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] K. Takagaki, H. Ohsaki, and M. Murata, "Analysis of a window-based flow control mechanism based on TCP Vegas in heterogeneous network environment," *IEICE Trans. Commun.*, vol. E85-B, no. 1, pp. 89–97, 2002.

[10] 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.

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

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

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

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

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

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

[17] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-speed and Long Distance Networks," in *Proc. of IEEE INFOCOM*, pp. 1–12, April 2006.

[18] 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.

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

[20] 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.

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

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," in *Proc. of IEEE INFOCOM*, pp. 2490–2501, March 2004.

[30] T. Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," in *ACM SIGCOMM Computer Communication Review*, vol. 33, pp. 83–91, April 2003.

[31] L. Guo and I. Matta, "The War between Mice and Elephants," in *tech. rep. BU-CS-2001-005, Boston University, Computer Science Department*, May 2001.

[32] S. Boyden, A. Mahanti, and C. Williamson, "TCP Vegas Performance with Streaming Media," in *IEEE IPCCC 2007*, pp. 35–44, April 2007.

[33] D.P. Hong, C. Albuquerque, C. Oliveira, and T. Suda, "Evaluating the impact of emerging streaming media applications on TCP/IP performance," in *IEEE Communications Magazine*, vol. 39, pp. 76–82, April 2001.

[34] O. Ait-Hellal and E. Altman, "Analysis of TCP Vegas and TCP Reno," in *IEEE ICC'97*, vol. 1, pp. 495–499, June 1997.

[35] Y.-C. Lai and C.-L. Yao, "Performance Comparison between TCP Reno and TCP Vegas," in *IEEE ICPADS'2000*, pp. 61–66, July 2000.

[36] A. De Vendictis, A. Baiocchi, and M. Bonacci, "Analysis and Enhancement of TCP Vegas Congestion Control in a Mixed TCP Vegas and TCP Reno Network Scenario," in *Performance Evaluation (Elsevier)*, vol. 53, pp. 225–253, August 2003.

[37] K. N. Srijith, L. Jacob, and A. L. Ananda, "TCP Vegas-A: Solving the Fairness and Rerouting Issues of TCP Vegas," in *IEEE IPCCC'2003*, pp. 309–316, April 2003.

[38] S. Floyd and V. Jacobson, "Connection with Multiple Congested Gateways in Packet-Switched Networks, Part1: One-way Traffic," in *ACM Computer Communication Review*, vol. 21, pp. 30–47, August 1991.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

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

[45] 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.

[46] 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.

[47] 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.

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

[49] 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.

[50] 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.

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

[52] H. Wang, H. Xin, D. S. Reeves, and K. G. Shin, “A Simple Refinement of Slow-start of TCP Congestion Control,” in *IEEE Computers and Communications (ISCC)*, pp. 98–105, July 2000.

[53] H. Wang and C. Williamson, “A New Scheme for TCP Congestion Control: Smooth-Start and Dynamic Recovery,” in *MASCOTS’98*, pp. 69–75, July 1998.

[54] Y. Nishida, “Smooth Slow-Start: Refining TCP Slow-start for Large-Bandwidth with Long-Delay Networks,” in *Annual Conference on Local Computer Networks*, pp. 52–60, October 1998.

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

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

[57] Y.-T. Li, “Evaluation of TCP Congestion Control Algorithms on the Windows Vista Platform,” in *SLAC-TN-06-005*, pp. 1–28, June 2006.

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

[59] D. Chiu and R. Jain, “Analysis of the increase and decrease algorithms for congestion avoidance in computer networks,” in *Computer Networks*, vol. 17, pp. 1–14, June 1989.

[60] 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.

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

[62] M. Rio, T. Kelly, M. Goutelle, R. Hughes-Jones, and J.-P. Martin-Flatin, “A Map of the Networking Code in Linux Kernel 2.4.20,” tech. rep., March 2004.

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

Curriculum Vitae

Cheng-Yuan Ho was born in Taipei, Taiwan, R.O.C., in 1981. He received his B.S. degrees in Mathematics, and Information and Computer Education from National Taiwan Normal University in 2003. Then, he studied in the Department of Computer Science and Information Engineering, National Chiao Tung University. After one year, he enrolled into Ph.D. program directly. Now, he is currently pursuing the Ph.D. degree in Computer Science, National Chiao Tung University. He was working for the Wireless and Networking Group of Microsoft Research Asia, Beijing, China from Dec., 2005 to Sep., 2006. His research interests include the design, analysis, and modelling of the congestion control algorithms, high speed networking, QoS, and mobile and wireless networks.

Publication List

- Journal Paper

1. Cheng-Yuan Ho and Yaw-Chung Chen, "Snug-Vegas and Snug-Reno: Performance Improvement of TCP over Heterogeneous Networks," *IEE Proceedings – Communications*, Vol. 153, Issue 2, pp. 169-176, April 2006.
2. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, "Gallop-Vegas: An Enhanced Slow-Start Mechanism for TCP Vegas," *Journal of Communications and Networks* (*KICS*), Vol. 8, No. 3, pp.351-359, September 2006.
3. Cheng-Yuan Ho, Yaw-Chung Chen, and Cheng-Yun Ho, "Improving Performance of Delay-based TCPs with Rerouting," *IEEE Communications Letters*, Vol. 11, Issue 1, pp. 88-90, January 2007.
4. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, "TCP-Ho: A Congestion Control Algorithm with Design and Performance Evaluation," *IEICE Transactions on Communications*, Vol. E90-B, No. 3, pp. 516-526, March 2007.
5. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, "WARD: A Deterministic Fluid Model," *IET Communications*[1], Vol. 1, Issue 4, pp. 711-717, August 2007.
6. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, "WARD: a transmission control protocol-friendly stateless active queue management scheme," *IET Communications*, Vol. 1, Issue 6, pp. 1179-1186, December 2007.
7. Cheng-Yuan Ho, Yaw-Chung Chen, Yi-Cheng Chan, and Cheng-Yun Ho, "Fast Retransmit and Fast Recovery Schemes of Transport Protocols: A

[1]It is re-branded from IEE Proceedings – Communications because of the merger of The Institution of Electrical Engineers (IEE) and Institution of Incorporated Engineers (IIE).

Survey and Taxonomy," *Computer Networks*, Vol. 52, Issue 6, pp. 1308-1327, April 2008.

8. Yi-Cheng Chan, Chia-Liang Lin, and Cheng-Yuan Ho, "Quick Vegas: Improving Performance of TCP Vegas for High Bandwidth-Delay Product Networks," *IEICE Transactions on Communications*, Vol. E91-B, No. 4, pp. 987-997, April 2008.

9. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, "Design and Performance Evaluation of an End-to-end Congestion Control Algorithm," Submitted to *IEEE/ACM Transactions on Networking.*

10. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, "Medley TCP: Modeling, Stability, and Performance," Submitted to *IEEE Transactions on Automatic Control.*

11. Chen-Hua Shih, Cheng-Yuan Ho, Wen-Kang Jia, and Yaw-Chung Chen, "A Cross-Layering IPv6 Fast Handover Scheme for Real-Time Applications in IEEE 802.16e Network," Submitted to *IEEE Communications Magazine.*

12. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, "Is TCP's Structure Possible to Be Changed?" Revised for *Communications of the ACM* (*CACM*).

13. Cheng-Yuan Ho, Yi-Cheng Chan, Chen-Hua Shih, and Yaw-Chung Chen, "Improving Performance of TCP over Wireless Networks," Submitted to *IEEE Communications Letters.*

14. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, "Performance Improvement of Delay-based TCPs in Asymmetric Networks," Submitted to *Computer Networks.*

15. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, "Demo-Vegas: An Efficient Mechanism of TCP-Vegas on Mobile IP Networks," Submitted to *Computer Networks.*

16. Cheng-Yuan Ho, and Yaw-Chung Chen, “Demo-Vegas Plus: Improving Performance of TCP-Vegas for Mobile IP Handoffs,” Submitted to *Computer Networks*.

17. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, “Aid-Vegas: An Aided Congestion Avoidance Mechanism for TCP Vegas,” Submitted to *Journal of Communications and Networks* (*KICS*).

18. Cheng-Yuan Ho, Yaw-Chung Chen, and Cheng-Yun Ho, “Fleet TCP: Refining Slow-Start Mechanism for High-Speed Networks,” Submitted to *Journal of Communications and Networks* (*KICS*).

- Conference Paper

1. Yi-Cheng Chan, Chia-Tai Chan, Yaw-Chung Chen, and Cheng-Yuan Ho, “Performance Improvement of Congestion Avoidance Mechanism for TCP Vegas,” in *Proceedings of The 10th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2004*, pp. 605-612, Newport Beach, California, USA, July 2004.

2. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, “An Enhanced Slow-Start Mechanism for TCP Vegas,” in *Proceedings of The 11th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2005*, Vol. 1, pp. 405-411, Fukuoka Institute of Technology (FIT), Fukuoka, Japan, July 2005.

3. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, “An Efficient Mechanism of TCP-Vegas on Mobile IP Networks,” in *Proceedings of IEEE INFOCOM 2005*, Vol. 4, pp. 2776-2780, and *IEEE Global Internet Symposium in conjunction with IEEE INFOCOM 2005*, pp. 1-5, Miami, Florida, USA, March 2005.

4. Cheng-Yuan Ho, Chen-Hua Shih, Yaw-Chung Chen, and Yi-Cheng Chan, “An Aided Congestion Avoidance Mechanism for TCP Vegas,” in *Proceedings of 2005 International Conference on Computer Networks and*

Mobile Computing, ICCNMC 2005, pp. 961-971, Zhangjiajie, Hunan, China, August 2005.

5. Cheng-Yuan Ho, Chen-Hua Shih, Yaw-Chung Chen, and Yi-Cheng Chan, "An Aided Congestion Avoidance Mechanism for TCP Vegas," *Lecture Notes in Computer Science (LNCS) 3619/2005*, pp. 961-971, August 2005.

6. Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen, "A TCP-Friendly Stateless AQM Scheme for Fair Bandwidth Allocation," in *Proceedings of International Conference on Autonomic and Autonomous Systems, and International Conference on Networking and Services, IEEE ICAS 2005 and ICNS 2005*, pp. 14-14 (6 pages), Papeete, Tahiti, French Polynesia, October 2005.

7. Yi-Cheng Chan, Chia-Tai Chan, Cheng-Yuan Ho, and Chia-Liang Lin, "Improving Performance of TCP Vegas for High Bandwidth-Delay Product Networks," in *Proceedings of The 8th IEEE International Conference on Advanced Communication Technology, ICACT 2006*, Vol. 1, pp. 459-464, Phoenix Park, Gangwon-Do, Korea, February 2006.

8. Wen-Kang Jia, Yaw-Chung Chen, and Cheng-Yuan Ho, "Traffic Behavior Analysis of Frame Bursting for SISO IEEE 802.3z Networks," in *Proceedings of Asia-Pacific Optical Communications Conference, SPIE (APOC 2006)*, Vol. 6354, pp. 63541L-1-63541L-11, Kimdaejung Convention Center, Gwangju, Korea, September 2006.

9. Kun Tan, Jingmin Song, Murari Sridharan, and Cheng-Yuan Ho, "CTCP-TUBE: Improving TCP-Friendliness Over Low-Buffered Network Links," in *Proceedings of Sixth International Workshop on Protocols for FAST Long-Distance Networks (PFLDnet 2008)*, The Schuster Building, The University of Manchester, UK, March 2008.