ABSTRACT

TCP PERFORMANCE ENHANCEMENT IN WIRELESS NETWORKS VIA ADAPTIVE CONGESTION CONTROL AND ACTIVE QUEUE MANAGEMENT

by
Kai Xu

The transmission control protocol (TCP) exhibits poor performance when used in error-prone wireless networks. Remedy to this problem has been an active research area. However, a widely accepted and adopted solution is yet to emerge. Difficulties of an acceptable solution lie in the areas of compatibility, scalability, computational complexity, and the involvement of intermediate routers and switches.

This dissertation reviews the current start-of-the-art solutions to TCP performance enhancement, and pursues an end-to-end solution framework to the problem. The most noticeable cause of the performance degradation of TCP in wireless networks is the higher packet loss rate as compared to that in traditional wired networks. Packet loss type differentiation has been the focus of many proposed TCP performance enhancement schemes. Studies conduced by this dissertation research suggest that besides the standard TCP's inability of discriminating congestion packet losses from losses related to wireless link errors, the standard TCP's additive increase and multiplicative decrease (AIMD) congestion control algorithm itself needs to be redesigned to achieve better performance in wireless, and particularly, high-speed wireless networks. This dissertation proposes a simple, efficient, and effective end-to-end solution framework that enhances TCP's performance through techniques of adaptive congestion control and active queue management. By end-to-end, it means a solution with no requirement of routers being "wireless-aware" or "wireless-specific".

TCP-Jersey has been introduced as an implementation of the proposed solution framework, and its performance metrics have been evaluated through

extensive simulations. TCP-Jersey consists of an adaptive congestion control algorithm at the source by means of the source's achievable rate estimation (ARE) – an adaptive filter of packet inter-arrival times, a congestion indication algorithm at the links (i.e., AQM) by means of packet marking, and a effective loss differentiation algorithm at the source by careful examination of the congestion marks carried by the duplicate acknowledgment packets (DUPACK).

Several improvements to the proposed TCP-Jersey have been investigated, including a more robust ARE algorithm, a less computationally intensive threshold marking algorithm as the AQM link algorithm, a more stable congestion indication function based on virtual capacity at the link, and performance results have been presented and analyzed via extensive simulations of various network configurations. Stability analysis of the proposed ARE-based additive increase and adaptive decrease (AIAD) congestion control algorithm has been conducted and the analytical results have been verified by simulations. Performance of TCP-Jersey has been compared to that of a "perfect", but not practical, TCP scheme, and encouraging results have been observed. Finally, the framework of the TCP-Jersey's source algorithm has been extended and generalized for rate-based congestion control, as opposed to TCP's window-based congestion control, to provide a design platform for applications, such as real-time multimedia, that do not use TCP as transport protocol yet do need to control network congestion as well as combat packet losses in wireless networks.

In conclusion, the framework architecture presented in this dissertation that combines the adaptive congestion control and active queue management in solving the TCP performance degradation problem in wireless networks has been shown as a promising answer to the problem due to its simplistic design philosophy, complete compatibility with the current TCP/IP and AQM practice, end-to-end architecture for scalability, and the high effectiveness and low computational overhead. The

proposed implementation of the solution framework, namely, TCP-Jersey, is a modification of the standard TCP protocol rather than a completely new design of the transport protocol. It is an end-to-end approach to address the performance degradation problem since it does not require split mode connection establishment and maintenance using special wireless-aware software agents at the routers. The proposed solution also differs from other solutions that rely on the link layer error notifications for packet loss differentiation.

The proposed solution is also unique among other proposed end-to-end solutions in that it differentiates packet losses attributed to wireless link errors from congestion induced packet losses directly from the explicit congestion indication marks in the DUPACK packets, rather than inferring the loss type based on packet delay or delay jitter as in many other proposed solutions; nor by undergoing a computationally expensive off-line training of a classification model (e.g., HMM), or a Bayesian estimation/detection process that requires estimations of *a priori* loss probability distributions of different loss types.

The proposed solution is also scalable and fully compatible to the current practice in Internet congestion control and queue management, but with an additional function of loss type differentiation that effectively enhances TCP's performance over error-prone wireless networks.

Limitatioms of the proposed solution architecture and areas for future researches are also addressed.

TCP PERFORMANCE ENHANCEMENT IN WIRELESS NETWORKS VIA ADAPTIVE CONGESTION CONTROL AND ACTIVE QUEUE MANAGEMENT

by
Kai Xu

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Engineering

Department of Electrical and Computer Engineering

May 2005

APPROVAL PAGE

TCP PERFORMANCE ENHANCEMENT IN WIRELESS NETWORKS VIA ADAPTIVE CONGESTION CONTROL AND ACTIVE QUEUE MANAGEMENT

Kai Xu

Dr. Nirwan Ansari, Dissertation Advisor — Date
Professor of Electrical and Computer Engineering, New Jersey Institute of Technology

Dr. Sirin Tekinay, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, New Jersey Institute of Technology

Dr. Mengchu Zhou, Committee Member — Date
Professor of Electrical and Computer Engineering, New Jersey Institute of Technology

Dr. Yun-qing Shi, Committee Member — Date
Professor of Electrical and Computer Engineering, New Jersey Institute of Technology

Dr. Cristian M. Borcea, Committee Member — Date
Assistant Professor of Computer Science, New Jersey Institute of Technology

BIOGRAPHICAL SKETCH

Author: Kai Xu

Degree: Doctor of Philosophy

Date: May 2005

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Engineering,
  New Jersey Institute of Technology, NJ, U.S.A., May, 2005
- Master of Science in Computer Imaging Engineering,
  Chiba University, Chiba, Japan, March, 1993
- Bachelor of Science in Electrical Engineering,
  Zhejiang University, Hangzhou, China, July, 1989

Major: Computer Engineering

Presentations and Publications:

Kai Xu, Ye Tian, and Nirwan Ansari, "TCP-Jersey for Wireless IP communications," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 4, pp. 747-756, May 2004.

Kai Xu, Ye Tian, and Nirwan Ansari, "Improving TCP Performance in Integrated Wireless Communications Networks," *Elsevier Journal on Computer Networks*, vol. 47, no. 2, pp. 219-237, February 2005.

Kai Xu and Nirwan Ansari, "Stability and Fairness of Rate Estimation Based AIAD Congestion Control in TCP," *IEEE Communications Letters*, vol. 9, no. 4, pp. 378-380, April 2005.

Ye Tian, Kai Xu, and Nirwan Ansari "TCP in Wireless Environment: Problems and Solutions," *IEEE Communications Magazine*, vol. 43, no. 3, pp. S27-S32, March 2005.

Gang Cheng, Kai Xu, and Nirwan Ansari, "Core-Stateless Proportional Fair Queuing for AF Traffic," *Proc. IEEE GLOBECOM 2004*, Dallas, Texas, U.S.A., December 2004.

Li Zhu, Gang Cheng, Kai Xu, and Nirwan Ansari, "Edge-Based Adaptive Queue Management," *IEE Proceedings on Communications*, under peer review.

To my wife for her love, patience, sacrifice, and support.

To my parents for their continuous encouragement.

ACKNOWLEDGMENT

I am fortunate to have Dr. Nirwan Ansari as my Ph.D dissertation research adviser, who has inspired, encouraged, supported, and given me the freedom to conduct research in the area of my own interest.

I am also grateful to the members of my dissertation committee, Prof. Sirin Tekinay, Prof. Mengchu Zhou, Prof. Yun-qing Shi, and Prof. Cristian M. Borcea, for their constructive discussions, suggestions, and careful reviews of this research.

Throughout my academic years in NJIT, I have met and made friends of many talented fellow students and researchers both in the Advanced Networking Laboratory and within the Electrical and Computer Engineering Department. I have enjoyed the friendly, competitive, collaborative, and pleasant research atmosphere that the lab and the department have fostered. The days we have shared common visions, debated different viewpoints will stay in my memory and inspire my future professional endeavor.

I am deeply in debt to my wife, without whom this work would be impossible. Her invisible hands of care and support filled the space between the lines, graphs, and equations.

Last, but not least, I would also like to thank the New Jersey Commission on Higher Educaton and New Jersey Commission on Science and Technology, the funding agencies who have provided financial support throughout my Ph.D. studies.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter Page

TABLE OF CONTENTS
(Continued)

Chapter Page

TABLE OF CONTENTS
(Continued)

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

OVERVIEW

In the past 15 years, the world has experienced the phenomenal growth and success of the global Internet, the packet-switched communications platform that has affected and changed, in many ways, our daily business and personal activities. Many so-called "disruptive technologies" have been invented or re-engineered as Internet Protocol (IP) centric, with examples such as the World Wide Web, on-line business transactions, IP-PBX, voice-over-IP (VoIP) and video-on-demand (VoD), grid computing, etc.

The Transmission Control Protocol (TCP) has been the protocol of choice for many applications requiring reliable connections, ranging from interactive sessions such as remote terminals, e-mail, and Web browsing, to bulk data transfers such as file transferring (e.g., FTP), distributed database and data warehouse networks, and storage area networks (SANs). Nowadays, most of the Internet traffic are carried by TCP. In June 1999, measurements at backbone routers, edge routers, and major exchange points showed 90-95% of the bytes in the network belonging to TCP traffic [1]. A February 2001 measurement from a translantic link showed that 95% of the bytes were from TCP [2]. By far, the TCP/IP protocol suite has delivered satisfactory performance due to its simplistic design philosophy and modular architecture. TCP with its associated window-based algorithms is the primary mechanism in today's Internet congestion control.

1.1 Empowering TCP for Wireless Communications

With the accelerated proliferation of wireless technologies and the applications and devices that make people connected whenever-and-wherever, wireless IP communications is becoming a integral part of our daily lives. Consumers and enterprises, as they become more and more dependent upon the TCP/IP technologies, demand that all IP-based applications, tools, and convenience to be available on the emerging new mobile and wireless devices.

While the Internet is growing exponentially in size and envisioned to become even more ubiquitous, and hence increasingly heterogeneous [3], the design of TCP has been challenged by the extension of connections over wireless links. Network designers are faced with the question of how to empower TCP so it works well in such hybrid wired/wireless environment, where packets can be lost not just because of link congestion, but for a variety of reasons [4].

TCP was originally designed for the wired networks with an assumption that the packet loss caused by bit errors in such networks would not exceed 1%. However, wireless links usually exhibit much higher bit error rate (BER) in the order of 10^{-3} to 10^{-6} [5, 6]. Assume the bit errors occur independent of each other, for a connection with average packet size of 1000 bytes, this BER translates to about 10^{2} or more packet error rate (PER) due to corruptions. However, TCP's congestion control is loss-driven. In other words, a TCP source probes the available network resource in a fairly slow pace, i.e., increment of one packet in window size for every round-trip time period, or linear in RTT, but reduces it transmission rate by 50% upon the detection of a packet loss. In a error-prone wireless network environment, this would cause severe throughput degradation, which is otherwise unnecessary. Packet loss due to wireless channel errors should be recovered without sacrificing the throughput. To this end, many solutions have been proposed; but as of today, there is no widely accepted and adopted remedy

to the performance degradation problem of TCP in wireless networks. A solution with minimum protocol stack software changes, minimum dependency on new, if any, or "wireless-aware" network equipment would be preferred for compatibility and practical gradual deployment. This is the motivation behind this dissertation research. Specifically, this dissertation investigates methodologies and algorithms, particularly from congestion control and active queue management perspective, in designing an end-to-end solution to enhance the TCP's performance in wireless networks.

1.2 Contributions

This dissertation contributes a new architecture for efficient TCP communications in the wireless and hybrid networks. In the same spirit of the Internet's simplistic design philosophy, this architecture effectively provides an end-to-end solution to the TCP performance degradation problem in wireless networks. We propose to approach the problem from two inter-related fronts. First, we propose to modify the TCP congestion control by using the sender's achievable rate estimation (ARE) based additive-increase and adaptive-decrease (AIAD) algorithm so that, upon the detection of packet loss, the sender's transmission rate reduction becomes less drastic as compared to standard TCP's static rate halving, and adaptive to the changing network conditions. Second, we propose an effective loss differentiation technique by a novel application of the packet-marking based active queue management (AQM) scheme, called congestion warning (CW) marking. The proposed mechanism combines the adaptive congestion control and explicit loss type differentiation to provide an integrated solution to TCP performance enhancement that preserves the TCP's end-to-end connection semantics.

The proposed solution is a modification of the standard TCP protocol rather than a completely new design of the transport protocol. It is an end-to-end approach

since it does not require split mode connection establishment and maintenance using special wireless-aware software agents at the routers. Our solution also differs from other solutions that rely on the link layer error notifications for packet loss differentiation.

The proposed solution is also unique among other proposed end-to-end solutions in that it differentiates packet losses attributed to wireless link errors from congestion induced packet losses directly from the explicit CW marking in the DUPACK packets, rather than inferring the loss type based on packet delay or delay jitter as in many other proposed solutions [7, 8, 9, 10]; nor by undergoing a computationally expensive off-line training of a classification model (e.g., HMM) [11], or a Bayesian estimation and detection process [12, 13, 14] that requires estimations of *a priori* loss probability distributions of different loss types.

The proposed solution is also scalable and fully compatible to the current practice in Internet congestion control and queue management, but with an additional function of loss type differentiation that effectively enhances TCP's performance over lossy wireless networks.

1.3 Organization of Dissertation

The rest of the dissertation is organized as follows. Chapter 2 provides background information of the congestion control algorithms implemented in today's standard TCP protocols.

Chapter 3 first illustrates the throughput degradation problem exhibited when unmodified TCP is used for data transferring over error-prone wireless links; and then follows the review and discussion of current state-of-the-art solutions proposed for its remedy.

Chapter 4 presents our design framework for a solution architecture that can effectively enhance TCP's performance.

Chapter 5 presents, in great detail, the implementation and performance evaluations of TCP-Jersey, an integrated realization of the solution framework previously introduced.

Chapter 6 is dedicated to the stability and fairness analysis of the achievable rate estimation based AIAD congestion control, with particular focus on the proper choice of the ARE estimator. It concludes both analytically and through simulations that an ARE estimator based on the inter-arrival time gaps of the data packet at the receiver is preferred over the original ARE that uses inter-arrival time gaps of the ACK packets at the source.

Chapter 7 applies the results from the previous chapter and presents an improved version of TCP-Jersey that uses the TS-ARE, a timestamps-based achievable rate estimation. In this chapter, a implementation of TS-ARE without code modification at the receiver side is presented and simulation results are analyzed. This chapter also takes into consideration the correlated error process on the wireless link.

In Chapter 8, the roles the active queue management (AQM) link algorithm play in the context of TCP congestion control and TCP-Jersey source's loss differentiation capability is studied. A basic network model is established and is later used to analyze the TCP/AQM pair as a feedback control system. Detailed equilibrium analysis is presented to compare the attainable throughput by standard AIMD as well as that by the proposed AIAD congestion control algorithm. Two improvements to the CW marking algorithm are introduced and extensive simulation results are analyzed.

With the ever-growing demand for real-time multimedia streaming, particularly in wireless or wired/wireless hybrid networks, congestion control and packet loss differentiation remains to be challenges for any rate-based protocols. Chapter 9 extends the ARE-based AIAD congestion control of the proposed TCP-Jersey

to potential applications or implementations of a rate-based transmission control protocol for real-time multimedia content delivery.

Specifically, Chapter 9 provides an analytic model of a generalized rate-based congestion control system that captures the essence of TCP-Jersey's source algorithm in a discrete-time form. The model developed in this chapter serves as a starting point or a step-stone toward future extension of TCP-Jersey, or more precisely, the design framework of TCP-Jersey, for real-time applications that do not use TCP as the transport protocol but do need an efficient congestion control as well as an effective loss differentiation in wireless networks.

Conclusions of this dissertation research are presented in Chapter 10 and outlook of future researches is discussed in Chapter 11.

1.4 Summary

This dissertation addresses the problem of TCP performance degradation in wireless IP communications. To solve this problem, we introduce an architecture that relies on the adaptive congestion control and simple effective loss differentiation at the source, supported by packet marking AQMs in the network. We use this architecture to develop an end-to-end solution called TCP-Jersey. Analysis and simulations show TCP-Jersey can achieve much better throughput as compared to other TCP variants in an error-prone wireless network environment. Several further enhancements of TCP-Jersey are also presented and substantiated with extensive simulations.

CHAPTER 2

BACKGROUND OF INTERNET CONGESTION CONTROL VIA TCP

In this chapter, we summarize the current Internet's congestion control mechanism, which is practically carried out by the operation of the Transmission Control Protocol, or TCP, residing at the fourth layer, the transport layer of the Open System Interconnection (OSI) network architecture.

2.1 Operation of TCP

A TCP session is an end-to-end two-way communication between the sender and the receiver. The sender sends a stream of packets to the receiver while the receiver acknowledges the receptions of correctly and in-order delivered packets in a way such that lost packets can be detected and retransmitted. The sender maintains a window, called the congestion window, or *cwnd*, by convention, that controls how many unacknowledged packets the sender is allowed to transmit into the network. This congestion window slides as the session progresses. The size of the congestion window also grows or shrinks according to the network conditions and the congestion control algorithms described below.

Each TCP packet, or, segment, in TCP's terminology, is assigned a sequence number indicating its relative position in the stream of the packets that the sender is transmitting[1]. The sequence number increases if proper acknowledgment is

[1]In real-world TCP implementations, sequence numbers are associated to each byte of data instead of a segment. The TCP sender still hands down the segments (a segment consists of multiple bytes, where the size of the segment is determined at the initial three-way handshake procedure [15]) to the IP layer function for transmission. The ACKs also bare sequence numbers indicating the received bytes. In the mean time, the *cwnd* and other window related parameters are also in unit of byte. Therefore, the granularity of the *send-ACK-send-more* process is in bytes. This is why TCP is sometimes referred to as a byte streaming protocol.

returned from the receiver. The receiver sends acknowledgment packets (ACK) to the sender in such a way that only the successfully received in-order packets are acknowledged. Therefore, it is in general a *positive acknowledgment* scheme. In particular, the receiver employs a *cumulative acknowledgment* mechanism such that an ACK is also assigned a sequence number indicating the sequence number of the *next* packet in-order that the receiver is expecting from the sender. In essence, in the cumulative acknowledgment scheme, the acknowledgment information carried in a later ACK packet covers the information carried in the previous ACK packet.

For example, if segments $1, 2, 3, 4, 5$ have been successfully received in the order they were sent, the receiver sends an ACK with sequence number of 6 to signify the sender that it is expecting segment 6. On the other hand, if, for instance, at the time the sender has its *cwnd* equal to 5; and packets $1, 2, 3, 4, 5$ have been sent, but packets 3 and 4 are lost in the network, while packet $1, 2, 5$ are received. In this scenario, the receiver sends ACK with sequence number 3 upon receiving packet 2 indicating that it has received successfully up to packets 2, and is expecting packet 3. Then when packet 5 is received, the receiver will still send an ACK with sequence number of 3, since it has not received the lost packet 3 yet. This ACK is called the duplicate ACK, or DUPACK. Therefore, the arrival of a DUPACK serves as the implicit negative acknowledgment, upon which the sender can infer that packet 3 was not properly received yet, and a retransmission may be in order.

Besides the DUPACKs triggered by the reception of out-of-order packets, DUPACKs are also sent by the receiver in case no packets have been received for a

---

This implementation detail should not affect our descriptions of the protocol nor would it have any significant impact to our analysis in this dissertation. Through out the analysis and simulations in this dissertation, we assume the granularity in packet rather than in byte. In fact, all our analysis and simulations assume fixed size packets for any particular TCP session. This assumption is valid since we focus our interest in the TCP sessions of long durations such as FTPs, where most of the packets or segments are of fixed size determined by the underlying MAC/PHY layers' maximum transmission unit, or MTU. The size of the TCP segment in bytes is called maximum segment size, or MSS, in TCP implementations.

Throughout this dissertation, we use packets and segments interchangeably.

pre-determined time period (particularly $200ms$, in many UNIX implementations). In this case, the receiver sends a DUPACK according to the sequence number of the last properly received packet. In addition, the TCP sender also maintains a timer, namely retransmission time-out (RTO) timer. When a packet is transmitted, the sender resets the RTO timer. If the corresponding ACK (i.e., an ACK packet with the sequence number greater than the sequence number of this data packet) is not returned before this timer expires, the sender determines that the packet has been lost due to severe network congestion and triggers the timeout action (see below). This is often referred to as TCP's coarse timeout.

2.2 TCP Congestion Control in Detail

TCP sender's congestion window grows upon receiving ACKs and decreases when either a pre-determined number (usually 3) of DUPACKs are received, or a RTO expiry has occurred.

There are quite a few different variants of TCP implementation in today's Internet installation. All of the different TCP implementations use the above increase-decrease rules to achieve congestion control, but with slightly different parameters and strategies. Among the many implementations, TCP-Tahoe and TCP-Reno, originally implemented in the BSD 4.0 operating system, are the most representative versions, of which TCP-Reno is considered to be the most widely deployed.

The goal of TCP is to reliably and in-order transport packets from one end-station to the other in such a way that the network capacity is utilized efficiently and competing TCP flows share the capacity fairly. TCP achieves this goal without such helps as virtual circuit establishment, or resource pre-allocation or reservation, from network routers, nor does it rely on any complicated signaling protocols. The design

of the TCP protocol reflects the fundamental principle of Internet architecture [16], that is "keep it simple".

TCP does not assume any congestion control functionality from the IP routers.[2] In TCP, flow control is imposed by the receiver, and congestion control is exercised by the sender. This is because the receiving host inserts its view of the congestion window size, called receiver advertised window size, into the acknowledgment packets. The receiver advertised window size reflects the receiving host's internal available buffer space for packet processing and reassembling. When the sender decides to change the congestion window, it takes the smaller value of the receiver advertised window size and its own intended new window size.

Therefore, TCP's congestion control is often referred to as end-to-end congestion control. To achieve the end-to-end congestion control, TCP adopts the increase-decrease window adjustment rules, which is referred to as the additive-increase and multiplicative-decrease, or AIMD, congestion control algorithm [17, 18, 19]. In the following sub-sections, the essence of TCP's AIMD algorithm is summarized.

2.2.1 Probing Phase

For TCP to achieve high network utilization without prior knowledge of the end-to-end path topology and capacity of each link along the path, or the help from any auxiliary signaling protocol for resource discovery and/or reservation, the TCP's sender has to probe for the available network capacity.

TCP divides its probing phase into two procedures, namely Slow Start and Congestion Avoidance.

[2]From TCP's perspective, IP routers are to store-and-forward IP packets. The routers are assumed to have the simplest queue management, i.e., tail-dropping. In fact, even today, most of the IP routers installed in the global Internet are tail-dropping routers.

Slow Start TCP sender maintains two state variables, the congestion window (*cwnd*) and the slow start threshold (*ssthresh*). Initially, *ssthresh* is set to the maximum integer value allowable by the underlying operating system and processor of the end-station,[3] and *cwnd* is set to 1. Upon receiving of each returning ACK, the sender increases *cwnd* by 1. This is equivalent to doubling of *cwnd* for every RTT, because the *cwnd* roughly limits the number of packets the sender can send in one RTT. This procedure continues until the network capacity is reached and the routers along the path starting to drop packets because of buffer overflow. Loss of packets triggers the TCP receiver to send DUPACKs.

After the sender has received 3 DUPACKs *for the first time*, the initial Slow Start procedure ends. After retransmitting the lost packet, the sender sets the *ssthresh* to $\frac{cwnd}{2}$ and reduces the *cwnd*. The reduction of *cwnd* differs between TCP-Tahoe and TCP-Reno. In Reno, the *cwnd* is reduced to *ssthresh*, which is half of its original value, and the control procedure is switched to the Congestion Avoidance mode. Whereas, in Tahoe, the *cwnd* is reduced to 1, and a second Slow Start procedure starts with the newly adjusted *ssthresh*. The Slow Start procedure ends when $cwnd \geq ssthresh$, and the control is switched to Congestion Avoidance mode.

Congestion Avoidance The TCP sender enters the Congestion Avoidance mode when $cwnd \geq ssthresh$. In this mode TCP continues to probe the available network capacity, but more cautiously. During the Congestion Avoidance, the sender increases *cwnd* by $\frac{1}{cwnd}$ for every returned ACK, which is equivalent to an increment of *cwnd* by 1 for every RTT. Therefore, the window increment while in Congestion Avoidance represents the additive-increase (AI) part of the TCP's AIMD algorithm.

[3]This is the case, at lease, in Linux kernel 2.4 networking protocols implementation.

The AI continues until another packet loss is detected through 3 DUPACKs. At this point, the sender retransmits the lost packet and enters the transmission throttle phase, in which TCP-Tahoe and TCP-Reno, again, behave slightly differently.

2.2.2 Throttle Phase

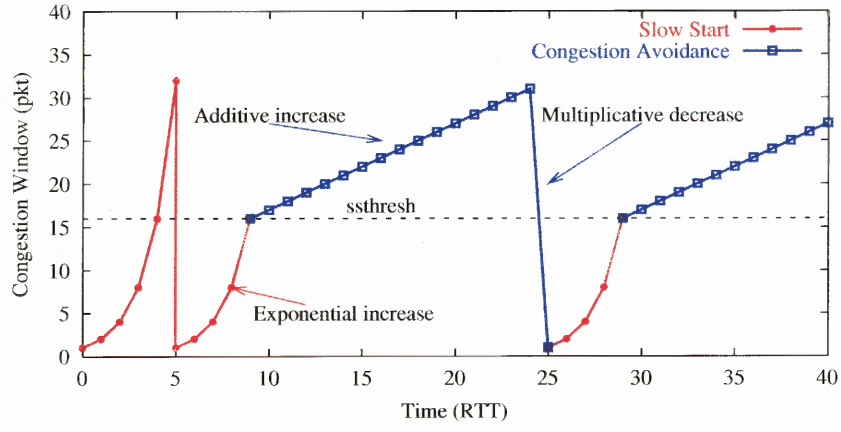
Standard TCP congestion control algorithm regards *all* packet losses as signals of network congestion, and hence enters the transmission throttle phase by reducing the sender's transmission rate in order to alleviate the congestion.

For TCP-Tahoe, the sender sets the $ssthresh$ to half of the current $cwnd$ value, and reduces $cwnd$ to 1; and therefore, the cycle restarts again with Slow Start mode. For TCP-Reno, however, the sender sets both $ssthresh$ and $cwnd$ to half of the current $cwnd$, and thus, drives the cycle to restart with Congestion Avoidance mode (since $cwnd \geq ssthresh$). This handling by Reno is often referred to as Fast Recovery, as compared to Tahoe's restart with window size of 1.[4] The transmission throttle phase accounts for the multiplicative-decrease (MD) part of the congestion control algorithm.

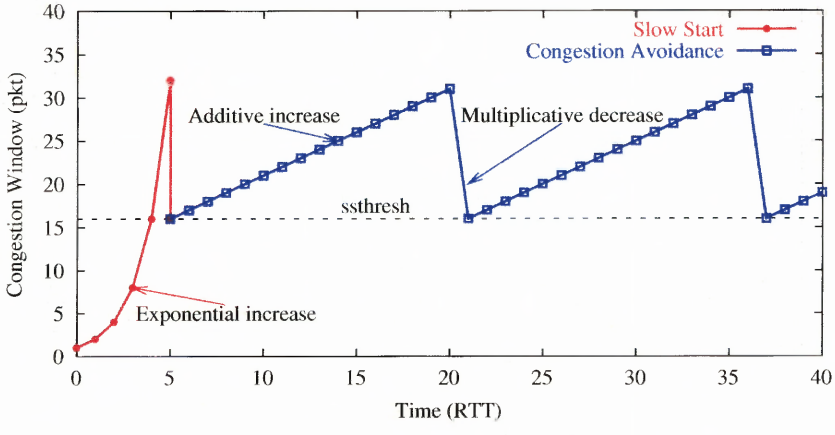
RTO expiry does not occur very often due to TCP sender's dynamic measurement of the end-to-end RTT and RTT variance, and setting the RTO timer according to these measures (see [18, 15, 20] for more details on the TCP's RTT estimation algorithm and implementation). In the event that RTO does expire, both Tahoe and Reno reduce $cwnd$ to 1 and hold the next transmission for an interval of which the length increases exponentially in number of RTO expiries (again, see [15, 20, 21] for details). This is referred to as the exponential back-off.

The following Figure 2.1 illustrates the $cwnd$ evolutions of TCP-Tahoe and TCP-Reno, respectively.

[4]The phrase Fast Retransmit often seen in descriptions and literature of TCP refers to the sender's detection of a packet loss by receiving 3 DUPACKs instead of waiting for the RTO to expire, which was the case in TCP implementations prior to TCP-Tahoe.



(a) Illustrative congestion window evolution of TCP-Tahoe



(b) Illustrative congestion window evolution of TCP-Reno

Figure 2.1 Illustrations of congestion window (*cwnd*) evolutions of TCP-Tahoe and TCP-Reno, showing the typical cyclic sawtooth pattern.

2.2.3 Throughput Analysis

As one would have imagined from the descriptions of the TCP's AIMD congestion control algorithm, the *cwnd* evolves in a repetitive sawtooth pattern. The throughput is approximately proportional to the area covered under the *cwnd* curve, and the bulk of which is accounted for by the Congestion Avoidance mode operations of the connection. In general, Reno can achieve higher throughput than Tahoe, which should be obviously evident from Figure 2.1 by graphic approximation.

Considering the case of TCP-Reno, if we ignore the Slow Start and RTO timeout events, and focus on the dominant Congestion Avoidance modes, and let $\{w_k\}$ be the sequence of the sender's congestion window, p be the probability of packet loss, we can express the TCP's window dynamics as

$$w_{k+1} = \begin{cases} w_k + \frac{1}{w_k}, & \text{with probability } 1-p \\ \frac{w_k}{2}, & \text{with probability } p \end{cases}. \tag{2.1}$$

Denote $\overline{w}$ as the expected value $E\{w_k\}$, i.e., the average window size, and approximate $E\{1/x\}$ by $1/E\{x\}$. Taking the expectation on both sides of (2.1), and after simple algebraic rearrangement, we have

$$\overline{w} = \sqrt{\frac{2(1-p)}{p}}. \tag{2.2}$$

Since, on average, TCP only sends $\overline{w}$ packets in 1 RTT, the average throughput of TCP, in packets per second, can be expressed as a function of p, the packet loss probability, and $\overline{R}$, the average round-trip time as $B(p) = \overline{w}/\overline{R}$. Assuming a small p, we then have

$$B(p) \approx \sqrt{\frac{2}{p\overline{R}^2}}. \tag{2.3}$$

This recovers the well known "square-root" formula [22, 23], with difference only in the constant term, which indicates that TCP throughput decreases as packet loss probability increases if all packet losses are treated as signals of congestion.

More detailed studies can be found, for example, in [21, 24]. In [24], it is shown that TCP throughput starts to deteriorate sharply when $p(\mu T)^2$ is less than 10, where (μT) is the bandwidth-delay product of the TCP connection. It implies that the AIMD algorithm as described in this chapter will see sharp throughput degradation in high bandwidth-delay product networks even with a very small packet loss probability.

In [21], a detailed model of TCP-Reno's congestion control, including Slow Start and RTO expiries, is studied and a closed form expression of TCP-Reno's throughput is given, and validated through simulations, as follows

$$B(p) \approx min\left\{\frac{w_m}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 min\left\{1, 3\sqrt{\frac{3bp}{8}}\right\} p(1+32p^2)}\right\}, \tag{2.4}$$

where w_m is the maximum value the TCP sender can have for $cwnd$, b is 1 or 2 depending on whether the receiver sends an ACK for every packet or for every 2 packets[5] delivered, respectively, and T_0 is the RTO timer value.

In essence, since standard TCP congestion control algorithm treats all packet losses as signals of network congestion, and accordingly reduces its transmission rate multiplicatively, the packet loss probability p plays a vital role in the attainable TCP throughput, and hence, the protocol efficiency and network utilization.

[5]This is referred to as delayed acknowledgment.

2.3 Summary

This chapter presents an overview of the operation and the algorithms involved in TCP congestion control. It provides the knowledge basis for the discussions and analysis in subsequent chapters.

CHAPTER 3

TCP PERFORMANCE IN WIRELESS NETWORKS: PROBLEMS AND SOLUTIONS

In this chapter, we present our motivation behind this dissertation research. In Section 3.1, we briefly summarize the performance degradation as observed in applying standard TCP over wireless IP networks. Section 3.2 discusses the current state-of-the-art solutions toward the problem. In the following Chapter 4, we presents our philosophy in designing an end-to-end solution to TCP performance enhancement for wireless IP communications by outlining the basic framework of the proposed solution architecture, namely adaptive congestion control through achievable rate estimation, and effective loss type differentiation through explicit packet marking by active queue management.

3.1 TCP Performance Degradation in Wireless IP Communications

With the advances of wireless technologies and the ever-increasing user demands, IP protocols suite has to extend its horizon to encompass the wireless communication arena. In reality, the future all-IP data networks would most likely be heterogeneous networks, meaning the communication path from one end to another will consist of both wired and wireless links.

However, the TCP protocol we have enjoyed and relied upon in the wired networks exhibits its weaknesses in such hybrid environments. The problems stem from the unique characteristics of the wireless links as compared to the wired links and the current TCP protocol's design assumption of the packet loss model. The problems manifest in various applications as degradation of throughput, inefficiency in network resource utilization, and excessive interruption of data transmissions.

Unlike the fiber optical backbone and copper wired access networks, wireless links use the open air as the transmission medium and are subject to many uncontrollable quality-affecting factors such as weather conditions, urban obstacles, multi-path interferences, large moving objects and mobility of wireless end devices, etc. As a result, wireless links exhibit much higher bit error rate (BER) than the wired links. Also, limitations of the radio coverage and user mobility require frequent hand-off processes, which result in temporal disconnections and re-connections between the communicating end points during a communication session. Packet loss in wireless environments tends to be bursty as well, or, in other words, packet losses can be correlated.

In this section, we analyze the performance problem exhibited when the unmodified standard TCP protocol is used over networks consisting of wireless links.

The standard TCP protocol as we described before cannot deal with the high BER and frequent disconnections effectively. Since all packet losses are inferred as the result of network congestion in the standard TCP, random packet losses caused by the high BER of the wireless link would mistakenly trigger the TCP sender to reduce its sending rate unnecessarily. The Fast Retransmit and Fast Recover algorithms introduced by TCP Reno can recover the sporadic random packet losses fairly quickly if such losses only occur once within a round trip time (RTT). However, lightning and shot noises in the wireless environment usually cause the random bit errors to occur in short bursts, thus leading to a higher probability of two or more random packet losses within one RTT. The packet loss during the retransmission will cause the TCP sender to exponentially back off its retransmission timer at a typical granularity of 500ms. Therefore, two random packet losses within a RTT leads to the TCP sender to stall its transmission for a period of about 1 second. The inability of the standard TCP protocol in distinguishing the congestive packet

losses from the random packet losses contributes to a drastic decrease of TCP's throughput.

Figure 3.1 and Figure 3.2 illustrate this throughput degradation when a bulk data transfer using TCP Reno is simulated on a 2Mbps wireless link with the RTT equal to 20ms. The simulations are implemented using the NS-2 [25] network simulator. A simple one-hop network topology is used in the simulations, where the sender transmits a large file via FTP to the receiver. We ran the simulation once for each of the two flows. When Flow 1 was simulated, we instructed the simulator to drop two packets, one drop at the 10th RTT and the other at the 50th RTT. As seen from the figure, which shows the congestion window ($cwnd$) growth with time, the TCP's Fast Retransmit and Fast Recover algorithms recover the single packet drop within one RTT fairly well.

For Flow 2, the simulator was instructed to drop two packets within a very short interval around the 10th RTT. In fact, the second dropped packet is the retransmitted packet of the first lost packet. This caused the TCP to stall its transmission for about one second, from the 10th RTT till the 60th RTT. High BER in wireless channels makes it more likely that multiple packet corruptions and drops occur within a short time interval.

Figure 3.2 plots the dynamics of the packet sequence number of the two flows in the same simulations. It is seen that for Flow 1, where the two packet drops were far apart, the flow transported about 400 packets in 100 RTT time, whereas Flow 2 which has experienced two packet drops within one RTT only managed to transport about 200 packets in 100 RTT time, which is 50% decrease of the throughput.

Hand-off is an inevitable process in mobile wireless communications due to the limited radio coverage of the wireless devices, as well as the user terminal mobility.

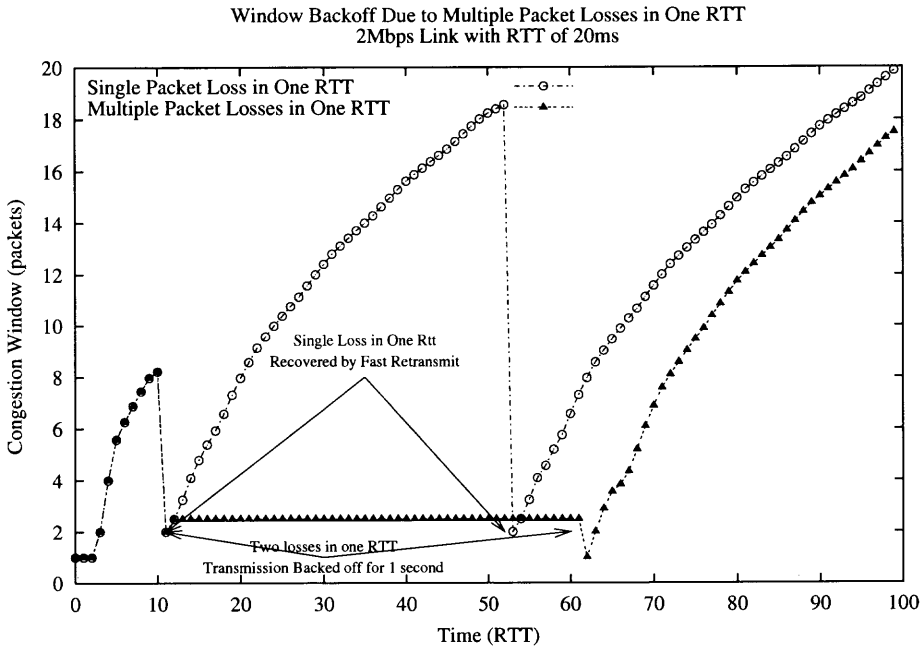


Figure 3.1 Stall of transmission because of the window back off due to multiple packet losses within one RTT.

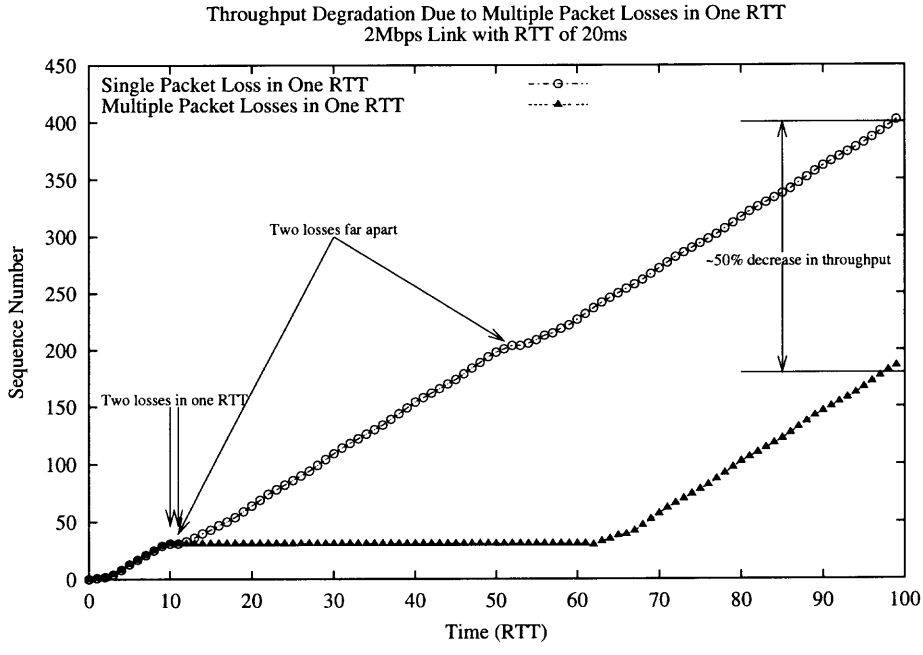


Figure 3.2 Large throughput decrease as the result of multiple packet losses within one RTT.

In digital cellular systems such as GSM, the hand-off typically requires 1 or 2 seconds [26]. Although the CDMA spread spectrum cellular systems provides the unique soft hand-off capability where there is no physical channel changes occur during the hand-off process, which could speed up the process, it is generally reasonable to assume that the hand-off process would cause up to a couple of tens of milliseconds of temporary disconnection of the link. In Figure 3.3 and Figure 3.4, transmission back-off and consequently the loss of throughput by standard TCP are shown for the case where a short link disconnection occurs during the data transfer session.

Fading can also cause a period of silence on the channel, during which neither the sender nor the receiver can hear each other. Fading is often caused by the movement of the mobile nodes, or by large objects, such as a truck or a subway train, that move in front of or around the mobile nodes.

In the wireless Ad Hoc network, because of the infrastructure-less and high node mobility nature of the network, frequent route changes and network partitions also present a great challenge to TCP throughput.

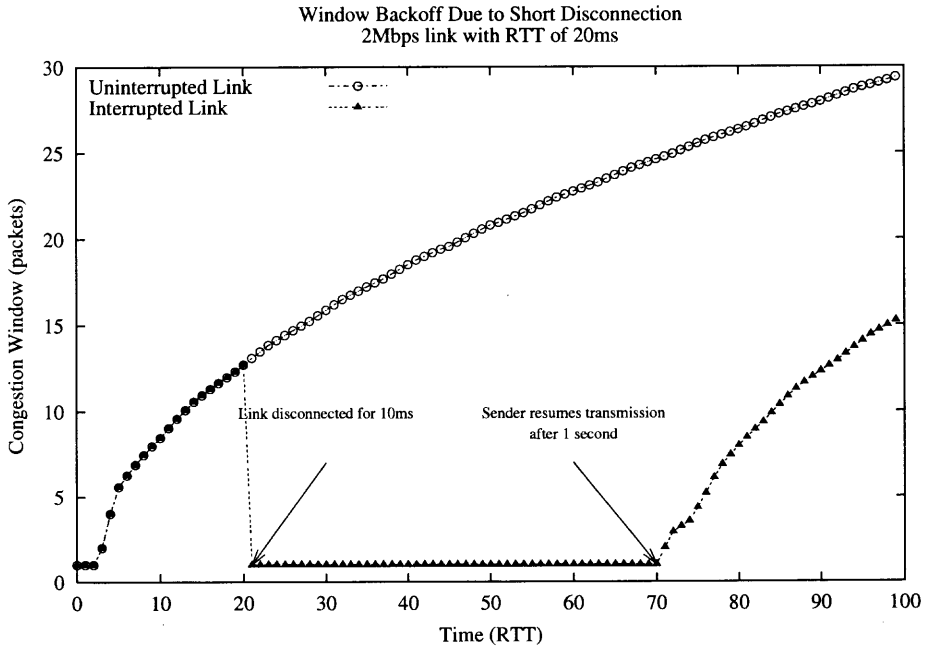


Figure 3.3 Transmission back off due to short link disconnection. The wireless link is temporarily disconnected for 10 ms at the 20th RTT time for the flow represented by the solid triangle dots. The disconnection causes the TCP sender's retransmission timeout to occur and the sender's transmission to back off for about one second before it is resumed at the 70th RTT time.

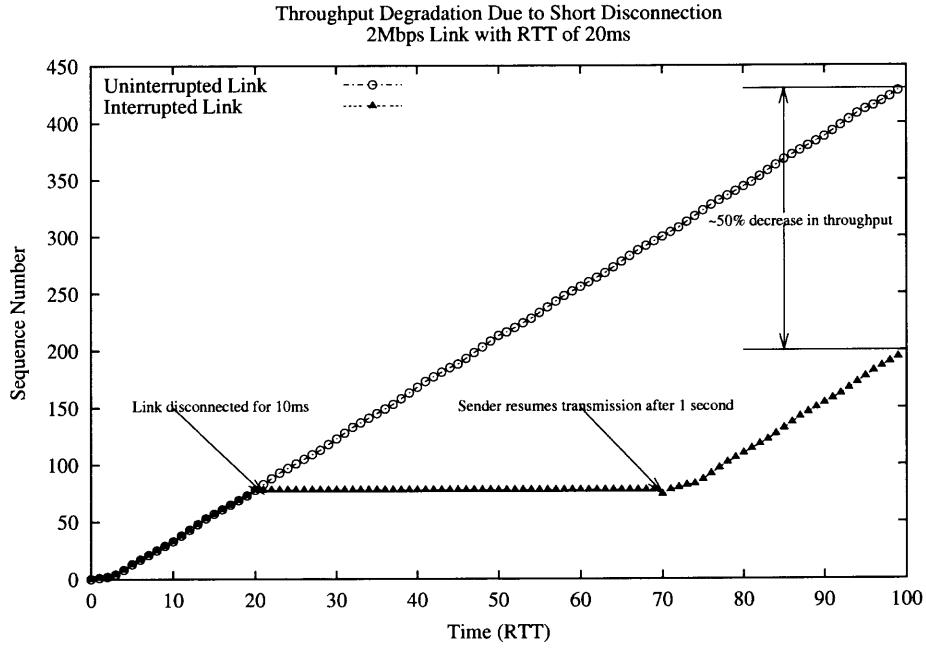


Figure 3.4 Throughput degradation due to short link disconnection. The sequence number of the packets of the flow that experiences the short link disconnection stops to grow between the 20th RTT time and the 70th RTT time, and therefore, results in a 50% of decrease in its throughput.

3.2 Current Solutions to TCP Performance Enhancement for Wireless IP Communications

In this section, we review the current state-of-the-art solutions to the problem of TCP performance degradation in wireless IP communications.

We can divide the currently proposed solutions to TCP performance enhancement over wireless networks into three categories, namely, the split mode approach, the link layer approach, and the end-to-end approach.

The split mode approach, such as the I-TCP (Indirect TCP) by Bakre and Badrinath [27], the M-TCP by Brown and Singh [28], and the Mobile-End Transport Protocol by Wang and Tripathi [29], tries to shield the wireless network from the wired network. TCP connections of the wired and wireless networks are separated. The intermediate node, usually the base station, or wireless router, sets up the connection with the fixed host and the mobile host, and is responsible to recover packet losses caused by wireless link errors. This approach requires large buffers at base stations, and the end-to-end TCP's semantics are sometimes not preserved.

The link layer approach, such as the Snoop by Balakrishnan [30], the SMART retransmission by Keshav [31], the WTCP by Ratnam [32], the transport unaware link improvement protocol (TULIP) by Parsa [7], and the explicit loss notification by Balakrishnan [33], rectifies wireless link errors at the second layer. Techniques such as forward error correction (FEC), automatic repeat request (ARQ), and explicit loss notification (ELN) have been proposed for this class of solutions. However, such approach requires protocols at different layers to interact and coordinate closely, thus increasing the complexity of protocol implementation.

In the end-to-end approach, TCP senders and receivers are responsible for the proper classification of loss types and appropriate actions; hence, the end-to-end semantics are preserved. TCP Peach by Akyildiz et al. [34] is particularly designed for the satellite communication environment, where large bandwidth-delay product

is the nature. Among the four phases of TCP, TCP-Peach leaves the Congestion Avoidance and the Fast Retransmit intact. It replaces the Slow Start and the Fast Recovery with algorithms called Sudden Start and Rapid Recovery, respectively. In the Sudden Start and the Rapid Recovery, the sender probes the available network bandwidth in only one RTT with the help of low-priority dummy packets. An important assumption made by TCP Peach is that the routers must support priority queuing. In TCP-Peach Plus by the same group [35], the actual data packets with lower priority replace the low priority dummy packets as the probing packets to further improve the throughput.

TCP-Westwood by Casetti et al. [36] is a rate estimation based additive-increase/adaptive-decrease congestion control algorithm, in which the sender estimates the available network bandwidth dynamically by measuring and averaging the rate of returning ACKs. TCP-Westwood achieves higher performance in terms of throughput in wireless networks than the traditional TCP variants, such as Tahoe and Reno, in an implicit way. Although TCP-Westwood does not explicitly differentiate the packet losses caused by wireless link errors from those caused by network congestion, it treats all packet losses as signals of congestion and reduces its transmission rate to the estimated available bandwidth, by which the packet losses caused by wireless link errors would have less drastic impact on the overall throughput as compared to the standard halving the transmission rate on every lost packet. The end-to-end approach maintains the network layer structure and requires minimum modification at end hosts and router.

Standard TCP schemes may suffer from a severe degradation in performance over the wireless links. Modifications to the standard TCP protocol to remedy its deficiency in wireless IP communications have been a very active research area. Many schemes have been proposed.

3.2.1 TCP for Different Wireless Applications

From the applications' point of view, there are different wireless environments, and wireless TCP can be specifically designed to accommodate their needs. The most common wireless networks include satellite networks, ad hoc networks, and general wireless platforms such as wireless LANs (WLAN) and cellular systems. The design of wireless TCP considers the characteristics of a particular type of wireless networks and its needs; for instance, the satellite network has long propagation delay, and the ad hoc network is infrastructure-less.

Nevertheless, an obstacle that all wireless networks have to face is high BER. In heterogeneous networks, to explicitly differentiate the cause of packet losses becomes the primary goal of efficient TCP design. Such approaches aim at finding an explicit way to inform the sender of the cause of the packet drop, due to either congestion or random transmission errors. Hence, the sender is expected to make appropriate decisions on how to control the congestion window. The standard Reno scheme halves its window size when experiencing a packet loss regardless of the cause. If the loss is due to network congestion, this behavior alleviates network congestion. However, it is not efficient in the case of non-congestive random loss.

Satellite Network TCP schemes for satellite networks are designed based on the observation that it takes the sender of a TCP connection a fairly long interval to reach a high sending rate during the traditional Slow Start phase. While many TCP applications such as HTTP are based on the transfer of small files, with the long propagation delay of the satellite link, it can happen that the entire transfer occurs within the Slow Start phase and the connection is not able to fully utilize the available network bandwidth. TCP-Peach [34] employs two new mechanisms, the Sudden Start and Rapid Recover, in combination with the traditional TCP algorithms, namely the Congestion Avoidance and Fast Retransmit, to cope with

the TCP performance degradation over satellite communication links characterized by large propagation delay and high link error rate. Sudden Start introduces the use of dummy packets that are copies of the last data packet the sender has sent. The sender sends multiple dummy packets between two consecutive data packets; the reception of ACKs for the dummy packets indicates the availability of the unused network resource and the sender is triggered to increase its sending window quickly. Dummy packets are labeled with low priority; therefore, a router would drop the dummy packets first in the event of congestion. Simulations show that the Sudden Start algorithm can reach the available bandwidth within 2 RTTs whereas the traditional Slow Start in TCP-Reno takes about 7 RTTs to reach the same transmission rate. Rapid Recover replaces the classical Fast Recover mechanism in an effort to improve the throughput in the presence of high link error rate. In the Rapid Recover phase, instead of trying to distinguish congestive loss from error loss, the sender uses the lower priority dummy packets to interleave the data packets and inflates the sending window size upon the reception of ACKs for the dummy packets.

Ad Hoc Network In an Ad Hoc networks, high error rate and frequent route changes and partitions are typical. Both characteristics result in packet loss besides network congestion, and hence, should be treated differently. ATCP [37] is designed as an end-to-end solution to improve TCP throughput for such an environment. It is implemented as a thin layer inserted between the standard TCP and IP layers. It relies on the Explicit Congestion Notification (ECN) to detect congestion and to distinguish congestion loss from error loss, and the ICMP Destination Unreachable message to detect the change of route or temporary partition of the Ad Hoc network. According to these feedbacks from the network, the ATCP layer puts the TCP sender into either the persist state, congestion control state, or retransmit state accordingly.

When the packet loss is due to the high BER, ATCP retransmits the lost packet so that TCP does not perform the congestion control; if the ICMP message indicates a route change or network partition, ATCP puts the TCP sender into the persist state awaiting for the route to reconnect. In case of packet reordering, ATCP reorders the packets so that TCP would not generate duplicate ACKs. When ECN indicates a real congestion, ATCP enters the normal congestion control stage. By inserting the ATCP layer between the TCP and IP layers, the scheme does not modify the TCP code; in addition, since ATCP does not generate ACK packets by itself, the end-to-end TCP semantic is maintained.

Cellular Network For cellular networks, where the base station connects a fast fixed network and a slow mobile network, the modifications on TCP algorithms focus on cellular characteristics such as hand-off, and the commonly shared problem for all wireless networks, high BER. Freeze-TCP [38] is another end-to-end solution to improve the TCP performance in the mobile environment. It imposes no restrictions on the routers and only requires code modifications at the mobile unit or the receiver side. It addresses the throughput degradation caused by frequent disconnections (and re-connections) due to mobile hand-off or temporary blockage of the radio signals by obstacles. The assumption is that the mobile unit has the knowledge of radio signal strength, and therefore can predict the impending disconnections. The Freeze-TCP receiver on the mobile unit proactively sets the advertised window size to zero in the ACK packets in the presence of impending disconnection. Because the TCP sender selects the window size to be the minimum of its vision of the window size and the receiver's advertised window size, this zero window size ACK packet would force the sender into the persist mode where it ceases sending more packets while keeping its sending window unchanged. To prevent the sender from exponential backing off, when it detects the re-connection,

the receiver sends several positive ACK packets to the sender acknowledging the last received packet before the disconnection so that the transfer can resume quickly at the rate before the disconnection has occurred. To implement the scheme proposed in [8], cross-layer information must be exchanged, and the TCP layer protocol must be exposed to some details of the roaming and hand-off algorithms implemented by the NIC vendors on the interface devices.

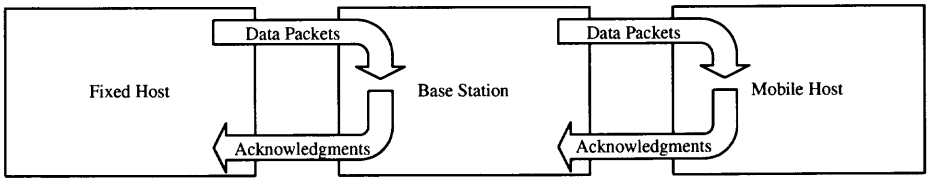
3.2.2 Implementation of Wireless TCP

From the implementation point of view, wireless TCP algorithms can be designed in either split-mode or end-to-end manner. Due to the significant difference in characteristics between wireless and wired links, the split mode divides the TCP connection into wireless and wired portions, and the acknowledgments are generated for both portions separately by the wireless-aware routers or base stations. By doing so, the performance on the wired portion is least affected by the relatively unreliable wireless portion.

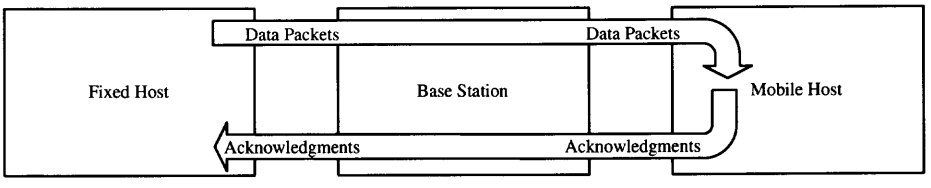
Whereas, the end-to-end mode treats the route from the sender to the receiver as an end-to-end path and the sender is acknowledged directly by the receiver. This maintains the end-to-end semantic of the original TCP design.

Split Mode Implementation The wired portion of the heterogeneous network is more reliable than the wireless portion in terms of link capacity and error rate; however, the transmission would be bottlenecked at the slow and lossy wireless link. The split mode attempts to shield the wireless portion from the fixed network by separating the flow control at the intermediate router (or a base station for a cellular network), so that the wireless behavior has the least impact on the fixed network. The intermediate router behaves as a terminator/re-generator for both the fixed and wireless portions. Both end hosts communicate with the intermediate

router independently without the knowledge of the other end. The intermediate router is reinforced with functionalities to coordinate the transaction between the two network portions. This is illustrated in Figure 3.5(a). As an example, in I-TCP (Indirect TCP) [27], the MSR (mobile support router) connects the MH (mobile host) to the FH (fixed host), and establishes two separated TCP connections with the FH and the MH, respectively. The MSR communicates with the FH on behalf of the MH. The congestion window is maintained separately for the wireless and fixed connections. When the MH switches cells, a new MSR takes over the communication with the FH seamlessly. Therefore, the FH is shielded from the unreliable feature of the wireless connections.



(a) A split mode TCP connection.



(b) An end-to-end TCP connection.

Figure 3.5 An split mode TCP connection and an end-to-end TCP connection in communications involving wireless links.

End-to-End Implementation In the split mode, an intermediate router has to retrieve the information in the TCP packet and process related data before it reaches the destination, thus violating the end-to-end semantic of the original TCP. Split mode approach also faces challenges of scalability. Because the router or base station must keep the state information of every TCP connection that passes through it, buffer space and processing overhead increase as the number of

connections increases. Also, when the end-to-end route changes during a session, the corresponding flow state information must be transported to the new responsive router or base station.

In the end-to-end approach, only the end hosts participate in the flow control. The receiver provides the feedbacks reflecting the network condition, and the sender makes decisions for the flow control. This is illustrated in Figure 3.5. In the end-to-end approach, the ability of accurately probing the available bandwidth is the key for a better performance, which is still a great challenge. The end-to-end approach can have its flow control manipulated in two manners, reactively and proactively, respectively. In the reactive manner, the sender rectifies the flow control when the network situation becomes marginal or has crossed a threshold. In the proactive manner, the feedbacks from the network guide the senders to reallocate the network resource in order to prevent congestion.

Reactive Congestion Control The standard Reno scheme employs reactive flow control. The congestion window is adjusted based on the collective feedbacks of ACKs and DUPACKs generated at the receiver. TCP probes the available bandwidth by continuously increasing the congestion window gradually until the network reaches the congestion state. In this regard, congestion is inevitable. TCP will then fall back to a stage of a much slower transmission rate, which may be unnecessary for wireless random loss. Many TCP schemes have been proposed as an enhancement to the standard Reno scheme in this reactive manner. The Fast Recovery algorithm of Reno takes care of a single packet drop within one window. After one lost packet is recovered, Reno terminates the Fast Recovery mechanism. Due to the nature of wireless networks, correlated errors may induce multiple packet loss in a short time period. Therefore, the Reno scheme would be

forced to invoke Fast Recovery back and forth and slows down the recovery of the lost packet.

New Reno [39] modifies the Fast Recovery mechanism of Reno to cope with multiple losses from a single window; this is one of the characteristics of wireless networks where fading channel may cause contiguous packet loss. In New Reno, the Fast Recovery mechanism does not terminate until multiple losses, indicated by the reception of *partial acknowledgments*, from one window are all recovered. The limitation of New Reno is, however, that it cannot distinguish the cause of the packet loss, and thus a more effective Fast Recovery algorithm cannot be implemented.

TCP SACK [40] is a selective acknowledgment option for TCP, targeting at the same problem that Reno tries to tackle. While the feedbacks of Reno and New Reno are based on the cumulative acknowledgments, SACK employs the selective repeat retransmission policy. It indicates a block of data that have been successfully received and queued at the receiver when packet loss occurs instead of sending a partial acknowledgment as in New Reno. Hence, the sender has better knowledge about the exact number of packets that have been lost rather than the limited knowledge about the loss at the left edge of the window only as in the standard cumulative ACK scheme. SACK requires the modification at the sender side as well as the receiver side. SACK blocks are encoded in the TCP option field, which inherently limits the number of SACK blocks that one ACK can carry. Moreover, as in Reno, SACK reactively responds to packet losses, and therefore has limited ability of active congestion avoidance.

Proactive Congestion Control In the proactive flow control, the sender attempts to adjust the congestion window proactively to an optimal rate according to the information collected via feedbacks, which can be translated into as an indication of the network condition. By doing so, the sender reacts intelligently to the network

condition or the cause of the packet drop, due to either congestion or random errors, and therefore prevents the network from entering the undesirable state, e.g., congestion or unnecessary decrease of the congestion window. Different strategies can be employed in the design to provide the sender with the explicit network condition.

TCP Vegas [41] estimates the backlogged packets in the buffer of the bottleneck link. Vegas selects the minimal RTT as a reference to derive the optimal throughput the network can accommodate. It also records the actual sending rate at the sender during the transmission to derive the actual throughput. The difference between the optimal throughput and the actual sending rate can be used to infer the amount of backlogged data in the network. It then sets two thresholds corresponding to two network stages, one of which represents too few backlogged data and the other of which represents too many backlogged data. For the former stage, Vegas increases the congestion window linearly. For the latter stage, Vegas decreases the congestion window linearly. By doing so, Vegas tries to stabilize the network congestion state around the optimal point by proactively adjusting the congestion window without a dramatic change in the congestion window.

TCP Veno [42] adopts the same methodology as Vegas to estimate the backlogged packets in the network. It further suggests a way to differentiate the cause of packet loss. If the number of backlogged packets is below a threshold, the loss is considered to be random. Otherwise, the loss is said to be congestive. Suppose the loss is congestive, Veno adopts the standard Reno scheme. For the loss due to a random error, it increases the congestion window in a conservative manner, i.e., sending one packet upon receiving every other ACK.

TCP Westwood [36] is a rate estimation based end-to-end approach, in which the sender estimates the *available network bandwidth* dynamically by measuring and averaging the rate of returning ACKs. Under the assumption of a perfect reverse

path, the inter-ACK gap reflects the available network resource. TCP Westwood deploys an available bandwidth measurement module at the sender side based on the interval of the returning ACKs. It calculates the explicit available bandwidth and uses it to guide the sending rate. When TCP Westwood determines that the link is congested after having received 3 DUPACKs, it sets the slow start threshold, $ssthresh$, to reflect its estimated bandwidth delay product. TCP Westwood claims improved performance over TCP Reno and SACK while achieving fairness and friendliness.

3.3 Summary

In this chapter, we have presented the problem of throughput degradation when unmodified TCP is used in error-prone wireless networks. We have also presented a comprehensive review of currently proposed solutions.

CHAPTER 4

FRAMEWORK OF PROPOSED SOLUTION TO TCP PERFORMANCE ENHANCEMENT FOR WIRELESS IP COMMUNICATIONS

In this chapter, we present the framework of our proposed approach to remedy the TCP performance degradation problem in wireless IP communications.

First, we propose a modification to the standard TCP protocol instead of a replacement of the standard TCP with a specialized wireless IP transport protocol. Our solution should be able to co-exist with the standard TCP protocols and operate in a fair and friendly way in terms of network resource sharing. Second, we limit our design space to an end-to-end solution. Specifically, the TCP's end-to-end send-and-acknowledge semantics is to be preserved for better scalability, as opposed to the split mode approach. Third, our solution does not adopt the cross-layer design which requires interaction and coordination between layers lower than the IP and the transport layer.

The proposed solution includes two integral components, the congestion control part and the loss differentiation part. We believe that the standard TCP's drop-to-half window adjustment upon a packet loss is too dramatic and heuristic. Such a static reaction to congestion hinders the TCP from reaching high efficiency in terms of throughput and link utilization, particularly in high-speed networks. And, when in the wireless networks where packet losses may be caused by reasons other than congestion, this static rate halving congestion control further degrades the performance. Therefore, even without the proper loss differentiation, i.e., the sender always reacts to a packet loss as if it was caused by congestion, by setting the level, to which the congestion window is reduced upon a packet loss, dynamically according to the network condition would improve the TCP's performance.

4.1 Adaptive Congestion Control via Achievable Rate Estimation

The first question we ask is what this proper dynamic level would be, and how the TCP end stations[1] compute this proper dynamic level.

Our answer to this question is that the proper dynamic level where the congestion window should be reduced to in case of congestion should reflect the connection's achievable rate, i.e., the rate at which the connection can attain *without* causing congestion. This achievable rate can be estimated by either the sender or the receiver by examining the inter-arrival times of the packets. We give a simple justification to this idea as follows.

Consider a simple network consisting of one link with capacity of μ packets per second, an end-to-end round-trip delay of T seconds, and a buffer of B packets. One TCP connection is established across this network. As analyzed in [24], the maximum congestion window[2] W_{max} (in packets) that the TCP's sender can have without causing congestion is, according to the Little's Law [43],

$$W_{max} = \mu T + B.$$

Therefore, a sensible value of the congestion window that the sender should reduce to when congestion occurs (and, hence, packet losses are experienced) is μT_{min}, instead of blindly $\frac{W_{max}}{2}$, where T_{min} is the round-trip propagation delay, and $T_{min} \leq T$. Although the TCP sender does not know the actual link capacity μ, both T_{min} and μ can be estimated from the inter-arrival times of packets.

In general cases where there are multiple TCP connections traversing the same network, each connection i would have its maximum attainable congestion window

[1]We are interested in the end-to-end solution and we restrict our design space where there is only minimum support, in terms of feedback information on network conditions, from the routers. Thus, computation of the proper window drop level has to be done at the end stations.

[2]Recall the sender's congestion window, or *cwnd*, is the number of unacknowledged packets allowed in the network.

$$W_{max,i} = (\alpha_i \mu)T + \beta_i B,$$

where $\sum_i \alpha_i = 1$, and $\sum_i \beta_i = 1$. In the above, $\alpha_i \mu$ is the share of the total link capacity that connection i attained[3], and β_i reflects the connection's share of the bottleneck buffer space when packets have to be queued in the buffer. We call this as the connection's *achievable rate*, and the estimated value of this as the connection's *achievable rate estimate*, or ARE.

4.2 Loss Differentiation Through Active Queue Management

The second question is how the end stations differentiate the packet loss types, i.e., losses due to congestion vs. losses attributed to wireless link errors. The reason that loss differentiation helps improving the TCP performance is obvious, i.e., TCP's congestion window (henceforth, the transmission rate) would not be unnecessarily reduced if a packet loss could be classified as a loss due to wireless link errors.

Being an end-to-end solution to TCP performance degradation in wireless IP communications, we eliminate, from our solution, the problems of scalability in the split mode approach, and cross-layer complexity in the link-layer approach, but impose resource limitations from which the end stations can draw to distinguish different loss types.

End-to-end loss type diagnose or differentiation has attracted many research activities. Biaz and Vaidya in [44] studied the ability of three loss discriminators based on congestion predictors: Jain's delay-based congestion predictor [45], Wang's throughput-based predictor [46], and Vegas predictor [41]. These congestion predictors rely on round trip time and/or throughput changes in

[3]From the perspective of network bandwidth allocation, this is the portion of the total bandwidth allocated to the connection through the TCP protocol itself as well as the particular router queue management algorithm, if any.

response to congestion window size changes to infer the level of congestion on a path. Therefore, the cause of a detected packet loss is inferred indirectly. However, these discriminators were shown to perform as poorly as a random coin tossing predictor. Samaraweea in [47] proposed a non-congestion packet loss detection (NCPLD) technique, which is based on Jain's predictor, to enable TCP to distinguish congestion losses from random wireless error losses. Kim et al. proposed a technique named LIMD/H in [48] that uses the history of packet loss and the evolution of transmission rates to infer the cause of the loss. This is similar to Wang's predictor. TCP Santa Cruz proposed by Parsa and Garcia-Luna-Aceves in [49] updates a 3-state machine based on round trip time changes in response to congestion window changes; it is quite similar to the Vegas predictor, except that TCP Santa Cruz computes the directional congestion as opposed to Vegas' round trip congestion level.

Cen et al. proposed a hybrid loss discriminator named ZBS in [9] which is a combination of three previously proposed discriminators: ZigZag by Cen et al. [50], Biaz by Biaz and Vaidya in [51], and Spike by Tobe et al. in [52]. ZigZag and Biaz are both based on the inter-arrival times of packets at the receiver and the number of losses detected. Spike uses the relative one-way trip time (sender to receiver). ZBS is applied at the receiver. The ZBS discriminator switches dynamically between the three loss discriminators according to the observed network conditions. The accuracy of the ZBS discriminator depends on the number of flows sharing the bottleneck. As the number of competing flows increases so does the congestion loss misclassification as well as the wireless loss misclassification. The wireless loss misclassification exceeds 50% when the number of flows reaches 4. Studies showed that in ZBS, a low congestion loss misclassification is obtained at the expense of a high wireless loss misclassification. TCP Veno by Fu [42] uses the similar technique as in Vegas to estimate the backlog at the bottleneck and uses the result to infer the

cause of a loss. Jitter-based TCP in [10] measures the changes of the end-to-end delay jitter, and uses the measurement in loss type inference.

Liu et al. proposed an approach in [11] that integrates two techniques, namely, loss pairs (LP) and Hidden Markov Model (HMM). LP is based on the source sending two packets in a back-to-back fashion. When one packet of the pair is lost, the round trip time measured for the second packet of the pair *probably* bears *some* information about the cause of the loss. An HMM is then trained over the observed RTTs to infer the cause of the loss. The accuracy of LP-HMM in classifying the losses depends on the location of the wireless link relative to the bottleneck link. Misclassification of congestion losses could reach 64% under heavy traffic when the wireless link happens to be the bottleneck. This result makes the applicability of the approach problematic in many situations such as wireless LAN (WLAN) where the wireless link is usually the bottleneck. Misclassification of wireless losses could reach 79% under moderate traffic when the wireless link is at the downstream of the bottleneck.

In [53], Biaz and Vaidya proposed a loss discriminating technique that is based on a type of AQM called *biased queue management*. The rationale of this biased queue management is that when the link is congested and the router has to drop some packet, it can select, based on a pre-defined preference, as to which packet to drop by examining the special marks (or the lack of such) placed by the TCP sender where the packet is originated. On the other hand, the wireless error process does not have such preferential dropping capability; when the channel is in the error state, all packets have the same chance of being corrupted. The loss discriminator implemented in [53] is called Casablanca discriminator, and the modified TCP is named TCP-Casablanca. The sender of TCP-Casablanca marks the departing packets according to a pre-determined static rule (e.g., one in every k packets), which is also known by the receiver. The Casablanca discriminator operates at the receiver

diagnoses the cause of a packet loss by examining the pattern exhibited in the marks of the lost packets. If the pattern of loss seems random, then it infers the loss is caused by a non-biased process and thus classifies it as a wireless loss. Specifically, the Casablanca loss discriminator employs a discriminating function

$$F(x, r, k) = 1 - \lfloor k\frac{x}{r} \rfloor,$$

where k is the marking rate at the source (i.e., marking one for every k packets), x is the number of lost packets that are marked, and r is the total number lost packet within the observation period. A smaller value of F renders a higher likelihood of congestion losses. There are problems associated with the Casablanca scheme. First, F assumes a uniform distribution of wireless loss events. This is not true since TCP traffic is busty in nature, and so is the observed losses. Second, the accuracy of the discriminator function depends on the sample size, and therefore, accurate loss classification may not be possible in a timely manner. Per packet loss classification is difficult. Third, special routers with priority queuing must be deployed.

Unlike these previously proposed end-to-end packet loss type differentiation techniques that indirectly infer the loss type from measured delay and/or delay variation, our approach directly identifies the packet losses that are caused by congestion, and henceforth, those caused by wireless link errors. We achieve this loss differentiation by a novel application of *active queue management*.

Active queue management (AQM) is itself a very active research area. It has found applications in congestion control, quality of service (QoS), fair resource allocation, and differentiated services (DiffServ). Most routers in today's Internet are simple tail-drop routers. That is, incoming packets are dropped by the router if the router's buffer is full, otherwise enqueued for further delivery.

Random early detection (RED) [54] was proposed to combat the congestion collapse observed in tail-drop routers. A RED router informs the sender of incipient

congestion by probabilistically dropping packets before the buffer overflows. Two thresholds, namely, min_{th} and max_{th}, are maintained, and the average queue length, avg is computed through the exponentially weighted moving averaging (EWMA) of the instantaneous queue length by the router at every arrival of a packet. A RED router drops an incoming packet probabilistically if avg lies between the two thresholds. The dropping probability of an incoming packet is linear in avg. In particular, the dropping probability p_{drop} is expressed as

$$p_{drop} = \begin{cases} 0 & \text{if } avg < min_{th} \\ p_{max}\frac{avg - min_{th}}{max_{th} - min_{th}} & \text{if } min_{th} \leq avg < max_{th} \\ 1 & \text{if } avg \geq max_{th} \end{cases},$$

where p_{max} is a configurable constant that determines the maximum dropping probability.

The packet drop triggers the TCP receiver to send DUPACKs to the sender. The TCP sender is therefore able to adjust its window size in response to the packet drop by means of congestion control. The early packet drop prevents the router from entering the fully congested state. By doing so, the average queue length at the router can be kept small, hence reducing the queuing delay and improving the TCP throughput.

Explicit Congestion Notification (ECN) [55] is an extension to RED. Instead of randomly early dropping packets, an ECN router marks packets to alert the sender of incipient congestion. ECN is an explicit signaling mechanism designed to convey network congestion information from routers to end stations. However, since the signaling only uses one bit for such congestion information, the information conveyed is not quantitative.

For TCP congestion control, ECN works by configuring the intermediate routers to mark packets with CE (Congestion Experienced) bit in the IP header when

the router's average queue occupancy exceeds a threshold, so that the TCP receiver can echo this information back to the sender via ACK by setting the ECE (Explicit Congestion Echo) bit in the TCP header. The router's packet marking/dropping algorithm works as follows. ECN is usually implemented by configuring the RED router to mark instead of dropping packets when the average queue length lies between the min_{th} and max_{th} thresholds.[4]

In the forward direction of the TCP flow, when the receiver receives a packet that has its CE bit set in its IP header, the receiver sets the ECE bit in the TCP header for all subsequent ACKs it sends back to the sender until the sender signals the receiver of its action on the congestion notification. This is achieved by the sender setting the CWR (Congestion Window Reduced) bit in the TCP header. In the reverse direction of the flow, upon reception of a packet with the ECE bit set, the TCP sender immediately reduces its transmission rate by reducing the congestion window, and then invokes the congestion avoidance algorithm to gradually increase the rate back to a sustainable level (i.e., the normal congestion control algorithm as if a packet loss has been detected). The sender also sets the CR bit to signal the receiver of its action.

The principle of our proposed loss type differentiation algorithm is that the ECN marking (i.e., CE bit being set or cleared) of a DUPACK represents the congestion state of the router at the time the corresponding packet was lost. Because of the temporal proximity between the packet lost due to the congestion and the packet that triggered the DUPACK (which must be a packet enqueued after the lost packet), the congestion information carried by the CE bit of the DUPACK

[4]Here, we must note that although most ECN routers are implemented together with the RED functionality by instructing the RED router to mark packets instead of dropping packets, this practice is not mandated and may not be optimum. With packet marking, more design freedom are introduced in terms of when and how (i.e., by what probability or marking function) the packet is to be marked. An insightful discussion can be found in [56].

can be used to deduce whether the most immediate lost packet before this DUPACK was due to congestion or reasons other than congestion (in our case, the wireless link errors).

Therefore, the loss type differentiation in our end-to-end solution to TCP's performance degradation in wireless IP communications is essentially based on the sender's examination of the congestion notification information set by the AQM routers and carried back in the DUPACK packets. The sender, based on this explicit network congestion status, reacts to TCP's DUPACK differently, and thus reduces the number of *unnecessary* window reductions in case that packet losses are not caused by congestion.

By using the AQM's packet marking mechanism (e.g., ECN), which is originally designed for Internet congestion notification and prevention, to differentiate the packet loss type serves two goals with one tool. As pointed out in [56], packet marking instead of dropping in the routers' AQM schemes opens a new opportunity and venue for functions beyond the congestion control. Our proposed solution is, to the best of our knowledge, by far the first application of such philosophy in solving the wireless performance degradation problem of TCP. Our solution reacts to congestion notification markings by reducing sender's transmission rate to the estimated achievable rate (ARE), and therefore, avoids or prevents congestion, hence reducing congestion packet losses. At the same time, it uses the same information to deduce the cause of packet losses so that more sensible and efficient retransmission and rate adaptation mechanism can be implemented to improve TCP's performance in the wireless environment.

Loss type differentiation via explicit congestion notification is more direct, more accurate, less heuristic, and much less complex than other proposed mechanisms, such as the Hidden Markov Model (HMM) approach in [11] which requires off-line training of the models and Viterbi [57] algorithm for loss classifi-

cation, and the Bayesian approach in [14] that also requires HMM models for the estimation of the *a priori* probabilities.

4.3 Summary

In this chapter, we have presented our design philosophy for an end-to-end solution to TCP performance enhancement in wireless networks. We have outlined our design principles as first, a robust rate estimation algorithm that an adaptive congestion control algorithm can rely on, second, an adaptive congestion control algorithm that differs from standard TCP's drastic rate halving scheme, and third, a loss differentiation algorithm based on congestion marking by AQM-enabled link and its integration into the window-based congestion control.

In the following chapter, Chapter 5, we present, in details, the works we have conducted along these design principles.

CHAPTER 5

AN END-TO-END SOLUTION WITH ACHIEVABLE RATE ESTIMATION AND DETERMINISTIC PACKET MARKING

In this chapter, we present TCP-Jersey which is an end-to-end solution to the TCP performance degradation in wireless IP communications.

TCP-Jersey is an integrated implementation of our proposed design principle. It consists of two inter-related key components, namely achievable rate estimation (ARE) based additive-increase and adaptive-decrease (AIAD) congestion control at the sender side, and packet loss differentiation using a simple active queue management (AQM) router that *deterministically* marks packets when there is impending congestion, which we call the congestion warning (CW) marking scheme.

TCP-Jersey is a modification of the standard TCP protocol rather than a completely new design of the transport protocol. It is an end-to-end approach to the performance degradation problem since it does not require split mode connection establishment and maintenance using special wireless-aware software agents at the routers. TCP-Jersey also differs from other solutions that rely on the link layer error notifications for packet loss differentiation.

TCP-Jersey is also unique among other proposed end-to-end solutions to the performance degradation problem of TCP over wireless networks. It distinguishes packet losses attributed to wireless link errors from congestion induced packet losses directly from the explicit CW marking in the DUPACK packets, rather than inferring the loss type based on packet delay or delay jitter as in many other proposed solutions; nor by undergoing a computationally expensive off-line training

of a hidden Markov model (HMM), or a Bayesian estimation/detection process that requires estimations of *a priori* loss probability distributions of different loss types.

TCP-Jersey's loss differentiation mechanism is a straightforward and effective functional extension of the packet-marking based AQM. Combined with the sender side modification (i.e., ARE-based AIAD) of the standard TCP, our solution is fully compatible to the current practice in Internet congestion control and queue management, but with an additional function of loss type differentiation that effectively enhances TCP's performance over lossy wireless networks.

5.1 Estimating Achievable Rate via Returning ACKs

TCP Tahoe, Reno and their variants detect the available bandwidth on the bottleneck link by continuously increasing the window size until the network is congested and then decrease the window size multiplicatively, e.g., the AIMD algorithm. However, the decrement of window size is rather heuristic and coarse, i.e., halving the current window size. This is because these TCP schemes lack the ability to quantitatively estimate the available bandwidth before the congestion happens. These TCP schemes' congestion control mechanisms are reactive rather than proactive and preventive.

TCP Westwood [36] proposes a remedy to this problem by employing its available bandwidth estimation technique at the sender side based on the interval of the returning ACKs. In TCP Westwood, the bandwidth estimator works as follows. When the sender receives an ACK at time t_k, it records a sample of the bandwidth as

$$b_k = \frac{d_k}{t_k - t_{k-1}}, \tag{5.1}$$

where d_k is the amount of data the ACK acknowledged, and t_{k-1} is the time the previous ACK was received. This sample bandwidth is further smoothed by a first-

order low-pass filter resulting in the following discrete-time implementation

$$\hat{b}_k = \frac{\frac{2\tau}{t_k - t_{k-1}} - 1}{\frac{2\tau}{t_k - t_{k-1}} + 1}\hat{b}_{k-1} + \frac{b_k + b_{k-1}}{\frac{2\tau}{t_k - t_{k-1}} + 1}, \tag{5.2}$$

where $\hat{b}_k$ is the filtered measure of the available bandwidth at time t_k, and $1/\tau$ is the cut-off frequency of the filter. In addition, TCP Westwood also employs a timer in its bandwidth estimator such that if time τ/m, where m is suggested to be greater than 2, has passed without receiving a new ACK, the filter is fed with a sample of $b_k = 0$.

When TCP Westwood determines that the link is congested after receiving 3 DUPACKs, it sets the slow start threshold to reflect its estimated bandwidth delay product as

$$ssthresh = \frac{BWE \times RTT_{min}}{seg\_size}, \tag{5.3}$$

where BWE is the estimated bandwidth, RTT_{min} is the measured minimum round-trip-time, and seg_size is the TCP's segment size. The congestion window, $cwnd$, is then set to be the same as $ssthresh$ if the connection is not in the slow start phase, i.e., $cwnd > ssthresh$. However, authors in [36] does not describe the practical rules of choosing parameters τ and m.

We shall note that the available bandwidth estimation as described above is essentially an estimation technique based on the inter-packet dispersion. Network bandwidth and bottleneck measurement, estimation, and identification, either on-line or off-line, is itself an area of active research. Recent studies show that estimation techniques based on the inter-packet dispersion, such as the BWE, or $\hat{b}_k$, in TCP Westwood, does not yield the *available bandwidth* of the end-to-end path (see [58] and the references therein). The name "available bandwidth estimation" is, thus, inaccurate.

In (5.2), if we define $\Delta_k = t_k - t_{k-1}$, and $\alpha_k = \frac{2\tau - \Delta_k}{2\tau + \Delta_k}$, it becomes

$$\hat{b}_k = \alpha_k \hat{b}_{k-1} + (1 - \alpha_k)\frac{b_k - b_{k-1}}{2}. \tag{5.4}$$

We simplify the analysis by ignoring the time-varying nature of the inter-ACK dispersion, Δ_k, for the time-bing, by dropping the subscript of α_k, and take the z-transform of the above. The approximated discrete-time transfer function of the filter can be expressed as

$$H(z) = \frac{\hat{B}(z)}{B(z)} = \frac{(1-\alpha)(1+z^{-1})}{2(1-\alpha z^{-1})}, \tag{5.5}$$

where $\hat{B}(z)$ is the z-transform of the filter output $\{\hat{b}_k\}$, and $B(z)$ is the z-transform of the measurement sample $\{b_k\}$.

Applying the bilinear transformation [59], i.e., $z^{-1} = \frac{1-\frac{T_s}{2}s}{1+\frac{T_s}{2}s}$, where T_s is the sampling interval, we have the continuous-time transfer function of the filter as

$$H(s) = \frac{1}{1 + \frac{\tau T_s}{\Delta_k} s}. \tag{5.6}$$

Apparently, we do not have a constant sampling interval, T_s, since the intervals of the returning ACKs are not equally spaced. Since the available bandwidth estimator samples the instantaneous bandwidth measure $\{b_k\}$ at each arrival of the ACKs, we can approximate $T_s \approx \Delta_k$. Thus, the transfer function of the low-pass filter of the TCP Westwood's available bandwidth estimation becomes

$$H(s) = \frac{1}{1 + \tau s}. \tag{5.7}$$

Therefore, the agility and the smoothness of the filter is determined by the pre-configured parameter τ. This is a typical first order low-pass filter with a constant parameter, and as such, only agility or smoothness can be achieved, but not both at the same time. In other words, the filter is not adaptive.

5.2 Proposed Achievable Rate Estimation

As a matter of fact, the precise measurement of the *available bandwidth* may not be necessary for the TCP sender to adjust its sending rate. An approximate estimation of the sender's *achievable rate*, or, in other words, the throughput, *before* the congestion occurs would be enough for the sender to make a more sensible adjustment of its sending rate upon the detection of congestion. As discussed above, the on-line measure of the available bandwidth is an extremely difficult task, if not impossible, particularly when the measurement process has to be inter-woven with the TCP's window adjustment algorithm, we opt to measure the *sender perceived achievable rate*.

We adopt the same idea of estimating the achievable rate at the sender by observing the rate of the returning ACKs, but uses a rather simple, but adaptive estimator. The achievable rate estimator we propose in this work is derived from the Time-Sliding Window (TSW) estimator proposed in [60].

In [60], Clark and Fang proposed that the network router employs the following estimator to estimate the bandwidth occupied by individual flows for the purpose of fair bandwidth allocation

$$R_k = \frac{T_w R_{k-1} + L_k}{(t_k - t_{k-1}) + T_w}, \tag{5.8}$$

where R_k is the estimated flow rate when packet k arrives at time t_k, and T_w is a pre-configured time constant called the time window, and L_k is the size of the k^{th} packet. The authors show that this simple rate estimation has an attractive property such that the estimate decays with time, making it suitable for networks with non-static bandwidth delay product, a characteristic often exhibited in wireless networks.

A later IETF RFC2859 [61] based on the same estimation technique suggests that a suitable choice of the time window, T_w, should be approximately the flow's

round-trip-time (RTT). However, since individual flow's RTT is generally unknown to a router, in practice, T_w is set to be the conceivable approximate of the maximum RTT of all the flows that traverse the router.

Although the TSW flow rate estimator was proposed for a different purpose, the same idea is well suited for our need, namely estimation of the flow's achievable rate at the flow source. We propose a slightly modified TSW for the TCP sender to estimate its achievable rate based on the inter-arrivals of the returning ACKs. We call our rate estimator the *source perceived achievable rate estimator* (SPARE), or simply ARE. The ARE estimator is integrated into the TCP sender, it updates the estimation upon each received ACK as follows:

$$R_k = \frac{\tau_k \times R_{k-1} + L_k}{(t_k - t_{k-1}) + \tau_k}, \tag{5.9}$$

where τ_k is the sum of the sender's estimation of the end-to-end round trip time (RTT) and its variation at time k. That is

$$\tau_k = RTT_k + RTT\_VAR_k, \tag{5.10}$$

where the measurement of RTT_k and RTT_VAR_k is defined in [18] and implemented in all standard TCP protocols.

Let $\Delta_k = t_k - t_{k-1}$, and $\alpha = \frac{\Delta_k}{\Delta_k + \tau_k}$, we can rewrite (5.9) as

$$R_k = (1 - \alpha)R_{k-1} + \alpha r_k, \tag{5.11}$$

where $r_k = \frac{L_k}{\Delta_k}$ is the instantaneous rate sample on the k^{th} ACK arrival. A natural way to express (5.11) in the form of a differential equation is

$$\frac{dR}{dt} = aR(t) + br(t). \tag{5.12}$$

Then, in a sampled system, $R(t_k)$ is given by

$$R(t_k) = e^{a(t_k - t_{k-1})} R(t_{k-1}) - \int_{t_{k-1}}^{t_k} e^{A(t_k - \tau)} b d\tau r(t_k). \tag{5.13}$$

The differential equation matches the discrete time system exactly at the sample points and there is no accumulation of error as k increases [62].

Comparing the above equation with the original discrete time system equation, we can pair up the coefficients as

$$1 - \alpha = e^{aT_s}, \tag{5.14}$$

or

$$a = \frac{ln(1-\alpha)}{T_s} = -b, \tag{5.15}$$

where T_s is the sampling interval. Therefore, the behavior of $R(t)$ can be described as

$$\dot{R}(t) = \frac{ln(1-\alpha)}{T_s}(R(t) - r(t)). \tag{5.16}$$

The input-output transfer function of (5.16) is

$$H(s) = \frac{R(s)}{r(s)} = \frac{1}{1 - \frac{1}{\beta}s}, \tag{5.17}$$

where $\beta = \frac{ln(1-\alpha)}{T_s}$. This is a typical first-order low-pass filter with a time-constant of $T_f = -\frac{1}{\beta}$.

Recall that instantaneous rate samples are taken at each arrival of the ACKs; therefore, we have $T_s \approx \Delta_k$. Substituting α into the T_f, after algebraic simplifications, we have a time-varying filter time-constant as

$$T_f = -\frac{1}{\beta} = \frac{\Delta_k}{ln(\Delta_k + \tau_k) - ln(\tau_k)}, \tag{5.18}$$

where Δ_k is the inter-ACK arrival time of the k^{th} ACK, and τ_k is, according to (5.10), the sum of the TCP sender's estimations of the end-to-end RTT and the approximated standard deviation [18] of the RTT.

The filter time-constant T_f in the proposed achievable rate estimator is an increasing function in both the inter-ACK arrival time, Δ_k, and the sum of the end-to-end RTT and RTT variation, τ_k. When T_f is large, the filter becomes less responsive and the filtered output, the estimate, becomes smoother. On the other hand, a small T_f makes the estimator agile in tracking the changes in the sample data.

In the duration of a TCP flow, the inter-ACK arrival time, Δ, could increase for the following reasons.

- The sender throttles the sending rate in response to network congestion;
- The reverse path from the receiver to the sender is experiencing congestion, which in turn slows down the TCP's self-clocking pace;
- The forward path from the sender to the receiver is congested (since ACKs are triggered by the arriving packets at the receiver).

Similarly, the sum of the sender-measured RTT and its variation, τ, could increase during a TCP session for reasons such as

- increased congestion at the bottleneck link,
- volatility of the traffic load at the bottleneck link.

In both cases, (i.e., increase of Δ and increase of τ) the end-to-end system, which consists of the TCP flow, the cross-traffic, and the bottleneck router queue, is in a less stable state; as a result, the instantaneous rate samples $\{r_k\}$ fluctuate in a wider range.

The proposed achievable rate estimation (ARE) algorithm (5.9) automatically adapts to the changing network conditions in such a way that when the network is stable, the estimator has a smaller T_f and is thus agile and quick in tracking the *non-transient* changes of the achievable rate; while the network is in a transient unstable state, the estimator resumes a larger value of T_f and therefore becomes less affected by the short-term abrupt changes.

This is, in essence, in the same spirit of the flip-flop TSW filter designed in [63] and the flip-flop EWMA (exponentially weighted moving average) filter proposed in [8]. However, the proposed adaptive estimator requires no pre-configured flipping thresholds as in [63, 8].

The following experiment demonstrates the adaptiveness of the proposed ARE estimator. In this experiment, we use the NS-2 network simulator [25] to conduct a simulation on a simple network. The network consists of only one link connecting two nodes, the source node and the destination node. The link between the two nodes is a 1.5Mbps duplex link with one-way propagation delay of 10ms; it is configured with a buffer that can hold up to 100 packets.

We devise three packet flows from the source to the destination. The first is a standard TCP Newreno flow, in which we have implemented both achievable rate estimators (5.2) and (5.9) to record the corresponding estimated achievable rates by the two estimators. Note, in this experiment, the output of the rate estimator is not used in any way to change the TCP's congestion control algorithm, that is, the TCP employs the standard congestion control algorithm, i.e., AIMD. The TCP flow is active for the entire duration of the simulation.

The other two flows are both Pareto On-Off traffic carried by UDP packets. They are configured with the average burst time of 0.6s and average idle time of 0.4s. The Pareto shape parameters for both flows are set to be 1.5 and the average burst rates are both set to 0.8Mbps. The two Pareto flows are activated and

deactivated as follows. The first Pareto flow is initially activated at time 5s and then deactivated at time 20s, and later resumed at time 40s. The second Pareto flow is first activated at time 10s and then deactivated at time 30s, and later resumed at time 45s. The entire simulation is run for 60 seconds. We repeat the simulation several times for different settings of τ for the rate estimator of (5.2). We plot the output of the rate estimators and other related data in the following.

First, we record the instantaneous TCP rate samples $r_k = \frac{L_k}{\Delta_k}$, which is also referred to as b_k in (5.2), where L_k is the size of the k^{th} packet, and Δ_k is the inter-arrival time between packet k and $k-1$. The instantaneous rate sample is plotted in Figure 5.1. Here, we can see that the instantaneous rate samples are stable during the periods from 0s to 5s and from 30s to 40s when the TCP flow is the only traffic on the link. Whereas, in other time intervals, the rate samples appear to be fluctuating due to the congestion caused by the Pareto flows traversing the same link. The corresponding queue length (normalized by the total buffer size) of the

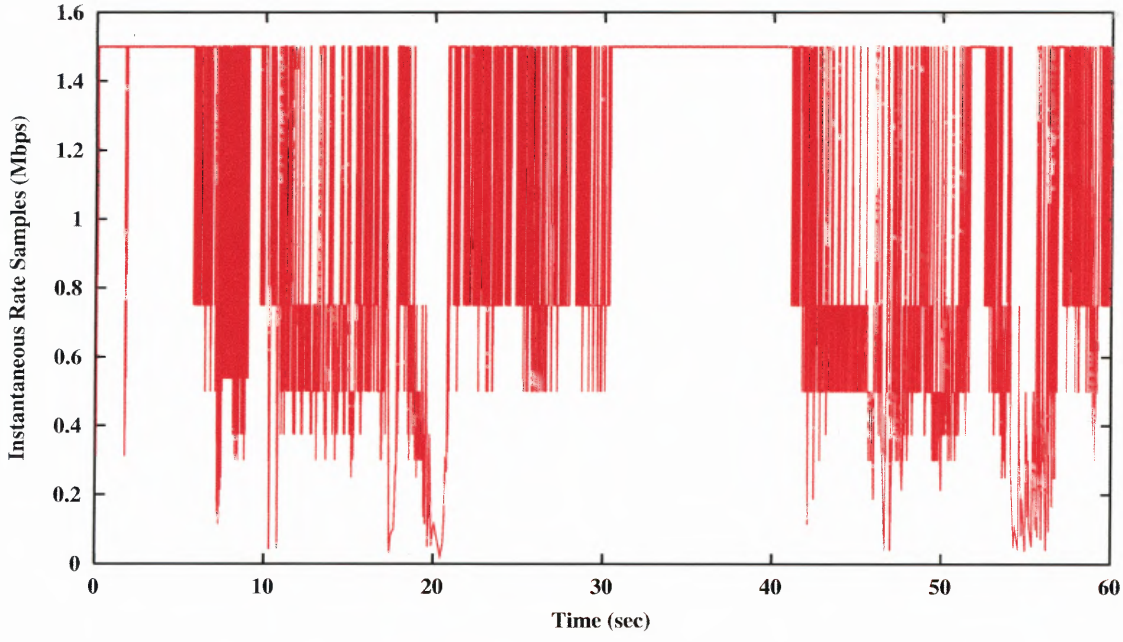


Figure 5.1 Instantaneous rate samples of the TCP flow.

bottleneck link, the average end-to-end RTT and RTT variation as measured by the standard TCP sender according to [18] are also plotted in Figure 5.2.

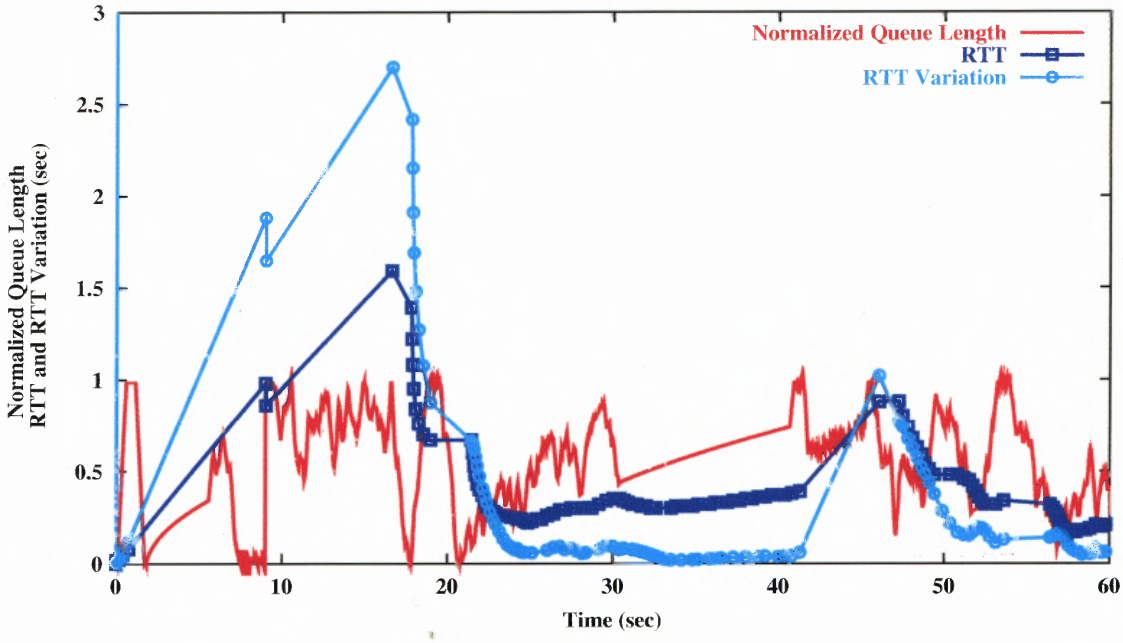


Figure 5.2 Normalized bottleneck queue length, TCP's measurements of end-to-end RTT and RTT variation.

Next, in Figure 5.3, we plot the output of the achievable rate estimator (5.9) implemented at the TCP sender. It is clear that the estimator exhibits good tracking ability while the system is relatively stable, and is smoother while the samples have large fluctuation.

As a comparison, Figure 5.4 plots several estimations of the estimator (5.2) with different sampling intervals, τ. As shown in the figure, when τ is set to 1.0s, which is approximately in the neighborhood of the average RTT, the output of (5.2) is comparable to that of (5.9). However, when $\tau = 0.1$, the output becomes too sensitive to transient changes of the samples; on the other hand, when $\tau = 5.0$, the estimator's response becomes rather sluggish.

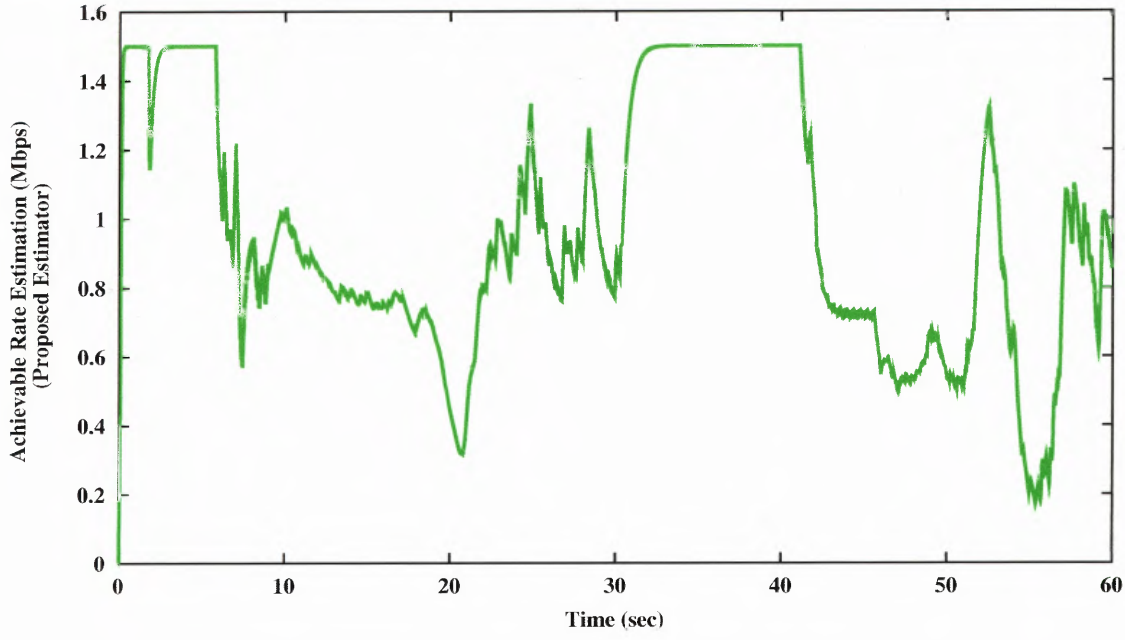


Figure 5.3 Achievable rate estimation by the proposed estimator (5.9).

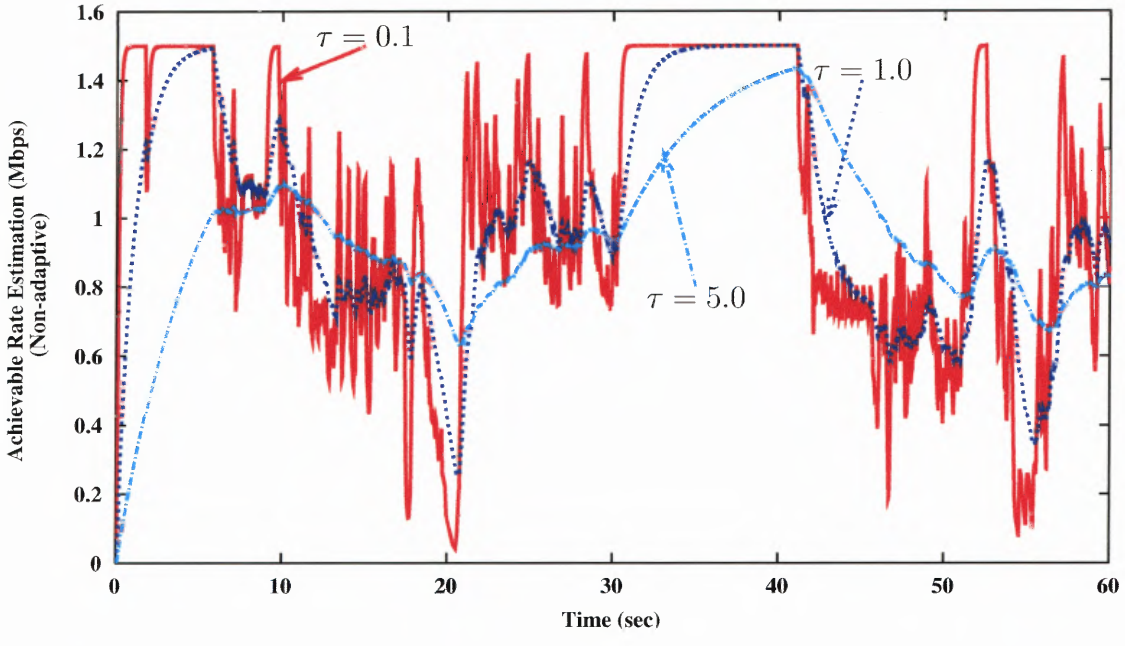


Figure 5.4 Achievable rate estimation by a non-adaptive estimator as (5.2), with τ set to 0.1s, 1.0s, and 5.0s, respectively.

It is worth noting that a simple modification to the first-order low-pass filter used in TCP Westwood's estimator (5.2) can be made by setting the parameter τ in (5.2) as

$$\hat{\tau} = RTT + RTT\_VAR, \tag{5.19}$$

instead of a pre-configured constant, where RTT and RTT_VAR are the TCP's measurement of the round-trip time and its variation. Accordingly, the modified Westwood estimator is implemented as

$$\hat{b}_k = \frac{\frac{2\hat{\tau}}{t_k - t_{k-1}} - 1}{\frac{2\hat{\tau}}{t_k - t_{k-1}} + 1}\hat{b}_{k-1} + \frac{b_k + b_{k-1}}{\frac{2\hat{\tau}}{t_k - t_{k-1}} + 1}, \tag{5.20}$$

and the modified filter's transfer function becomes

$$H(s) = \frac{1}{1 + \hat{\tau}s}. \tag{5.21}$$

This modified filter adaptively sets its time constant to the estimated average RTT of the flow when the network is stable, i.e., RTT_VAR is near zero. When the network is in the transient and unstable state, it increases the time constant and hence produces smoother output. The modified low-pass filter (5.21) is referred to as filter type 2 in subsequent chapters, and the adaptive TSW algorithm (5.17) used in the proposed ARE is referred to as filter type 1.

5.3 Congestion Warning: Explicit and Deterministic Packet Marking

The current ECN scheme [55] marks packets probabilistically while the average queue length lies between min_{th} and max_{th}. The router thereby not only informs the sender of the congestion but also influences on which TCP connection the congestion window size would be adjusted due to its randomness in packet marking. Echoing back ECN information from the receiver to the sender takes time whereas network situation is constantly changing. Although ECN provides valuable congestion information, this information may not be timely enough for the sender to make the right decision suitable for the current network status under all circumstances. Moreover, both RED and ECN are sensitive to parameter settings [64]. Improper parameter settings may lead to unsatisfactory TCP performance [65].

We therefore propose a simpler congestion notification scheme, namely Congestion Warning (CW), with fewer parameter settings, yet still providing essential and accurate congestion information to the sender. We propose that the router shall mark all the packets, deterministically rather than probabilistically, when the average queue length exceeds a threshold (thresh), and leave the TCP sender who receives marks to decide its window adjustment strategy. The router's marking scheme is depicted in Figure 5.5. CW inherits the same information bits used in the original ECN implementation, i.e., the CE bit in the IP header and the ECE and CWR bits in the TCP header to convey the congestion warning information. We assume that routers along the connection path implement the original ECN mechanism and its parameters can be configured to reflect the CW scheme without the need to change the router's software. We believe this is a fairly realistic assumption because RED and ECN have gained their popularity since they were first introduced in early 90's.

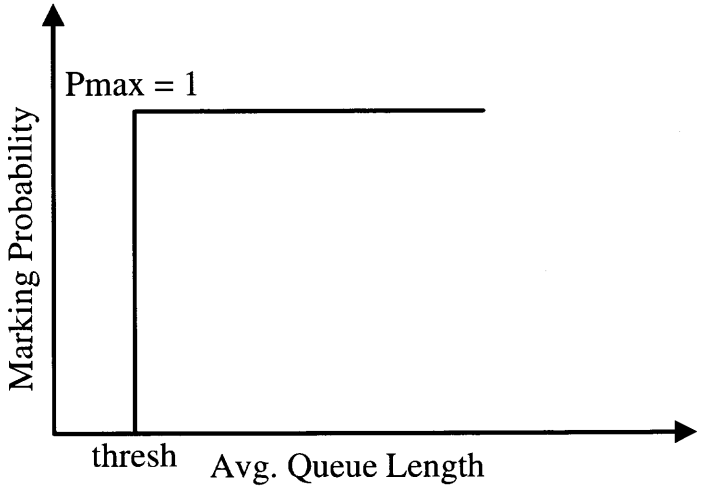
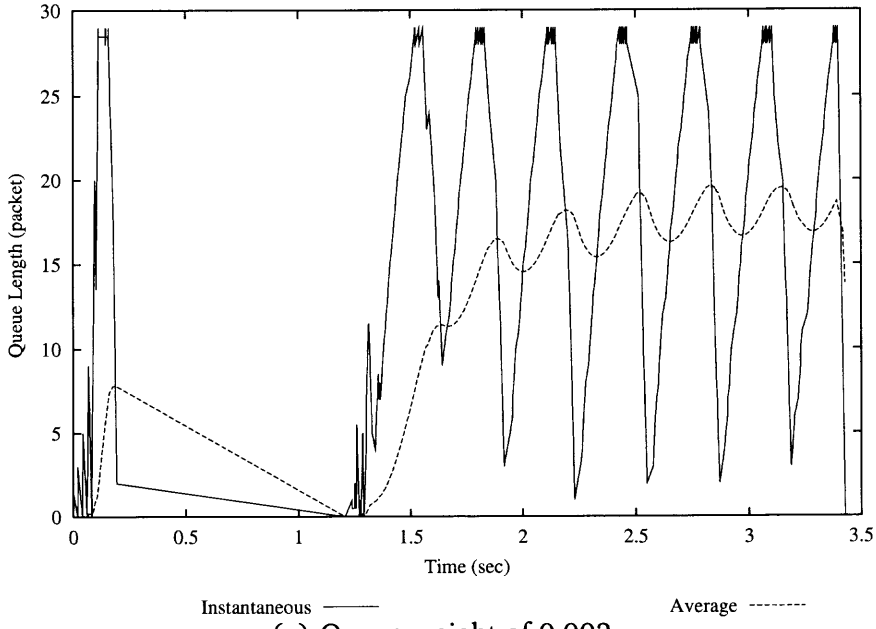


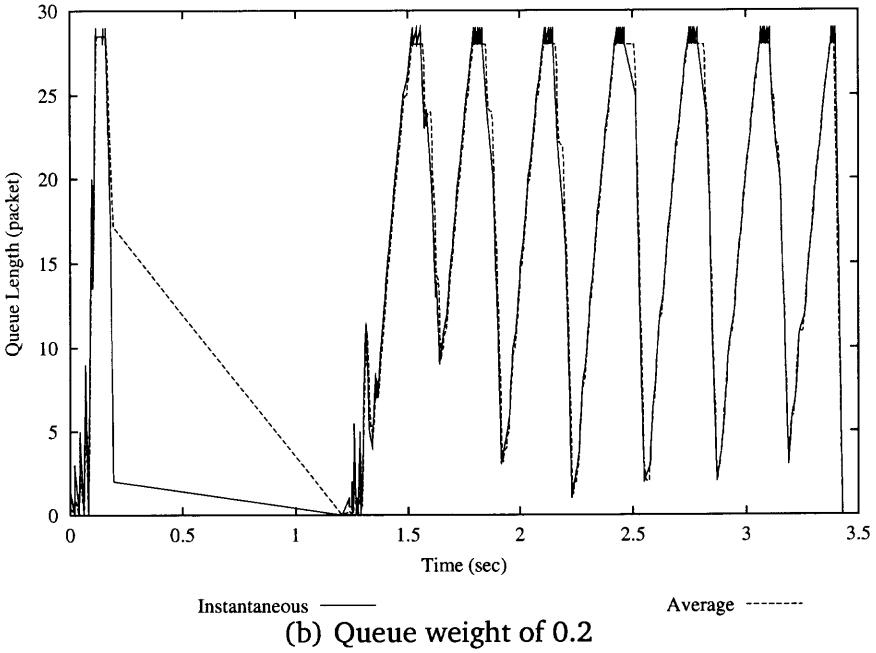
Figure 5.5 Deterministic Packet Marking of CW Scheme.

The calculation of the average queue length in CW also differs from what is in the original ECN. In the original ECN, the average queue length is largely dependent on a long-term averaging value of the instantaneous queue length [54]. However, for our case, a larger queue weight is preferred since we expect the average queue length to closely track the instantaneous queue length and at the same time smooth out small spikes in the instantaneous queue length. A close tracking of the instantaneous queue length provides the sender with more accurate buffer information of the router. The original queue weight suggested in [54] is rather small, e.g., 0.002. Experiments show that a queue weight of 0.2 would be good enough to track and smooth the instantaneous queue length. The effect of the queue weight on the average queue length is illustrated in Figure 5.6.

Figure 5.6(a) shows that the average queue length does not follow the instantaneous queue length when the queue weight is set to 0.002. Figure 5.6(b) shows that the average queue length closely follows the instantaneous queue length as expected for queue weight 0.2. Note that such settings make sense in our design. Since in our scheme, we do not rely on the router's AQM for TCP congestion control,



(a) Queue weight of 0.002



(b) Queue weight of 0.2

Figure 5.6 Average queue length vs. instantaneous queue length with different queue weights. In our proposed CW packet marking scheme, we prefer the larger weight.

rather, we only require the router to alert, in a timely manner, about the incipient congestion and let the TCP sender to carry out the control mechanism.

The deterministic packet marking of CW routers helps the TCP sender to differentiate the cause of packet losses. When the TCP sender receives DUPACK with the CW mark, it knows for sure that the network is in the congested state and assumes that the packet loss indicated by DUPACK is more likely to be caused by congestion. On the other hand, if a DUPACK without the CW mark is received, the sender could assume with higher confidence that the loss has occurred due to transmission errors. In the wired-wireless heterogeneous network, this packet loss is more likely attributed to errors on the wireless link. Suppose that the packet loss is caused by a random bit error in the wireless link, the wireless error recovery procedure at the TCP sender would not reduce its current window size, but retransmit the lost packets using the current window size since there is no sign of congestion; and the DUPACK without mark actually indicates that the network has the capability to deliver the packets with the current window size. We will discuss the procedure in more detail in Section 5.4.

Accurately differentiating wireless packet losses from congestion packet losses is one of the key issues in improving TCP performance for IP communications in the wireless environment. Here, we also must note that erroneously interpreting a packet loss due to the wireless error as congestion does not hurt too much. However, misinterpreting a packet loss triggered by congestion is more severe because the wireless error recovery will react as if there were no congestion. Suppose a packet loss is interpreted as a link error, but was actually caused by congestion, retransmission without reducing window size only aggravates the network congestion level. Therefore, we would rather set the threshold at the router aggressively to avoid missing any packet loss caused by buffer overflow. Experiments show that setting the threshold at 1/3 of the link's buffer capacity

is satisfactory. Almost 100% of congestion losses and most link losses are differentiated successfully.

5.4 TCP-Jersey: ARE-based Adaptive Congestion Control and Loss Differentiation via AQM Packet Marking

In this section, we present a modified version of TCP protocol, called TCP-Jersey, that incorporates the achievable rate estimation (ARE) and the simple congestion warning packet marking (CW) to combat the TCP's performance degradation in wireless IP communications.

As we have described before, the key factor that causes the performance degradation of the standard TCP protocol when used in the wireless environment is that the protocol does not differentiate the type of packet losses, but rather treats every packet loss as a sign of network congestion. As a result every packet loss triggers a reduction of the TCP sender's sending rate by half. As studied by Lakshman et al. [24, 66], for the traditional TCP (TCP-Tahoe and TCP-Reno), the window size at the beginning of the congestion avoidance phase strongly influences the throughput in the cycle[1], because the congestion avoidance phase accounts for the bulk of the packet transmission in the cycle. Since random packet losses due to wireless link errors are interpreted as congestion signals in the traditional TCP, packet losses occur relatively early in a cycle, resulting in small initial values for the congestion avoidance phase that follows. This results in small windows sizes, which are determined by the random wireless errors rather than the congestion, throughout the congestion avoidance phase, and therefore causes performance degradation in terms of throughput and link utilization.

[1]A cycle in TCP's window evolution, as defined in [24,66], starts from the previous window reduction and ends before the next, which is one tooth of the typical sawtooth pattern as we have described before, and illustrated in Figure 2.1.

TCP-Jersey presents an end-to-end solution to improving the performance in networks consisting of wireless links by taking a joint design approach that combines the CW packet marking at the routers and the ARE rate estimation at the sender. The rationale of proposing a joint approach with the combination of CW and ARE is justified below.

Intermediate routers generate ECN marking when there is a sign of congestion judged by comparing the average queue length with thresholds. In networks with the presence of wireless links, the bandwidth-delay-products of the links no longer stay static during a session. The congestion indication created by CW may not always be as reliable as we would hope. Blindly dropping the sender s congestion window by a constant upon receiving marked ACKs is certainly not an optimum solution for better TCP performance. For instance, in an all-IP network, many traditional non-IP communications are carried upon the IP network, most noticeably the voice-over-IP services (VoIP). Because of the on-off nature of the voice conversation, by the time a TCP flow sharing bandwidth with VoIP flows receives congestion notification via CW, the VoIP stream may happen to be in the off state, thus making the notification invalid.

We disagree with the traditional TCP's blindly halving its sending rate upon receiving indications of link congestion. There is a proposal of congestion response in TCP that implements the idea of reducing the sending rate in smaller steps, i.e., 1/8 or 1/16 of the current congestion window [67]. However, these fixed window decrease factors are still somewhat heuristic and do not adapt to the level of the link congestion. We believe that the one bit of information used in today's congestion notification mechanism is not enough for TCP to make intelligent adjustment to deal with link congestion. In other words, the single bit of information is not quantitative enough for the TCP to measure the actual level of congestion in the network. Estimation of the network congestion level by cumulating and computing

the frequency of the congestion bits has been proposed, e.g., in [68]. However, such an estimation scheme assumes a global constant parameter known to all TCPs and the routers in the network.

We either need to extend the current congestion notification scheme so that routers can notify TCP end stations of the link congestion in a quantitative way, or use another mechanism that can interact with the binary congestion signals. Our proposed approach, namely TCP-Jersey, combines ARE and CW, which is a minimally modified version of ECN at the router, so that the TCP sender could set its congestion window to a more sensible value when congestion is detected.

Also, as we have stated before, such a combination improves TCP's ability to differentiate random wireless packet losses from losses caused by congestion. We will discuss the details of our proposed algorithm in the next section.

In essence, TCP-Jersey uses ARE to guide the window reduction process so that upon congestion, the sender's congestion window is reduced to reflect a known achievable rate. Such dynamic, as opposed to static, window reduction also, indirectly, makes TCP-Jersey robust to random wireless errors, because it minimizes the rate reduction when a random packet loss is mis-classified as congestion loss. Also, by using the CW marking mechanism, the TCP-Jersey would be able to differentiate most of the random packet losses from those caused by network congestion.

5.5 Operation of TCP-Jersey

The pseudo-code of the TCP-Jersey's ARE algorithm is presented in Figure 5.7, where RTT and RTT_VAR are TCP sender's estimation of average end-to-end round-trip-time and its variance [18], and L is the size of the data packet that the ACK acknowledges. Procedure ARE is invoked by the sender upon receiving an ACK or DUPACK.

```
Initialization:
    ARE = 0;
    T_prev = 0;
(1) ARE()
(2) {
(3)       TW = RTT + RTT_VAR;
(4)       Delta = now - T_prev;
(5)       T_prev = now;
(6)       ARE = (TW * ARE + L) / (Delta + TW);
(7)       return ARE;
(8) }
```

Figure 5.7 Pseudo-code of the TCP-Jersey's ARE algorithm.

TCP-Jersey adopts Slow Start (SS), Congestion Avoidance (CA) and Fast Recovery from Reno but replaces Reno's Fast Retransmit with Explicit Retransmit and introduces the Rate Control procedure. The only difference between Reno's Fast Retransmit procedure and Jersey's Explicit Retransmit procedure is that, unlike Reno's retransmit procedure that halves the current congestion window before starting the retransmission, Explicit Retransmit keeps the current *cwnd*. It leaves the adjustment of the congestion window to the Rate Control procedure. The operation of the Rate Control procedure is also quite simple. The procedure sets the *ssthresh* to *ownd*, the optimum congestion window size computed using (5.22), where RTT_{min} is the minimum RTT measured by the sender, and *seg_size* is the size of the TCP segment; and sets the *cwnd* to the same as *ssthresh* if the connection is in the congestion avoidance phase.

$$ownd = \frac{RTT_{min} \times ARE}{seg\_size}. \tag{5.22}$$

The pseudo-code of the TCP-Jersey's sender receiving module is depicted in Figure 5.8. It operates as follows. Upon entry, it invokes the ABE procedure (Line 3). If an ACK is received without the CW mark, it proceeds as Reno, i.e., invoking SS or CA depending on whether or not the $cwnd$ is below the $ssthresh$ (Line 4-6). If the received ACK or the 3^{rd} DUPACK is marked with the CW bit, it calls the Rate Control procedure to adjust the window size and proceeds with SS or CA if it is an ACK (Line 7-10); or enters the Explicit Retransmit if it is the 3^{rd} DUPACK (Line 11-15). When the 3^{rd} DUPACK is received without the CW mark, TCP-Jersey renders that the packet loss was caused by a random wireless errors, and therefore it enters the Explicit Retransmit without adjusting the window size (Line 16-19).

In Figure 5.9 and Figure 5.10, we show the flow charts of the TCP-Jersey's response to DUPACK and that to normal ACK, respectively.

```
(1) recv()
(2) {
(3)       ARE();
(4)       if ACK and CW == 0    /* an ACK carries no CW mark */
(5)          SS() or CA();      /* proceed as traditional TCP */
(6)       end if
(7)       if ACK and CW == 1    /* an ACK carries CW mark */
(8)          rate_control();    /* adjust window according to ARE */
(9)          SS() or CA();      /* then proceed as traditional TCP */
(10)      end if
(11)      if 3 DUPACK and CW == 1/* packet lost due to congestion */
(12)         rate_control();    /* adjust window according to ARE */
(13)         explicit_retransmit(); /* retransmit lost packet */
                                    /* keep window unchanged */
(14)         fast_recovery();   /* proceed with Reno's fast recovery */
(15)      end if
(16)      if 3 DUPACK and CW == 0/* packet lost due to wireless errors */
(17)         explicit_retransmit(); /* retransmit lost packet */
                                    /* keep window unchanged */
(18)         fast_recovery();
(19)      end if
(20) }
```

Figure 5.8 Pseudo-code of the TCP-Jersey's receiving module.

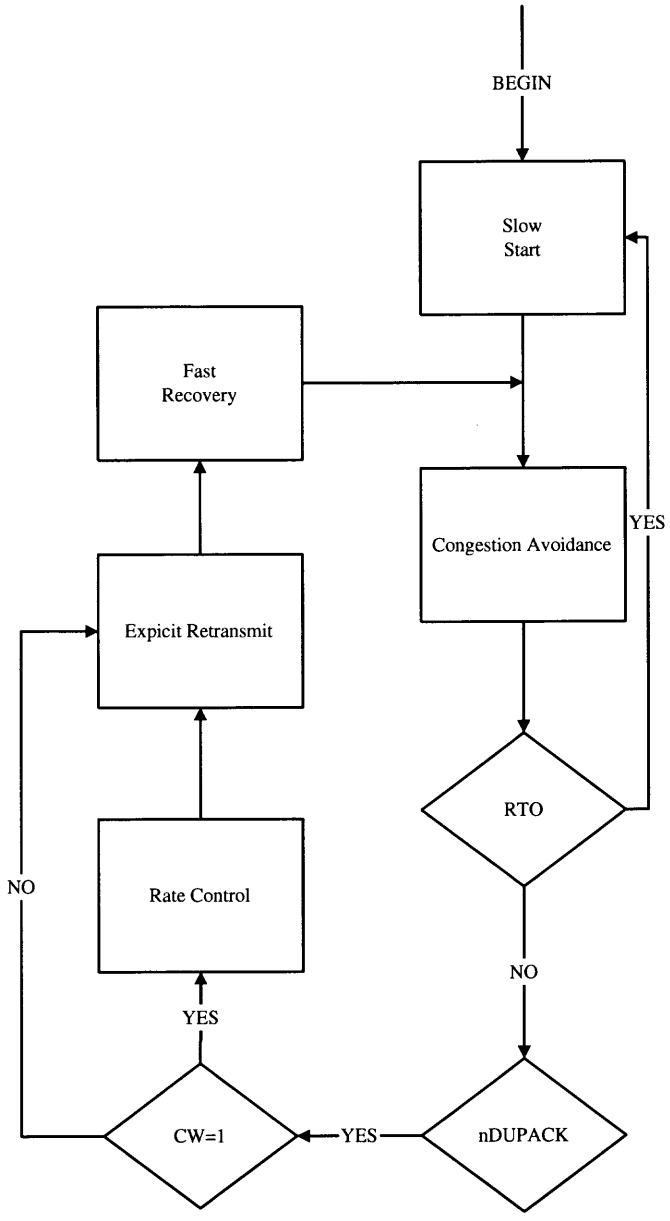


Figure 5.9 Flow chart of the TCP-Jersey's sender response to DUPACK.

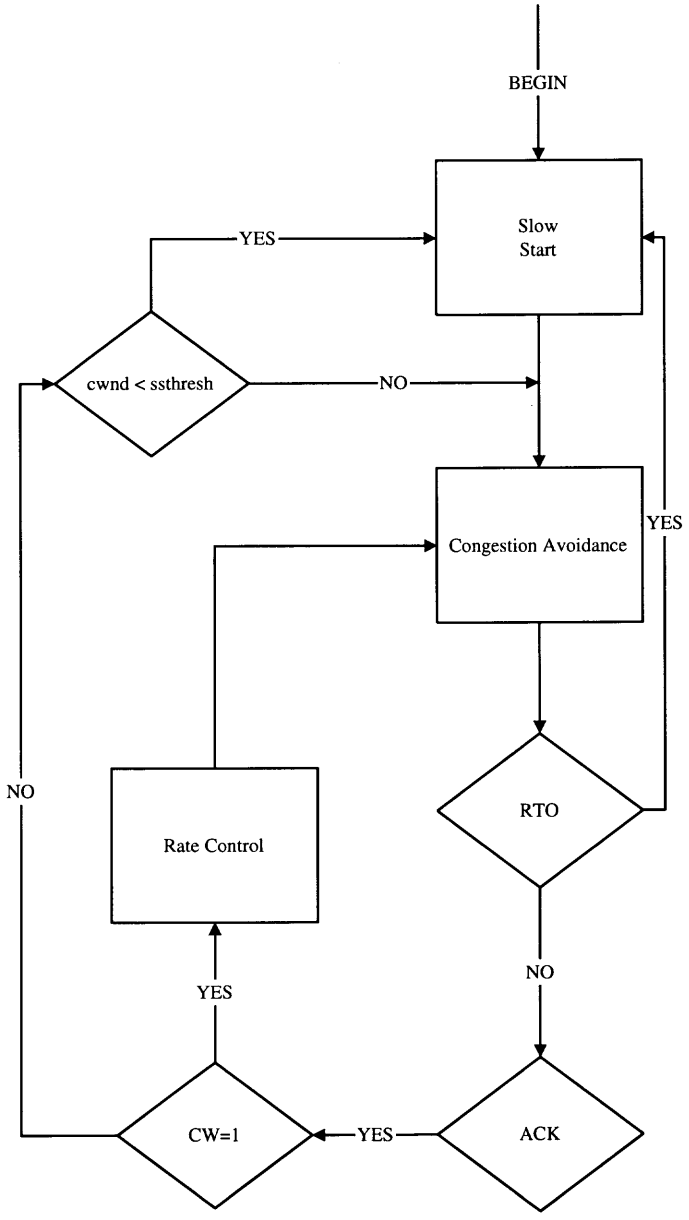


Figure 5.10 Flow chart of the TCP-Jersey's sender response to ACK.

5.6 Performance Evaluation of TCP-Jersey

In this section, we evaluate the performance of our proposed end-to-end solution, namely TCP-Jersey through packet level simulations. Simulations described in the following sub-sections evaluate TCP-Jersey's goodput performance, friendliness to other TCP variants, and fairness within group of TCP-Jersey flows, in mixed wired and wireless networks, with or without the presence of link loss and congestion.

The simulation tool we use is the NS-2 network simulator [25]. We have made the necessary code modification in NS-2 to undertake the experiments.

5.6.1 Goodput Performance

Goodput is the effective amount of data delivered through the network. It is a direct indicator of the protocol performance. It is slightly different from the throughput measure, which, in the context of TCP, includes retransmitted packets. We expect that a good TCP scheme transmit data as many as possible while behaving friendly to other TCP flows in terms of consuming network resource, e.g., bandwidth.

The simulation environment is depicted in Figure 5.11. The source connects to a wireless base station via a 10Mb error free link with 45ms one-way propagation delay. The base station is linked to the destination, a wireless mobile node, via a 2Mb lossy channel with 1ms of one-way propagation delay. A single TCP connection running a long-live FTP application delivers data from the source to the destination.

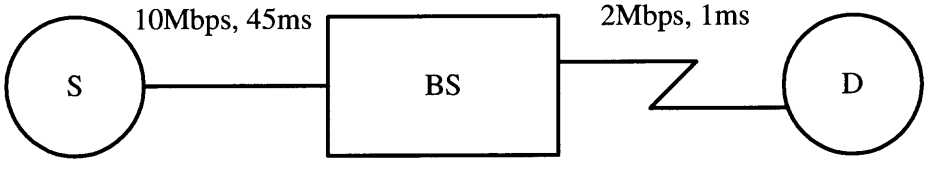


Figure 5.11 Simulation configuration for goodput performance evaluation.

We run the simulation for TCP Tahoe, Reno, Westwood and Jersey, respectively. Under such network configuration, there is almost no congestion. The

random link error rate [2] at the wireless bottleneck link varies from 0.001% to 10%. The simulation time is 100 seconds. The goodput is calculated based on the received ACKs at the sender side. The goodput result is shown in Figure 5.12, where the error rate is plotted in the logarithm scale.

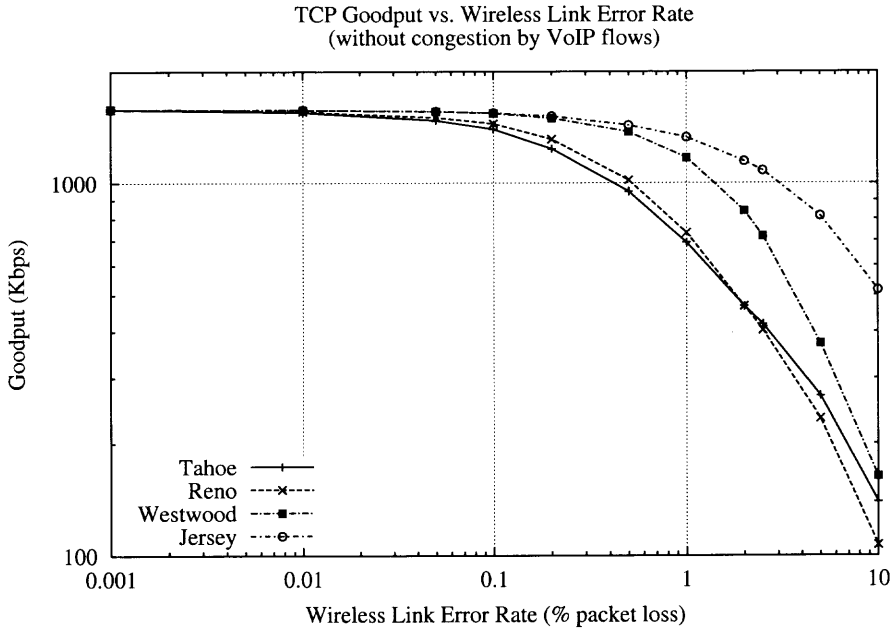


Figure 5.12 Comparison of goodputs of TCP Tahoe, Reno, Westwood and Jersey without presence of link congestion.

For link error rate smaller than 0.01%, all TCP schemes perform closely to each other. Beyond that point, TCP-Jersey starts to outperform the other TCP variants. The closest competitor is TCP Westwood. The goodput of Westwood does not fall behind Jersey very much until the error rate increases to 0.1%. At a very practical error rate for wireless loss, i.e., 1%, Jersey outperforms Westwood by 17% and Reno by 85%. Especially at the high error rate side, Jersey still has satisfactory goodput, whereas other TCPs experience a severe degradation in performance. This phenomenon is anticipated since Jersey handles wireless losses better than the other

[2]The link error rate used in our simulation is the packet loss rate instead of the bit error rate, or BER.

TCP variants because it can distinguish wireless losses from congestion losses and act accordingly.

Next, VoIP traffic is introduced into the same network in Figure 5.11 to create link congestion. The VoIP traffics consist of 5 voice-over-IP flows carried by the UDP protocol. Each VoIP flow is modeled as an independent exponentially distributed on-off source that has, on average, 50% of talk-spurt and 50% of silent period. The average call duration of each VoIP calls is 8 seconds of simulation time. Voices are carried by UDP at 96Kbps rate. New VoIP call arrivals to the pool of 5 VoIP sources form a Poisson process with the average inter-arrival time of 1 second. We run the simulation for TCP Tahoe, Reno, Westwood, and Jersey, respectively, and plotted the goodputs in Figure 5.13.

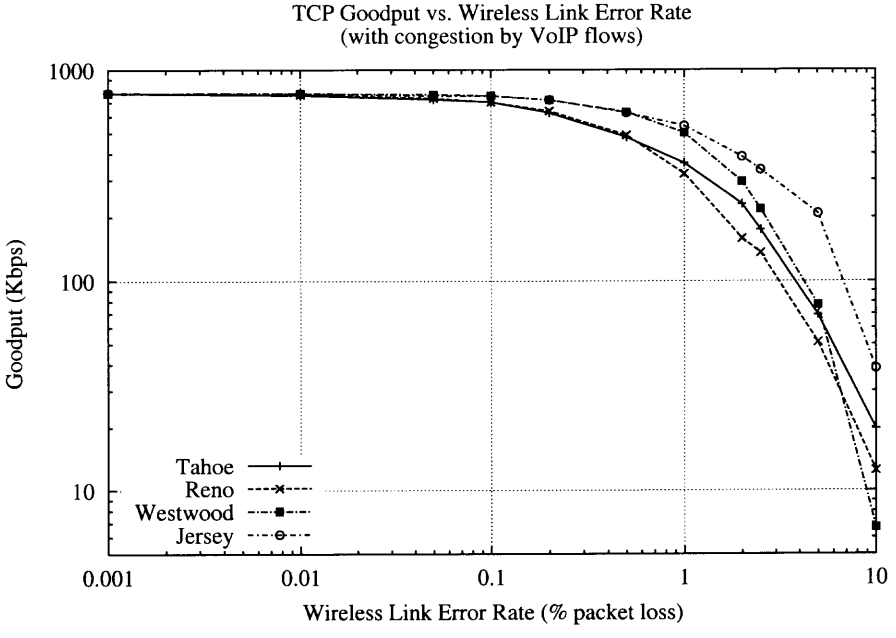


Figure 5.13 Comparison of goodputs of TCP Tahoe, Reno, Westwood and Jersey with presence of link congestion caused by non-responsive VoIP flows.

Each TCP scheme suffers from performance degradation when the network is under congestion, as compared to the goodput in Figure 5.13, because the VoIP flows carried by UDP do not have any congestion control mechanism and never

back off. In other words, they are not responsive to network congestion, and thus not friendly to TCP flows. It can be seen that TCP-Jersey outperforms other TCP schemes at all error rate levels. Particularly, at error rate of 1%, TCP-Jersey improves the goodput over TCP Westwood and TCP-Reno by 9% and 76%, respectively.

Figure 5.14 shows the overall VoIP flows' load and their throughput when the link error is set to be 1% of packet loss under TCP-Jersey. It is observed that VoIP flows pass through at a rate close to their actual loads, implying that at the presence of network congestion, TCP-Jersey does not *over aggressively* consume the available network bandwidth.

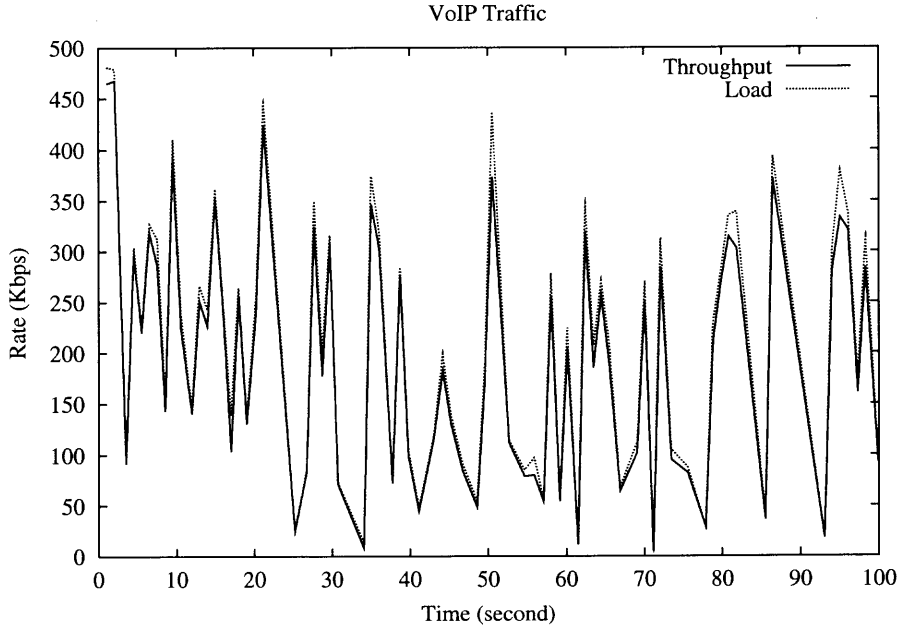


Figure 5.14 Offered load and throughput of the VoIP traffic, when competing with TCP-Jersey.

5.6.2 Fairness of TCP-Jersey

Another important issue of TCP performance is the fairness. Multiple connections of the same TCP scheme must inter-operate fairly and converge to their fair shares. We use the fairness index (5.23), introduced in [19], to measure the fairness of TCP schemes. The fairness index function is expressed as

$$F = \frac{(\sum x_i)^2}{n(\sum x_i^2)}, \tag{5.23}$$

where x_i is the throughput of the i^{th} connection, and n is the number of competing TCP connections. The fairness index F ranges from $1/n$ to 1.0. A perfectly fair bandwidth allocation would result in a fairness index of 1.0. On the contrary, if all bandwidth are consumed by one connection, (5.23) would yield $1/n$.

We setup the simulation as shown in Figure 5.15, where a total of 20 homogeneous TCP flows share a 20Mb bottleneck link. We run the simulation for

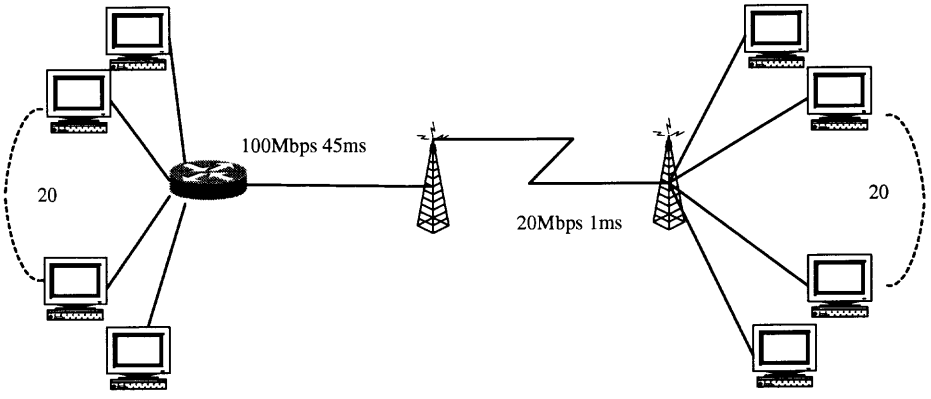


Figure 5.15 Simulation network configuration for fairness evaluation.

different TCP schemes and compare their fairness index; the results are summarized in Table 5.1.

In Table 5.1, the fairness index is calculated based on a total of 20 TCP connections running the FTP application. Error rate is in unit of percentage of packet drops. Link error rate varies from 0 to 10% packet drops. All TCP variants including Jersey achieve fairly satisfactory fairness index.

Table 5.1 Fairness Comparison Among Different TCP Schemes

Error Rate	TCP-Reno	TCP-Westwood	TCP-Jersey
0.0	0.98	0.99	1.0
0.1	0.99	0.97	1.0
0.5	1.0	0.99	0.99
1.0	0.99	1.0	0.99
5.0	0.99	0.99	0.97
10.0	0.97	0.97	0.97

5.6.3 Friendliness of TCP-Jersey

A friendly TCP scheme should be able to coexist with other TCP variants and not cause them starvation. To measure the friendliness of TCP Jersey, we construct a mixed wired and wireless network where TCP Jersey coexists with Reno. The simulation network is shown in Figure 5.16. The wired link has 100 Mbps bandwidth and 45ms one-way propagation delay, and the wireless link has 20 Mbps and 1 ms delay. There are 20 pairs of connections, of which m are TCP-Jersey connections and n are TCP-Reno connections.

We vary the proportion of these two TCP schemes in the network by adjusting the variables m and n, while keeping the total number of flows to be 20. Without the presence of congestion and link loss, all 20 connections are expected to share the bottleneck bandwidth equally, i.e., roughly 1 Mbps per connection. We first set the link error rate to 0% of packet loss at the bottleneck link and record the throughput of each connection at the bottleneck link. The mean throughput, in unit of Kbps, of TCP Reno and Jersey is calculated by summing up the individual throughputs of the homogeneous TCP flows and divided by the number of connections, respectively. The results are listed in Table 5.2.

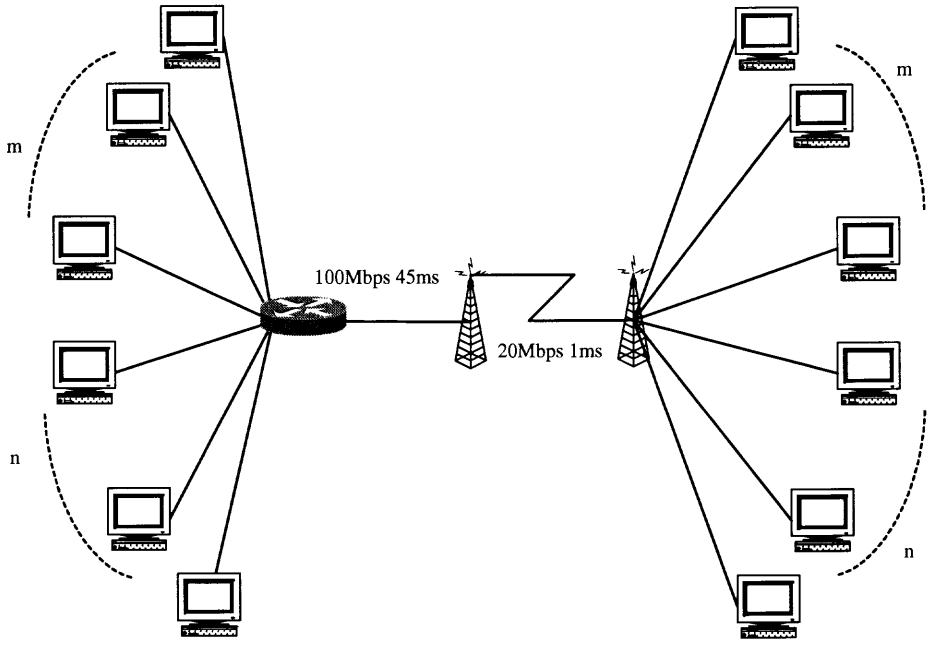


Figure 5.16 Simulation network configuration for friendliness evaluation.

From Table 5.2, it is observed that the bandwidth allocation of each TCP connection is close to its fair share at the bottleneck link.

We next set the link error rate to 0.1% of packet loss. The throughput results are listed in Table 5.3, where the mean throughputs are in units of Kbps. TCP Jersey achieves a slightly higher throughput than Reno when a lossy wireless link exists, but within a tolerable range. The mean throughput of both TCP schemes is still close to the fair share.

We also run the simulation with four different TCP schemes coexisting and competing for the bottleneck wireless link, i.e., coexistence of TCP Reno, Vegas, Westwood, and Jersey, as shown in Figure 5.17.

The random packet loss rate at the wireless bottleneck link ranges from 1% to 5%. The bottleneck link has 8Mbps of bandwidth.

The throughput results are listed in Table 5.4, where the four TCP connections are Reno, Vegas, Westwood, and Jersey; the error rate is in packet loss percentage; and throughputs are in units of Kbps.

Table 5.2 Throughput Comparison Over Error-free Wireless Link

Reno Source	Jersey Source	Mean Throughput (Reno)	Mean Throughput (Jersey)
3	17	961.50	958.79
5	15	959.94	958.95
10	10	959.43	958.97
15	5	959.21	959.16
17	3	959.16	959.40

Table 5.3 Throughput Comparison Over Lossy Wireless Link

Reno Source	Jersey Source	Mean Throughput (Reno)	Mean Throughput (Jersey)
3	17	895.23	965.76
5	15	903.53	972.33
10	10	916.20	993.86
15	5	937.57	1007.40
17	3	944.35	1015.16

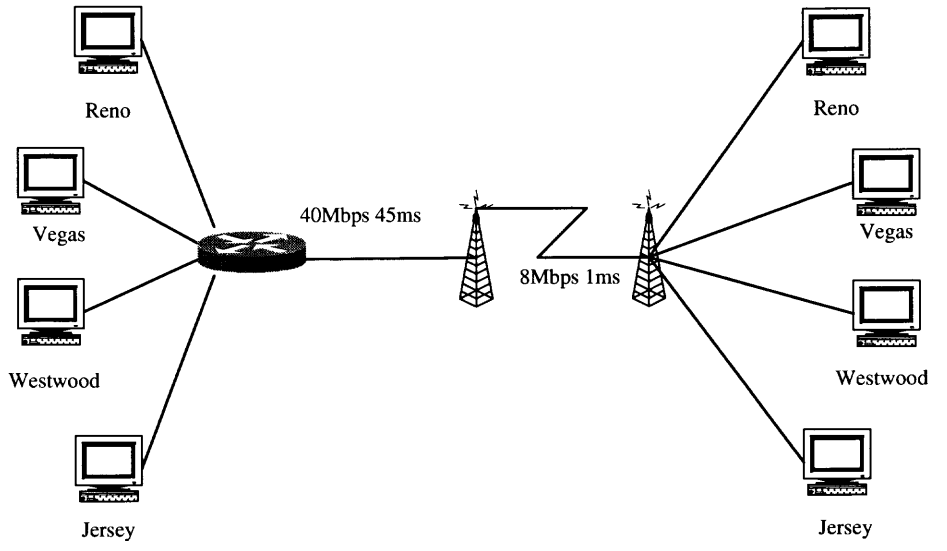


Figure 5.17 Simulation network configuration for coexistence of different TCP schemes.

For the error rate as low as 0.1%, four TCP schemes perform closely. As the error rate increases, TCP Jersey achieves significantly higher throughput than TCP Reno, Vegas and Westwood, conforming to the results obtained from the goodput simulation in Section 5.6.1.

It is also observed that the friendliness of TCP Jersey is within the same range of Westwood. Both Westwood and Jersey behave more aggressively than other non-wireless oriented TCP schemes when the random wireless link error rate is beyond 0.1%. This behavior is anticipated since TCP Jersey is designed to perform better in lossy wireless environment.

Table 5.4 Friendliness Comparison With Four Co-existing TCP Schemes

Error Rate (% packet loss)	Throughput (Reno)	Throughput (Vegas)	Throughput (Westwood)	Throughput (Jersey)
0.0	1664.22	1659.36	1663.98	1663.59
0.1	1158.94	1541.92	1354.50	1623.64
0.5	761.28	936.60	1338.91	1455.36
1.0	552.01	554.78	1077.56	1298.09
5.0	213.76	177.20	419.49	446.40

5.7 Summary

In this chapter, we have proposed a new TCP scheme, called TCP-Jersey, to improve the TCP performance in the heterogeneous network consisting of wired and wireless links.

TCP-Jersey adopts the rate-based congestion control methodology, and employs the achievable rate estimation (ARE) using a modified time-sliding window algorithm at the sender side to optimize the congestion window size when network congestion is detected. TCP-Jersey detects the congestion by means of the congestion warning (CW) mechanism implemented at the router that marks all packets when the router's buffer occupancy exceeds a threshold. The congestion warning mechanism also aids the TCP sender to effectively differentiate packet losses due to random wireless link errors from those caused by link congestion, so that the sender could react to DUPACK intelligently.

The proposed approach is easy to implement. It only requires simple configurations at the ECN-enabled routers and minimum changes to the TCP sender side code without altering the protocol itself.

Simulations in this chapter show that TCP-Jersey is a viable solution to the TCP performance degradation in wireless IP communications. Results from the simulations show that under 1% packet loss rate, a typical characteristic of wireless links, due to its fine-grained congestion window adjustment and the ability of distinguishing the congestion losses from the error losses, TCP-Jersey outperforms other TCP variants by 17% to 85% improvement in goodput when the network is not congested, and 6.15% to 19% goodput improvements when the network is congested by VoIP flows that share the same link. Our experiments also show that TCP-Jersey maintains the fair and friendly behavior to other TCP flows.

CHAPTER 6

STABILITY AND FAIRNESS OF RATE ESTIMATION BASED AIAD CONGESTION CONTROL IN TCP

In this chapter, we analyze two achievable rate estimators (AREs) that use different timestamps of consecutive packets. We examine the effect of the choice of rate estimators to the stability and fairness of a class of TCP protocols that use this estimated rate to implement the additive-increase and adaptive-decrease (AIAD) congestion control. The sender-side algorithm of the TCP-Jersey presented in Chapter 5 and the TCP-Westwood proposed by Casetti et al. [36] are two representatives of such an AIAD congestion control algorithm.

Simulation results in this chapter confirm our analysis that rate estimation based on the *inter-arrival times of the ACK packets* is not properly bounded and would cause instability of the AIAD algorithm and unfairness among competing TCP flows, particularly in networks with small or moderate buffer space. Whereas, the rate estimation based on the *inter-arrival times of the data packet at the receiver* maintains its accuracy even when the reverse path is congested, and enables the AIAD algorithm to maintain the stability and fairness as the number of competing flows increases.

The analysis also suggests a straightforward enhancement to TCP-Westwood and TCP-Jersey that would improve their stability and fairness. The enhanced algorithm can be easily implemented without any modifications to the TCP receiver-side code by enabling the TCP timestamps option.

Unlike the additive-increase and multiplicative-decrease (AIMD) congestion control algorithm [18] in traditional TCP that reduces the sending rate multiplicatively[1] upon packet loss due to congestion, TCP Westwood [36] and the

[1]More precisely, the sender decreases the transmission rate by a *static* multiplicative factor.

sender-side algorithm of TCP-Jersey adjust the sending rate adaptively based on the sender's estimations of the achievable rate or throughput. We call this class of TCP schemes as the achievable rate estimation (ARE) based additive-increase and adaptive-decrease (AIAD) congestion control, or simply AIAD throughout this chapter.

In the rate estimation based AIAD, the sender estimates the *achievable throughput*, or *achievable rate*, by continuously examining the timestamps and patterns of the returning acknowledgment packets (ACKs). When congestion is detected via packet losses, the sender adjusts the transmission rate according to the estimated achievable rate. The evolution of TCP's congestion window, $w(t)$, of such AIAD can be expressed as follows upon each returned ACK or the detection of a packet loss:

$$w(t+1) = \begin{cases} w(t) + \frac{1}{w(t)}, & \text{no congestion} \\ r(t)d, & \text{congestion loss} \end{cases}, \tag{6.1}$$

where $r(t)$ is the TCP's estimation of the achievable rate, and d is the round-trip propagation delay. It is clear from (6.1) that, in the absence of congestion, the sender's congestion window increases by $\frac{1}{w(t)}$ for every successfully delivered packet, or equivalently, an increase of 1 for every round-trip time (RTT), and hence, the additive increase (in RTT). On the other hand, if packet loss due to congestion is detected, the sender sets the congestion window according to the estimated achievable rate as $r(t)d$, i.e., the adaptive decrease.

It is obvious that over estimation of $r(t)$ above the bottleneck link capacity would cause adverse effect to the congestion condition. In Section 6.1, we analyze two rate estimators that use different timestamps. In Section 6.2, we establish a system model of the rate estimation based AIAD algorithm and study its conditions of stability when different rate estimators are used. Section 6.3 presents the

simulation results that confirm our analysis, and we summarize our analysis in Section 6.4.

6.1 Achievable Rate Estimations

We consider a simple network connected via bi-directional links, where there is only one bottleneck link, on which there are N TCP flows, indexed by $i = 1, 2, \ldots, N$, in the same direction contending for its capacity. [2] The bottleneck link is characterized by its forward capacity C, backward capacity γC, forward queue occupancy $q_f(t)$ as seen by the data packet arriving at the queue, and backward queue occupancy $q_b(t)$ as seen by the arriving ACK at the backward bottleneck queue. Packets are indexed by $n = 1, 2, \ldots$. A reception of a packet at the destination causes an *immediate* transmission of a corresponding ACK. [3]

As illustrated in Figure 6.1, associated with packet n are three timestamps, t_n, t'_n, and t''_n representing the time it is sent from the source, the time it departs from the *forward* bottleneck queue, and the time its corresponding ACK departs from the *backward* bottleneck queue, respectively, where τ_1^f, τ_2^f, and τ_1^b, τ_2^b are the propagation delays before and after the bottleneck queues in forward and backward paths, respectively. We assume all data packets have the same size of B bits, and the ACK packets are of size ρB bits, where $\rho < 1$.

Consider two consecutive packets, packet $n-1$ and packet n; for notational convenience, we define the following inter-packet time intervals $\Delta t_n = t_n - t_{n-1}$, $\Delta t'_n = t'_n - t'_{n-1}$, and $\Delta t''_n = t''_n - t''_{n-1}$.

[2] In the case of an end-to-end path along which there may be multiple bottlenecked links, the above network can be considered as a simplified lump-sum network model, where multiple bottlenecked links are lumped into one bottleneck link on which a packet experiences the most severe congestion.

[3] The TCP Delayed Acknowledgment algorithm [69] in some TCP implementations can be easily incorporated into the analysis with no major impact to the results.

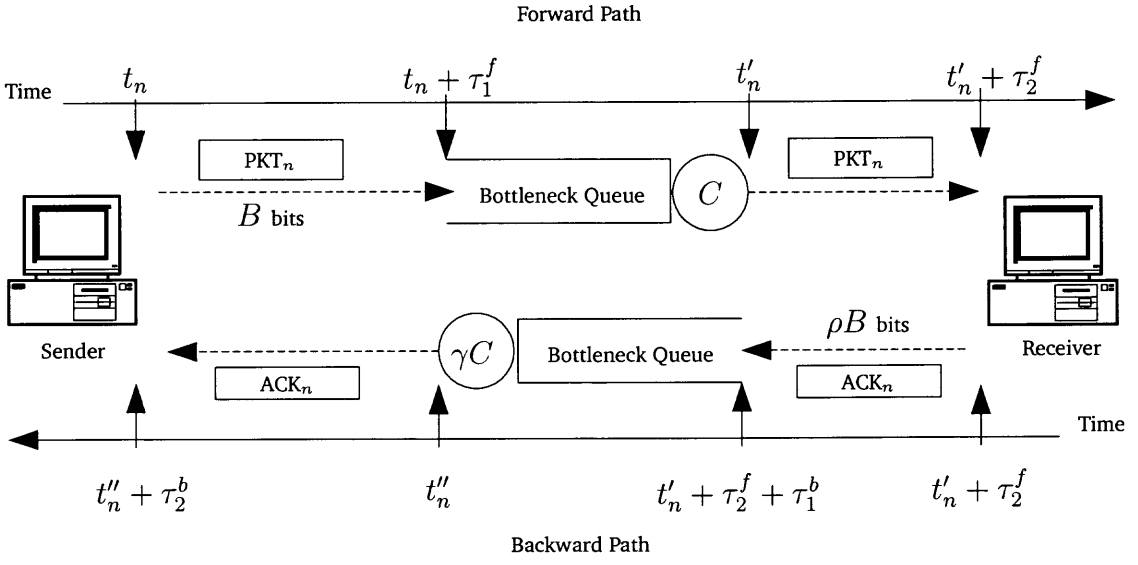


Figure 6.1 The passage of a data packet and the corresponding ACK packet, and the definitions of timestamps.

At each instance of either a transmission of a data packet or a reception of a data or an ACK packet, there are three instantaneous rate samples, measurable either at the source or at the destination, that indicate the achievable throughput or rate of the flow.

The three measurable instantaneous rate samples are,

Sending Rate : $x_{i,n} = \frac{B}{\Delta t_n}$ is an estimate sample, measurable at the source, based on the inter-transmission time between two consecutive data packets sent by the source;

Receiving Rate : $\varphi_{i,n} = \frac{B}{\Delta t'_n}$ is an estimate sample, measurable at the source by examining TCP timestamps option field [70] in the ACK packets. It is based on the inter-arrival time of the two consecutive data packets arrived at the destination;

ACK Rate : $\kappa_{i,n} = \frac{B}{\Delta t''_n}$ is an estimate sample based on the inter-arrival time of the returning ACKs measured at the source. It indicates the amount of *data* acknowledged during the interval.

We use $x_i(t)$, $\varphi_i(t)$, and $\kappa_i(t)$ to denote the underlying rate processes which could be constructed by low-pass filtering of the corresponding samples (see discussions in Chapter 5 regarding the low-pass filters used in TCP-Jersey and TCP-Westwood, respectively). The queuing processes at the forward and backward bottleneck links are

$$\dot{q_f}(t) = y_f(t) - C, \text{if } q_f(t) > 0, \tag{6.2}$$

and

$$\dot{q_b}(t) = y_b(t) - \gamma C, \text{if } q_b(t) > 0, \tag{6.3}$$

respectively, where $y_f(t) = \sum_{i=1}^{N} x_i(t)$ is the aggregated intensity of the data flows, and $y_b(t) = \rho \sum_{i=1}^{N} \varphi_i(t)$ is the aggregated intensity of the ACK flows, since the receiver *immediately* acknowledges the data packet upon reception.

Define $\Delta q_f(n) = \left[q_f(t_n + \tau_1^f) - q_f(t_{n-1} + \tau_1^f)\right]$ as the difference of the queue lengths seen by the two consecutive packets when they arrive at the bottleneck; we have

$$t'_n = \left(t_n + \tau_1^f\right) + \frac{q_f(t_n + \tau_1^f) + B}{C}, \tag{6.4}$$

and

$$\Delta t'_n = \Delta t_n + \frac{\Delta q_f(n)}{C} = \left(1 + \frac{\dot{q_f}(t)}{C}\right) \Delta t_n. \tag{6.5}$$

Here we have used the approximation $\Delta q_f(n) \approx \dot{q_f}(t) \Delta t_n$, when Δt_n is small. It is easy to derive from (6.5) and the definitions of $x_{i,n}$ and $\varphi_{i,n}$ that $\varphi_{i,n} = \frac{x_{i,n}}{\sum_{i=1}^{N} x_i(t)} C$, and hence, by low-pass filtering of the samples, we have

$$\varphi_i(t) = \frac{x_i(t)}{\sum_{i=1}^{N} x_i(t)} C. \tag{6.6}$$

Also, $\varphi_i(t) \leq C$, that is, the φ estimator is upper bounded by the bottleneck link's capacity. This is obvious from the fact that $x_i(t) \leq y_f(t)$. It can also be derived from

the *capacity constraint* that $t'_n - t'_{n-1} \geq B/C$, which, in turn, according to (6.5), leads to

$$\Delta q_f(n) \geq B(1 - \frac{C}{x_{i,n}}). \tag{6.7}$$

Applying the inequality (6.7) to the relationship

$$\frac{B}{\varphi_{i,n}} = \frac{B}{x_{i,n}} + \frac{\Delta q_f(n)}{C}, \tag{6.8}$$

we have, after algebraic simplifications, the bound $\varphi_{i,n} \leq C$.

Taking into the account that the reception of a B-bit data packet at the destination causes an immediate transmission of a ρB-bit ACK packet, which in turn acknowledges the delivery of a B-bit data packet, we can derive by symmetry that, on the backward path, the ACK rate sample is related to the receiving rate sample as $\kappa_{i,n} = \frac{\varphi_{i,n}}{\sum_{i=1}^{N} \varphi_i(t)} \frac{\gamma C}{\rho}$, and henceforth,

$$\kappa_i(t) = \frac{\varphi_i(t)}{\sum_{i=1}^{N} \varphi_i(t)} \frac{\gamma C}{\rho}. \tag{6.9}$$

Notice that $\kappa_i(t) \leq \gamma C/\rho$. The bound for the samples $\kappa_{i,n}$ can also be similarly derived by applying the *capacity constraint* of the reverse bottleneck link. That is, $\Delta t''_n \geq \frac{\rho B}{\gamma C}$ leads to $\kappa_{i,n} \leq \gamma C/\rho$. It is this upper bound that would lead to over estimation of the forward bottleneck capacity in case of reverse path congestion. This will become clear as we will see in Section 6.3.

In summary, the rate estimator, $\varphi_i(t)$, which is based on the packet receiving timestamps, is always upper bounded by the capacity of the *forward* bottleneck link of the end-to-end path; whereas estimate based on the inter-arrival time of ACK packets, the $\kappa_i(t)$ estimator, is limited above by the *backward* bottleneck link's capacity multiplied by the ratio of the ACK packet size and the data packet size. $\kappa_i(t)$ could take a value much larger than the capacity of the *forward* bottleneck link, i.e., an over estimation of the bottleneck link capacity.

6.2 Stability of Rate Estimation Based AIAD

In this section, we model the TCP dynamics, in particular, the evolution of the TCP source's sending rate, according to the AIAD algorithm (6.1). We discuss the stability and fairness issues of TCP flows when different rate estimators are used in (6.1), i.e., $r(t) = \kappa(t)$, vs. $r(t) = \varphi(t)$.

In the network model described in previous section, for flow i, let τ_i be the round-trip-time (RTT), d_i be the round-trip propagation delay, which is taken to be a fraction of the RTT as $d_i = \eta_i \tau_i$, and $0 < \eta_i \leq 1$. We also introduce the approximate relationship between the TCP source's sending rate and the congestion window size as

$$x_i(t) \approx \frac{w_i(t)}{\tau_i}. \tag{6.10}$$

This is the approximation widely adopted in many literature regarding TCP modeling. It is accurate in the time scale of RTT, since the window-based TCP congestion control limits (roughly) the number of packets sent by the sender in one RTT to be no more than its congestion window ($cwnd$)

Let $p(t) = p\left(y_f(t)\right)$ be the probability that a packet is lost due to congestion; it is a non-negative, non-decreasing function of the aggregated flow rate on the bottleneck link. Following the modeling method outlined in [62], since the TCP source increases the window size by 1 for every RTT and decreases the window size to $r_i(t)d_i$ if a packet loss is detected, with (6.10), the AIAD algorithm (6.1) can be translated into the following system of nonlinear delay differential equations

$$\begin{aligned} \dot{x_i}(t) &= \frac{1}{\tau_i^2} - [x_i(t) - \eta_i r_i(t)] x_i(t-\tau_i) p\left(y_f(t-\tau_i)\right), \\ \dot{q_f}(t) &= y_f(t) - C. \end{aligned} \tag{6.11}$$

In what follows, we study the choice of $r_i(t)$ and its effect on the stability of system (6.11). For clarity, subscripts of the flow index i are dropped.

Let $(x_0, q_{f,0})$ be the equilibrium of system (6.11); it can be derived by the fixed point that $x_0 = \varphi_0 = C/N$, and $\kappa_0 = \gamma C/\rho N$. Define $p_0 \doteq p(t)|_{x_0}$, and $p_0' \doteq \frac{\partial p}{\partial x}|_{x_0}$, linearizing of (6.11) around the equilibrium yields a linear system with time delay in the form of

$$\dot{\delta x}(t) = K_1 \delta x(t) + K_2 \delta x(t - \tau), \tag{6.12}$$

where K_1 and K_2 are to be determined by the choice of $r(t)$, and δx denotes the small deviation of x around the equilibrium , i.e., $\delta x = x - x_0$. See Appendix A.1.1 for the detailed derivation.

Taking the Laplace transform of (6.12), followed by a substitution of $\lambda \doteq s\tau$, we have

$$\tau K_1 e^{\lambda} + \tau K_2 - \lambda e^{\lambda} = 0 \tag{6.13}$$

as the characteristic equation of (6.12). For convenience, we restate the following lemma by Johari and Tan (see [71], Lemma 2, and also [72])

Lemma 6.1 *All the roots of $be^{\lambda} + c - \lambda e^{\lambda} = 0$, where b and c are real, have negative real parts if and only if: 1) $b < 1$, and 2) $b < -c$, and 3) $-c < \sqrt{a_1^2 + b^2}$, where a_1 is the root of $a = b \tan a$ such that $0 < a < \pi$. If $b = 0$, we take $a_1 = \pi/2$.*

Case 1: The rate estimation in the AIAD algorithm adopts the receiving rate, i.e., $r_i(t) = \varphi_i(t)$. We call this algorithm AIAD$_\varphi$. This can be implemented using the TCP's timestamps option defined in RFC 1323 [70], by which the receiver stamps the packet arrival time into the header of the corresponding ACK packet. The source, by extracting the timestamps from the ACK packets, can compute $\varphi_{i,n}$ and construct $\varphi_i(t)$ by some low-pass filters such as those discussed in Chapter 5. In this case, we have $K_1 = \left(\frac{N-1}{N^2}\eta - \frac{1}{N}\right) C p_0$ and $K_2 = \frac{\eta - 1}{N} C \left(p_0 + \frac{C}{N} p_0'\right)$. It is easy to verify that (6.13) satisfies the conditions 1) and 2) of Lemma 6.1, regardless of the delay.

It can also be shown (see Appendix A.1.2 for detailed derivations) that as long as the round-trip queuing delay $d_q = \tau - d$ satisfies

$$d_q < \frac{\pi N}{2C(p_0 + \frac{C}{N}p_0')}, \tag{6.14}$$

due to condition 3), the characteristic equation of (6.12) has all its roots with negative real parts, and hence, AIAD$_\varphi$ is locally stable.

Case 2: The AIAD algorithm adopts the ACK rate as its rate estimation, i.e., $r_i(t) = \kappa_i(t)$, which we call algorithm AIAD$_\kappa$. An example implementation of this algorithm is TCP Westwood [36], where the source examines the inter-arrival time of the ACK packets to compute the rate estimate samples $\kappa_{i,n}$, which are then fed to a low-pass filter to construct $\kappa_i(t)$.

In this case, we have (again, see Appendix A.1.2 for detailed derivations and justifications) $K_1 = \left(\frac{N-1}{N^2}\alpha\eta - \frac{1}{N}\right) Cp_0$, $K_2 = \frac{\alpha\eta - 1}{N} C\left(p_0 + \frac{C}{N}p_0'\right)$, where $\alpha = \gamma/\rho$.

Subjecting (6.13) to the stability conditions of Lemma 6.1, we can show that condition 2) requires that.

$$\alpha\eta \doteq \frac{\gamma}{\rho}\frac{d}{d + d_q} < \frac{2N + \frac{p_0'}{p_0}C}{2N + \frac{p_0'}{p_0}C - 1}, \tag{6.15}$$

and condition 3) requires

$$d_q < \frac{\pi N}{2C(p_0 + \frac{C}{N}p_0')} + (\alpha - 1)d. \tag{6.16}$$

Since γ, ρ, and d are fixed by the network configurations and the implementation of the TCP protocol, (6.15) implies that for system (6.12) with AIAD$_\kappa$ to remain stable as the number of flows increases, the network has to increase its buffer size, or, in other words, with AIAD$_\kappa$, fixed buffer size would lead to instability as the number of flows grows.

6.3 Validation of Analysis

We use the packet level network simulator, NS-2 [25], to verify our analysis presented in the previous sections. We also discuss the impact to the fairness among competing TCP flows when different rate estimator is used in the AIAD congestion control.

We use the TCP Westwood, that adopts ACK rate, i.e., $r_i(t) = \kappa_i(t)$, for its rate estimator, as the representative of the AIAD$_\kappa$ algorithm. To facilitate our study, we have also modified several lines of the Westwood code so that it uses the receiving rate, i.e., $r_i(t) = \varphi_i(t)$, as the rate estimator, and named our modified version as the AIAD$_\varphi$ algorithm.

Our simulations are conducted on a simple network, depicted in Figure 6.2, that consists of one symmetric bottleneck link with capacity of 3Mbps and buffer size of 20 packets in both forward and backward directions. A varying number of TCP flows traverse from the sources to the destinations in the forward direction via their respective 100Mbps access links to the bottleneck. To simulate the congestion at the backward bottleneck link, an ON/OFF constant-bit-rate (CBR) flow is devised to run in the backward direction.

All data packets are 1000 bytes in size, and since ACK packets are 40 bytes in size, our settings yield $\gamma = 1$ and $\rho = 0.04$. All links adopt the FIFO service discipline.

All simulations are run for 500 seconds, during which, starting from time 50s, the 2Mbps CBR flow is set to be ON and OFF for 50 seconds alternately.

In the first simulation, there is only one TCP flow in the forward direction. We run the simulation for the AIAD$_\kappa$ and AIAD$_\varphi$ algorithms separately, and plot the TCP's throughput as well as its rate estimation in Figure 6.3 and Figure 6.4, respectively.

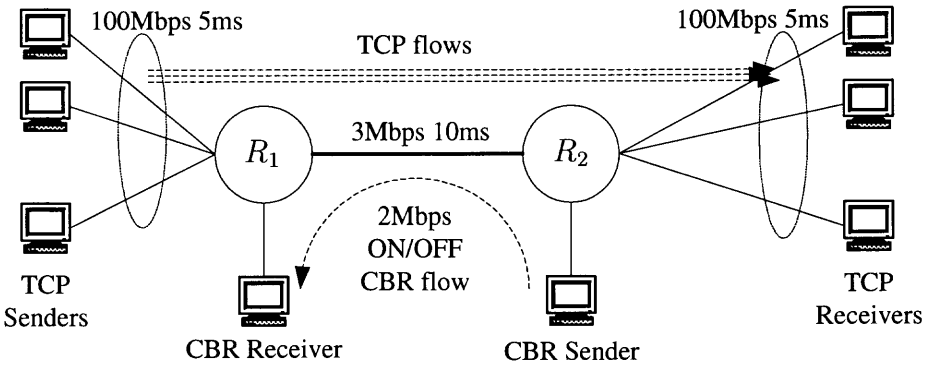


Figure 6.2 Simulation network configuration.

The results agree with our analysis in Section 6.1 that rate estimator $\kappa_i(t)$ exhibits significant over estimation during the periods when the backward bottleneck is congested by the CBR flow, thus leading to fluctuations of the throughput. This is because the over estimation of the achievable rate causes the sender to overflow the bottleneck buffer, and consequently, induces more packet losses.

On the other hand, algorithm AIAD$_\varphi$ (i.e., Case 2), using $\varphi_i(t)$ as the rate estimator, results in a more accurate estimate of the achievable rate, and hence, a more stable throughput. It is thus obvious that $\varphi_i(t)$ is more robust when ACKs are subject to congestion and loss.

The instability of the system (6.11) manifests itself in the fluctuations and uneven distribution of the throughputs among the flows. To study this, we conducted a second set of simulations. In the second set of simulations, we increase the number of TCP flows from 5 to 200 and observe the throughput distributions.

Let $\hat{x}_i$ be the flow i's measured throughput; we define the *throughput ratio* of flow i as $\hat{x}_i/x_0$. Figure 6.5 plots the standard deviation of the throughput ratio as N increases. It shows that as N increases, algorithm AIAD$_\kappa$ leads to a much more diverse throughput distribution than does algorithm AIAD$_\varphi$.

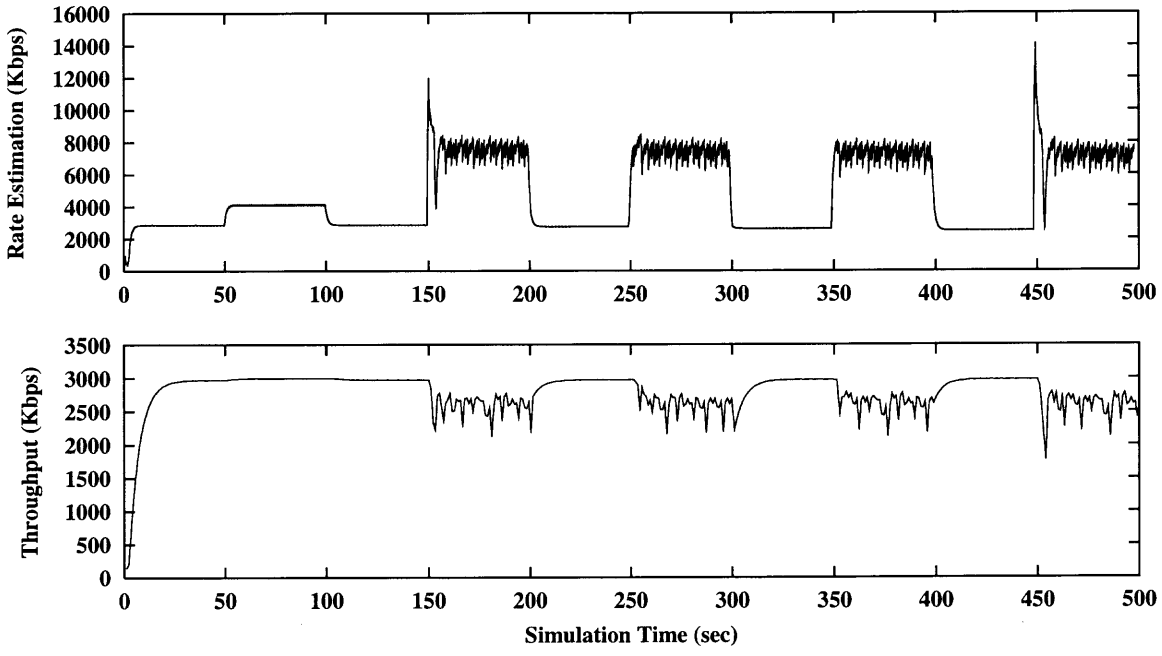


Figure 6.3 Throughput and rate estimation of a single TCP flow using AIAD$_\kappa$ passing through the network with congestion at the backward bottleneck link caused by the ON/OFF CBR traffic. The over estimation corresponds to the ON periods of the CBR flow, and the throughput fluctuates accordingly.

From the same set of simulations, we also plot, in the same graph, the Jain's fairness index ϕ as defined in [19]. $\phi = 1$ indicates a perfectly even distribution of resource, and $\phi = \frac{1}{N}$ implies that all but one flow gets the entire resource. A stable congestion control algorithm should result in a fair and stable ϕ as N increases. Figure 6.5 confirms our analysis in Section 6.2 that AIAD$_\kappa$ becomes unstable and unfair when N increases.

Also, recall (6.15), in the limiting case, we have $\alpha\eta < 1$. In our simulation settings, this is equivalent to $Q > 24Cd$, where Q is the total combined buffer size of the forward and backward bottleneck links. We have also verified by simulations, see Figure 6.5, that the performance of AIAD$_\kappa$ with buffer size of 200 packets becomes close to that of AIAD$_\varphi$ with buffer sizes of 20 packets.

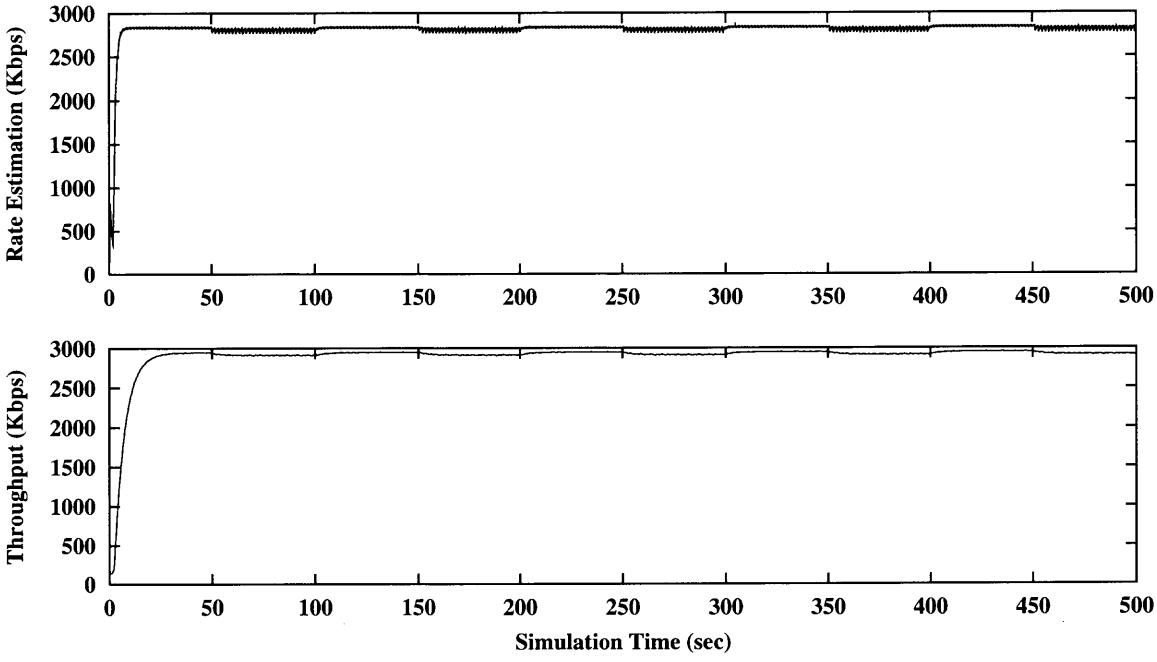


Figure 6.4 Throughput and rate estimation of a single TCP flow using AIAD$_\varphi$ passing through the network with congestion at the backward bottleneck link caused by the ON/OFF CBR traffic. No over estimation is observed, and the throughput is much more stable.

6.4 Summary

In this chapter, we have analyzed and verified via packet level simulations that, in the TCP protocol using the rate estimation based AIAD algorithm, the ACK rate estimator $\kappa_i(t)$ is not upper bounded by the forward bottleneck capacity and will result in significant amount of over estimation when the backward path experiences congestion; AIAD$_\kappa$ tends to become unstable and unfair as the number of competing flows increases. Whereas, the receiving rate estimator $\varphi_i(t)$ is robust to reverse congestion; when used in the AIAD algorithm, the system maintains its stability and fairness as the number of flows increases.

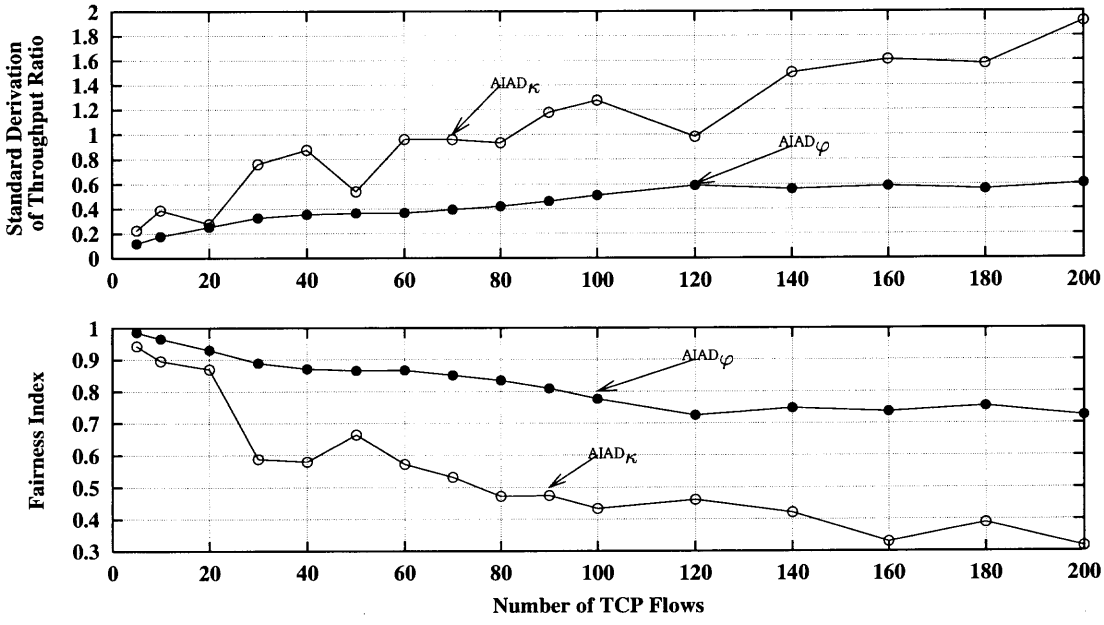


Figure 6.5 Standard deviation of throughput ratio $\hat{x}_i/x_0$, and fairness index $\phi = (\sum \hat{x}_i)^2/N \sum \hat{x}_i^2$. The buffer size is 20 packets.

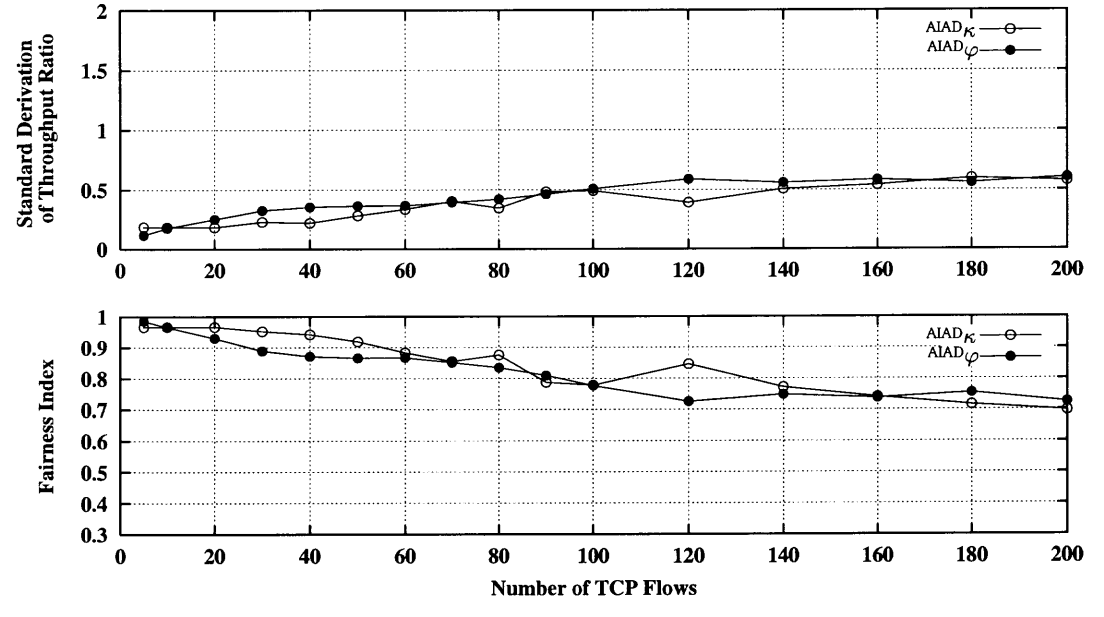


Figure 6.6 Standard deviation of throughput ratio $\hat{x}_i/x_0$, and fairness index $\phi = (\sum \hat{x}_i)^2/N \sum \hat{x}_i^2$. The performance curves of the AIAD$_\kappa$ is obtained when the buffer size is increased to 200 packets.

CHAPTER 7

ROBUST ACHIEVABLE RATE ESTIMATION

The first key component of the proposed end-to-end solution architecture to TCP's performance degradation in wireless networks is the achievable rate estimation (ARE). In TCP-Jersey, the first integrated implementation of the architecture, ARE is implemented based on the inter-arrival times of the returning ACK packets, which is referred to, in this Chapter, as the original ARE, or simply ARE.

The analysis and verifications in Chapter 6 provides a straightforward enhancement to the TCP-Jersey's ARE algorithm. That is to use the inter-arrival times of the *data* packets instead of the inter-arrival times of the ACK packets.

In this chapter, an enhanced version of the TCP-Jersey's ARE algorithm is evaluated. In this enhanced ARE algorithm the TCP's timestamps option defined in RFC 1323 is utilized for computing the achievable rate. The enhanced ARE algorithm is referred to as TS-ARE.

As it has been shown in Chapter 6, ARE[1] is accurate only if the reverse end-to-end path of the TCP connection can be assumed ideal, meaning that ACKs are not congested or lost on their way back to the sender. However, this assumption usually does not hold in reality due to link asymmetry and bandwidth sharing among two-way traffic. When the reverse end-to-end path is also subject to congestion, ARE will be likely to over-estimate the connection's achievable rate and adversely affect the stability and robustness of the congestion control algorithm, i.e., the rate estimation based AIAD algorithm of TCP-Jersey. The inter-ACK gap based rate estimation (i.e., ARE) is unsuitable for real-world applications, and particularly in wireless networks, due to the burstiness of TCP traffic and the possible link asymmetry.

[1]It is the $\kappa(t)$ in Chapter 6

7.1 Burstiness of ACK

TCP flow control is self-clocking. In essence, the arrivals of the ACKs at the sender trigger the sending of new packets and advances of the congestion window. TCP traffic is busty in nature. The arrival of an ACK that *cumulatively* acknowledges the previously unacknowledged data packets triggers the transmissions of multiple new data packets in a back-to-back fashion. ACK compression [73, 74], a phenomenon caused by the queuing in the reverse path of a TCP flow, can cause the almost instantaneous arrivals of bursts of ACKs at the sender. This can break TCP's self-clocking and cause long bursts. Suppose several ACKs have been lost between two successfully received ACKs, the sender will send out a number of packets, indicated by the newly received ACKs, back to back. This burdens the forward path with an abrupt load increase and exacerbates the forward path condition. The effect of this problem can be devastating to some rate-based TCP schemes, such as TCP Westwood and TCP-Jersey. Usually, such schemes assume that short intervals between ACKs received at the sender imply a large available bandwidth in the forward path, since it is assumed that the ACK stream on the reverse path preserves the inter-packet dispersion on the forward path. The TCP self-clocking mechanism would therefore be influenced, and the sender would make incorrect judgments of the forward path.

7.2 Link Asymmetry

In networks with asymmetric links, which may be common in wireless networks, TCP's performance in terms of the throughput also largely depends on the characteristics of the reverse link. Asymmetric links can be results of different physical designs of the forward and backward communication channels, or imbalanced network traffic in both directions. In the 3G cellular networks, multimedia data transmission is carried through the wireless channel, and so are the conventional

voice data. The traffic on the down-link from the base station to the mobile device tends to consume more bandwidth than the traffic on the up-link practically. Also the transmission power of a base station is much higher than a mobile device. 3G wireless networks consider these facts into its design and can possibly control the link asymmetry, e.g., via pricing policy. The asymmetric link problem results in rate and delay variation. In a data path, the reverse channel may suffer from packet loss even when the condition on the forward channel is moderate. The effect of this problem can be devastating to the design of some TCP schemes since most TCP schemes assume the packet loss occurs in the forward channel. The TCP self-clocking mechanism based on the ACK feedbacks would therefore be influenced and even be dominated by the poor condition on the reverse channel. The sender would make incorrect judgments on the forward channel condition, thus exacerbating the adverse effect on the reverse channel.

7.3 Implementation and Performance Evaluation of TS-ARE

To improve the robustness, in this section, we make use of the standard TCP's timestamps option defined in [70] and implemented in many TCP variants as RFC 1323, and re-implement the TCP-Jersey's ARE estimator using the inter-arrival time of *data* packets at the receiver [2]

The essential operation of RFC 1323 can be summarized as follows. In the TCP header, the timestamps option flag is set. Upon the transmission of each data packet, the sender stamps its current time[3] into the TCP header. At the packet's destination, the receiver examines the timestamps option flag. If the flag is set, the receiver copies the flag as well as the sender's timestamps into the TCP header of the corresponding ACK packet. In addition, the receiver stamps into the TCP

[2]This is measurable at the sender, and therefore, no receiver side code change is necessary.
[3]In terms of the number of seconds from the "Epoch", i.e., 00:00:00, GMT, January 1, 1970

header its own reading of the current time. Therefore, upon receptions of two ACKs, the TCP sender is able to extract the receiver's timestamps from the TCP header and computer the inter-arrival time of the two data packet at the receiver, and from which the achievable rate in the forward direction can be computed. This is essentially the $\varphi(t)$ estimator discussed in Chapter 6.

The implementation of this enhanced ARE, called TS-ARE, is thus simple. As compared to the original ARE in (5.9), the rate estimation update in TS-ARE becomes

$$R_k = \frac{\tau_k \times R_{k-1} + L_k}{(t'_k - t'_{k-1}) + \tau_k}, \tag{7.1}$$

where t'_k is the receiver's timestamps value extracted by the sender from the k^{th} ACK, instead of the actual arrival time of the ACK at the sender. As with (5.9), τ_k is the sum of the RTT and its variation at the time of the k^{th} ACK's arrival.

The original ARE and the enhanced TS-ARE are tested in a simulation network as depicted in Figure 7.1.

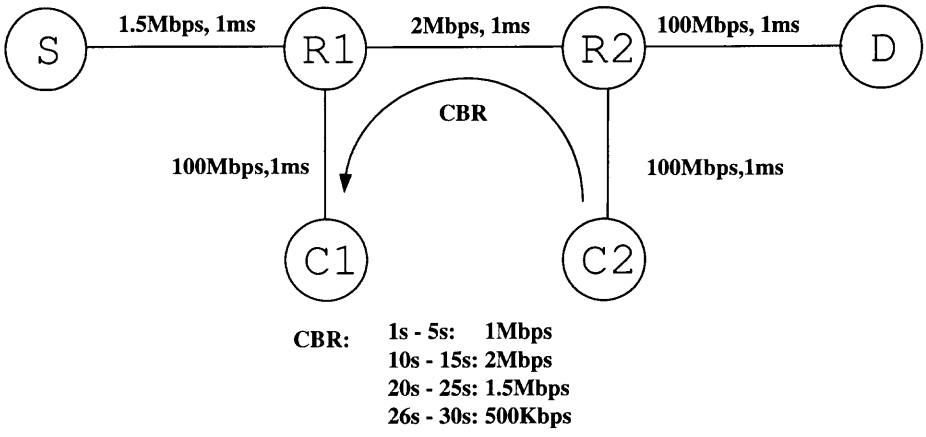


Figure 7.1 Network configuration for comparing ARE and TS-ARE.

The two rate estimators, i.e., the original ARE and the enhanced TS-ARE, implemented in TCP-Jersey, are tested in the simulation network, respectively. The TCP-Jersey sender-side code is modified in such a way that the choice of ARE or

TS-ARE can be made by a configuration switch. The simulation network consists of four end nodes, S, C1, C2, and D; and two routers, R1 and R2. The bottleneck link of the network is the 1.5Mbps link from S to R1. All links in the network are bidirectional and configured with a buffer size of 50 packets. A long-live FTP application with the packet size of 1000 bytes runs from node S to node D from the beginning of the simulation. The FTP is carried by TCP-Jersey.

A constant bit rate (CBR) traffic is injected from C2 to C1, traversing link R2 to R1, which is part of the reverse path of the TCP-Jersey flow from S to D. The rate of the CBR flow varies in time. From 1s to 5s, the CBR traffic has a rate of 1Mbps; it stops for 5 seconds and resumes at 10s for 5 seconds with a rate of 2Mbps. The CBR traffic resumes from 20s through 25s with a rate of 1.5Mbps, and then from 26s through 30s with a rate of 500Kbps.

Figure 7.2(a) and Figure 7.2(b) show the output from the achievable rate estimators of ARE and TS-ARE, respectively, where the adaptive time-sliding window (ATSW) filter (5.17) is used. It is observed from both figures that from time 10s through 15s, since the reverse link between R2 and R1 is severely congested by the 2Mbps non-responsive CBR flow, both ARE and TS-ARE estimations show a significant drop in the end-to-end achievable rate. This is the correct behavior since TCP is self-clocking and congestion experienced by the ACK packets will hold up the transmission of new data packets from the sender, and hence lowering the throughput.

It is also noticeable in Figure 7.2(a) that slight fluctuations occur during other periods when the CBR flow is active, e.g., from 1s to 5s, and from 26s to 30s. These periods correspond to the periods of slight and transient congestion in the reverse direction. When the TCP flow reaches 1.5Mbps throughput, which is the capacity of the bottleneck link of the end-to-end path, the returning ACK stream constitutes

a flow of 60Kbps in average, because the receiver acknowledges every data packet and the ACK packet size is 40 bytes while the data packet size is 1000 byte.

Therefore, in the intervals between 1s and 5s, and that between 26s and 30s, during which the reverse link between R2 and R1 is partially occupied by the CBR flow, the remaining average bandwidth left for the ACK flow is 1Mbps and 1.5Mbps, respectively. On the average, these left-over bandwidth of the reverse link would cause no congestion to the ACK flow of 60Kbps. However, as explained before, TCP flows are bursty in nature, transient congestion (queue build-up) does occur, resulting in possible back-to-back queuing of multiple ACK packets between R2 and R1, and hence the possible, though slightly, ACK compression.

As a result, the original ARE which uses the inter-arrival gaps of ACKs is affected by the slight ACK compression and results in over-estimation slightly over the 1.5Mbps line. However, the fluctuations are smoothed by the adaptive filtering of ATSW as was explained in Section 5.2. On the other hand, the TS-ARE output, plotted in Figure 7.2(b), is immune to ACK compression, and thus shows smooth estimation slightly below the 1.5Mbps line.

The most noticeable difference between the outputs of ARE and TS-ARE, i.e., Figure 7.2(a) and Figure 7.2(b), respectively, is the over-estimation of 33% by ARE at 15s corresponding to the sudden disappearance of the congesting CBR flow, and the abrupt burst of ACKs arrive at the sender.

In Chapter 5, a different adaptive low-pass filter, the filter type 2, (5.21), was introduced as a naive modification to the original Westwood's non-adaptive filter (5.7). In what follows, the filter type 2 (5.21) is implemented in both ARE and TS-ARE, and tested under the same conditions as set above.

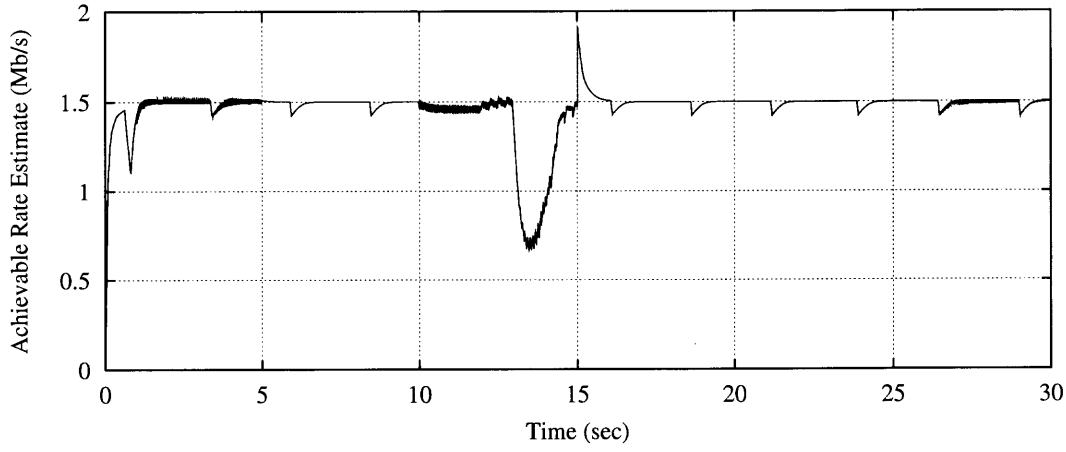
Figure 7.3 plots the outputs of the original ARE as well as that of the enhanced TS-ARE when both are using the naive adaptive low-pass filter (5.21). Comparing to Figure 7.2, the TS-ARE estimators in both cases produce almost identically correct

achievable rate estimates. On the other hand, the ARE estimator (Figure 7.3(a)) in this case produces an even worse result than that seen in Figure 7.2(a). The over-estimation at 15s is much more severe, and there are also significant over-estimations in period between 1s and 5s when the reverse bottleneck has only slight transient queue build-up.

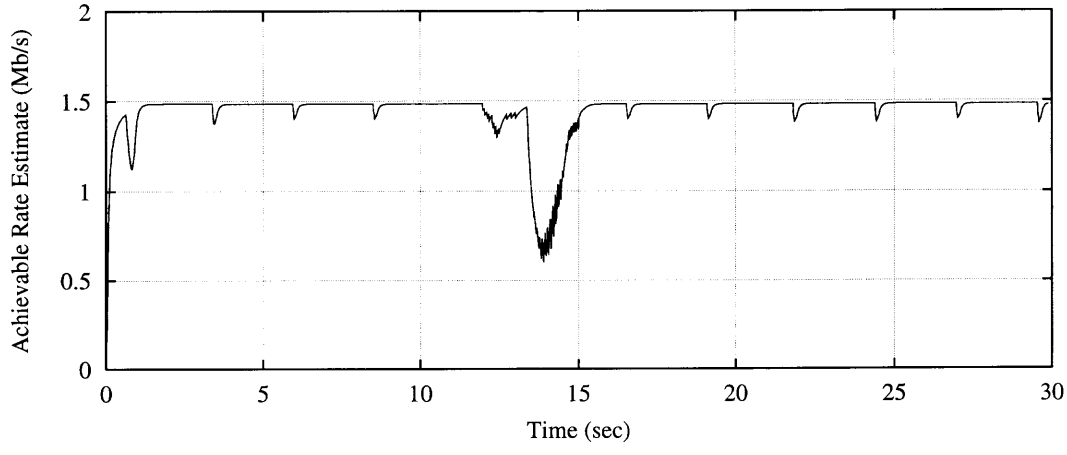
To provide a complete comparison, in Figure 7.4, the output of the "bandwidth estimations" (BWE) proposed and implemented in TCP Westwood is presented. The output is obtained by replacing the TCP-Jersey flow with the TCP Westwood flow [4]. The authors of [36] have provided two versions of TCP Westwood implementation in ns-2 simulator. The original one is based on TCP Reno [20], and the latest one is derived from TCP Newreno [39] (hence named TCP Westwood-NR). Both versions of TCP Westwood use the inter-arrival gaps of the ACK packets in calculating the "available bandwidth", and thus both are the $\kappa(t)$ estimator as discussed in Chapter 6. The TCP westwood-NR uses a slightly different low-pass filter that has more smoothing weights. As seen in Figure 7.4, both BWE estimators produce significant over-estimations during the periods of reverse path congestion. Most importantly, neither BWE estimator correctly outputs the reduction of the end-to-end achievable rate during the period between 10s and 15s, when the reverse path is severely congested by the non-responsive CBR flow.

Therefore, the enhanced TS-ARE with the adaptive filter type 1, (5.17), is the preferred choice for the implementation of TCP-Jersey. In the rare case where the TCP timestamps option, i.e., RFC 1323, is not implemented or available, ARE has to be used in the implementation. In this scenario, it is suggested that the adaptive filter type 1, i.e., (5.17), is used.

[4]The ns-2 implementations of TCP Westwood by the authors of [36] can be found and downloaded from *http://www.cs.ucla.edu/NRL/hpi/tcpw/index.html*.

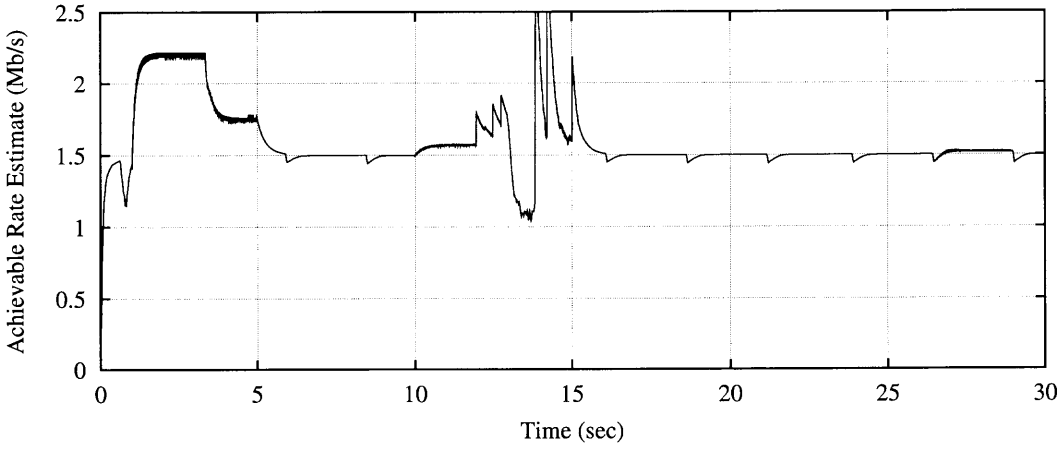


(a) Original ARE.

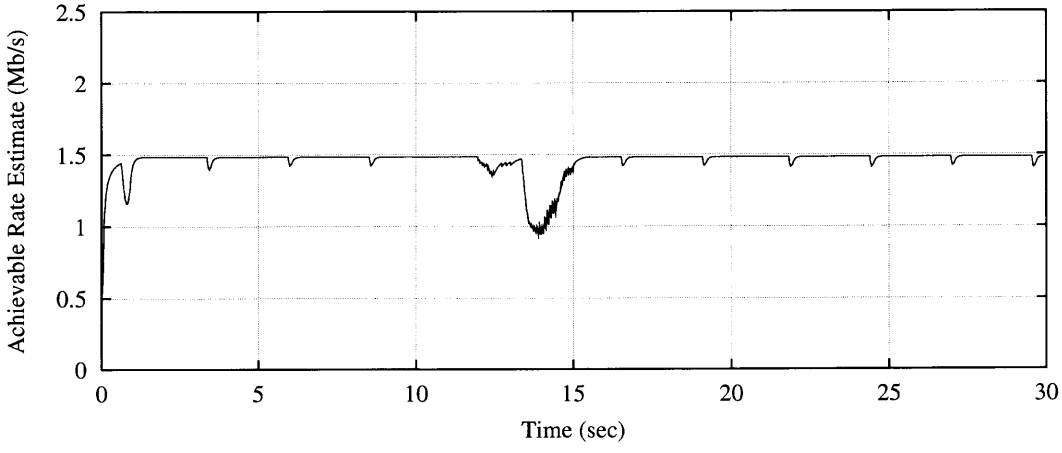


(b) Enhanced ARE.

Figure 7.2 Performance of original ARE vs. enhanced TS-ARE, using adaptive TSW (filter type 1).

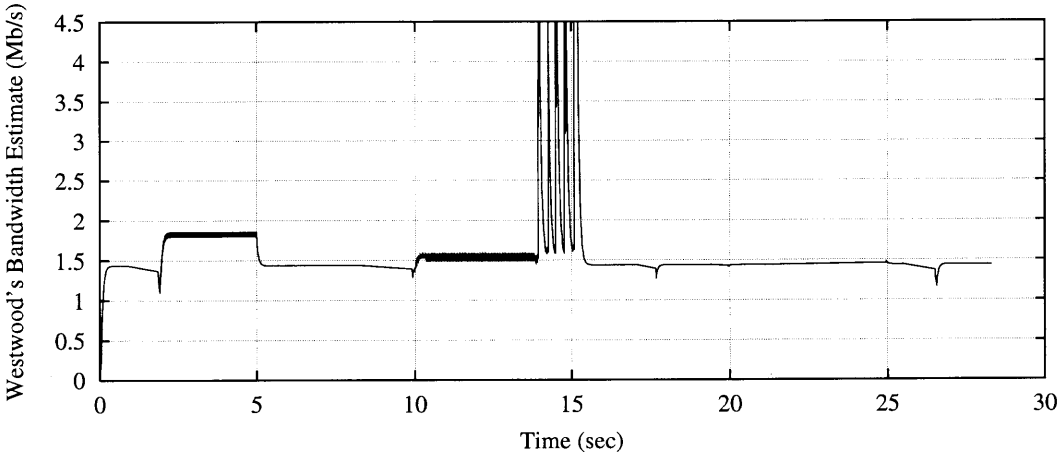


(a) Original ARE.

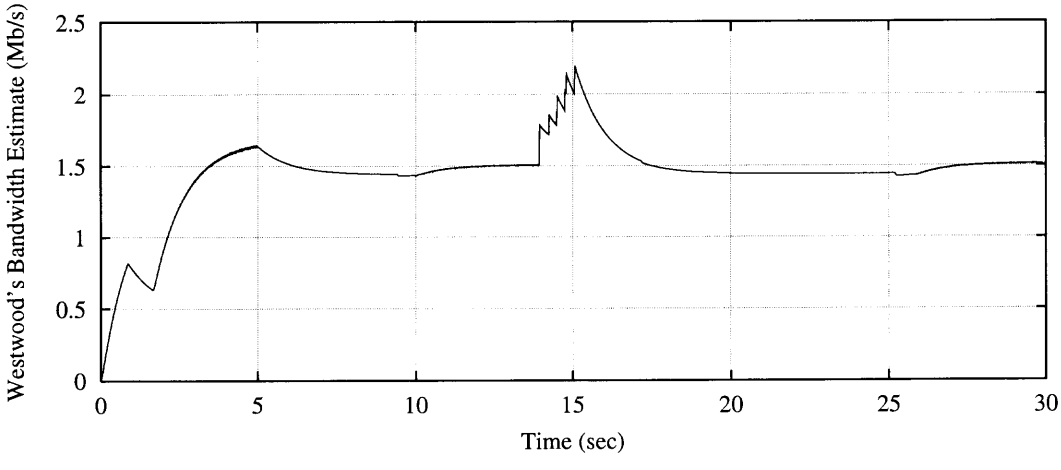


(b) Enhanced ARE.

Figure 7.3 Performance of original ARE vs. enhanced TS-ARE, using naive low-pass filter (filter type 2).



(a) Bandwidth estimation (BWE) output of original TCP Westwood.



(b) BWE output of TCP Westwood-NR.

Figure 7.4 Performance of the bandwidth estimation (BWE) implemented in the original TCP Westwood vs. the BWE implemented in TCP Westwood-NR. Both use the inter-arrival gaps of the ACK packets.

7.4 TS-ARE based TCP-Jersey in a Wired/Wireless Hybrid Network with Correlated Packet Losses

In this section, the performance evaluations of the enhanced TS-ARE based TCP-Jersey are conducted in comparison to other TCP schemes, including the original TCP-Jersey. In this section, instead of using the simple random packet drops on the wireless link, as was the case in previous simulations, the correlated packet losses are considered.

7.4.1 Correlated Errors

Unlike the bit errors in the wired networks, in which they are considered as random and independent, the error process in radio transmission is constituted by different phenomena, including path loss, fast fading, slow fading, and noise and interferences from other devices like microwave ovens [75]. The relationship between the distance and the bit error rate (BER) is not linear. Instead, the mean BER often remains constant up to a certain distance threshold and then degrades rapidly. This is due to the behavior on the receiver side, where all signals below a threshold are discarded. A simple and widely used model that characterizes the bit errors of the wireless channel between two stations is the Gilbert-Elliot model, also known as the two-state Markov model [75, 76, 77], where the process can be in either one of the two states, namely the good state and the bad state, and each state is associated with its BER, say, e_G and e_B, respectively.

Within each state, bit errors occur independently of each other. The transition between the two states is modeled as a continuous-time Markov chain as depicted in Fig 7.5, where P_{GB} and P_{BG} are the transition rates from the good state to the bad state and vice versa, respectively.

The mean sojourn time of staying in one state can be approximated by $T_G = \frac{1}{P_{GB}}$ and $T_B = \frac{1}{P_{BG}}$, where T_G and T_B are exponentially distributed. Let $e = [e_G, e_B]$,

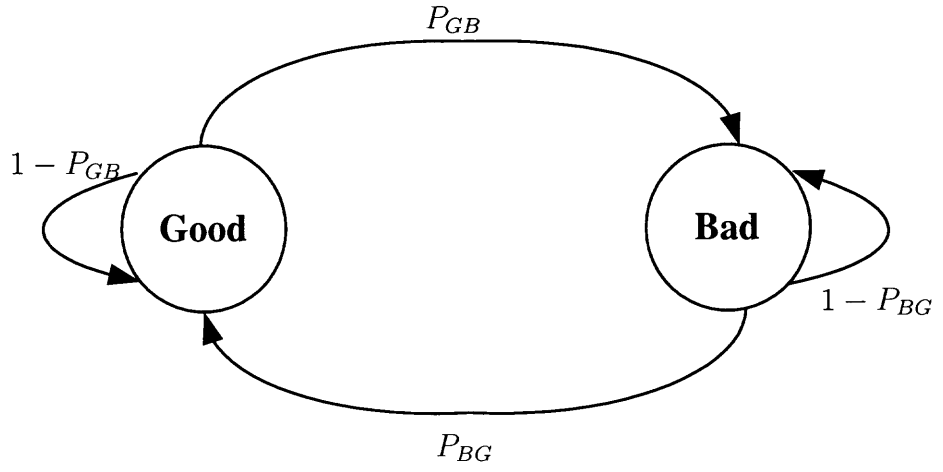


Figure 7.5 Two-state Markov Error Model

and $\pi = [\pi_G, \pi_B]$ be the steady state probabilities of the channel being in the good and bad states. The mean BER of the wireless channel under this model is then

$$\bar{e} = \pi e'. \tag{7.2}$$

Often, a long-term fraction of the time the channel spends in the good (or bad) state, i.e., f_G (f_B), is used to approximately represent the state probabilities. Therefore,

$$\pi_G \approx f_G = \frac{E[T_G]}{E[T_G] + E[T_B]}, \tag{7.3}$$

and

$$\pi_B \approx f_B = \frac{E[T_B]}{E[T_G] + E[T_B]}, \tag{7.4}$$

respectively.

Thus, the mean BER of the wireless channel under such an error model is

$$\begin{aligned} \bar{e} &= \pi e' \\ &\approx \frac{E[T_G]}{E[T_G] + E[T_B]} e_G + \frac{E[T_B]}{E[T_G] + E[T_B]} e_B. \end{aligned} \tag{7.5}$$

In many simulation implementations, the above is further simplified by assuming

$$\begin{aligned} e_G &= 0, \\ e_B &= 1. \end{aligned} \tag{7.6}$$

That is, when the channel is in the good state, no bit error is assumed to occur; whereas when the channel is in the bad state, all bits are assumed to be corrupted.

In reality, the actual packet loss probability due to the channel being in the bad state is different from the BER in (7.5). First, assuming the bit error occur independently from each other during the transmission of a packet of length N bits, the mean probability of that packet being corrupted[5] is $N\bar{e}$. Secondly, wireless link layer protocols such as the medium access control (MAC) and radio link control (RLC) often implement forward error correction (FEC) as well as its own packet retransmission mechanism, usually referred to as the automatic retransmission request (ARQ). These bit error correction and packet retransmission mechanisms can often reduce the mean probability of packet corruption from $N\bar{e}$.

In the simulations presented in the subsequent section, (7.6) is assumed and the error probability is computed as the final packet error rate (PER) directly, instead of BER, without considering the underlying FEC and ARQ mechanisms.

7.4.2 Goodput Evaluation via Simulations

A network setup, shown in Figure 7.6, with more complicated traffic patterns and correlated wireless errors is used to further evaluate the performance of the enhanced TS-ARE based TCP-Jersey.

The simulation network consists of 5 fixed nodes, S, C1, C2, C3, and C4; and one mobile node, D. There are two routers in the network, namely, R1 and BS. R1 is a general wired router, while BS is to simulate the wireless access point (AP)

[5]Here, a packet is considered corrupted if one or more of its bits are in error.

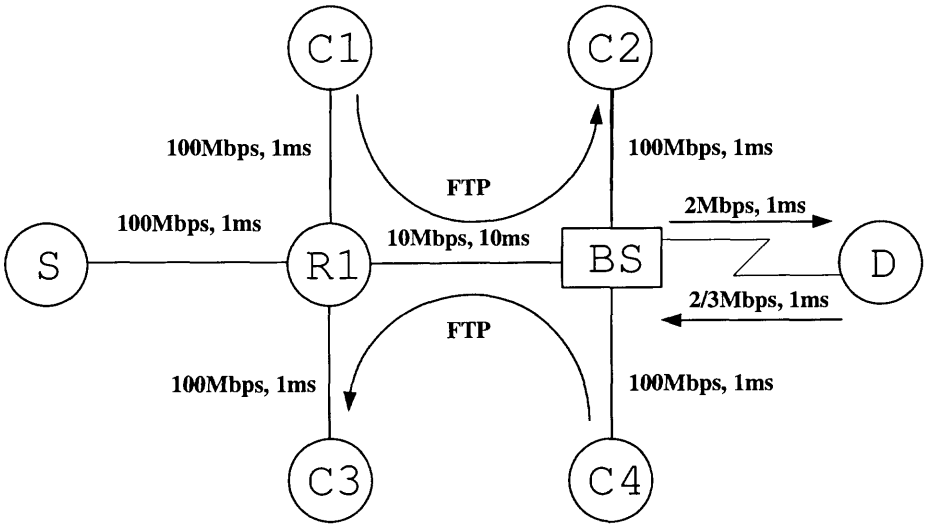


Figure 7.6 Simulation network setup of a wired/wireless hybrid network, resembing a typical SOHO networking configuration.

device found in many SOHO offices and campus buildings. The simulation network represents a typical wired and wireles hybrid network configuration in places such as wireless hot-spots, home networks with broadband access, and/or mobile networks in campus settings, where the BS, or AP, provides wireless 802.11 [78] connectivity to mobile computers such as laptops equipped with wireless network interface cards, and connects via wire (mostly Ethernet cables) to the broadband access network.

To facilitate the evaluation, three traffic flows are established in the network shown in Figure 7.6. The main flow is a long-lived FTP application carried by TCP-Jersey (and other TCP schemes under the evaluation) sending data from node S to node D. Two short-lived FTP applications from C1 to C2 and C4 to C3, respectively, constitute the cross traffic in both forward and backward directions. The two short-lived FTP flows are carried by the standard TCP Reno and are configured with independent exponentially distributed inter-session off-period with mean duration of 1s, and the sessions' on-period durations are drawn from independent exponential distributions with mean of 10s. These Poisson arrivals of

cross traffic flows simulate the realistic model of Internet mice traffic contending for the bandwidth. The last link in the network, namely from BS to D, is an asymmetric link, with the downlink capacity from BS to mobile of 2Mbps and the uplink capacity from the mobile to BS of 0.66Mbps.

Nine different TCP schemes are evaluated in this simulation. They are

- the enhanced TCP-Jersey using TS-ARE (with filter type 1),
- the original TCP-Jersey using ARE (with filter type 1),
- the original TCP Westwood based on TCP Reno,
- the TCP Westwood-NR, a later version based on TCP Newreno,
- the TCP SACK (selective acknowledgment),
- the TCP Vegas,
- the TCP Veno (a hybrid of Vegas and Reno),
- the standard TCP Reno,
- and the standard TCP Newreno.

Each TCP scheme is used in a separate simulation run as the main flow from node S to node D. Two sets of simulations are conducted, one in which an average correlated packet error rate (PER) of 5% on the wireless link is configured, and the other a PER of 10% is configured. Perfromance measures in terms of end-to-end goodput, i.e., the number of actual data packets correctly and in-order delivered, are evaluated and the results are plotted in Figure 7.7. In the figure, the y-axis represents the largest sequence number of the ACK packet received by the sender. Since all TCPs use cumulative acknowledgment, a curve in the figure shows the progression of the TCP flow in time, and the flow's goodput.

Under the network conditions set in this simulation, it is seen that with an average PER of 5%, drawn from the two-state Markov model, on the wireless link, both the original TCP-Jersey and the enhanced version perform about the same, and are noticeably better than the next-best performer TCP SACK, by a margin of about 12% (Figure 7.7(a)). When the wireless link is hit with an average of 10% correlated PER, the enhanced TCP-Jersey significantly out performs all other schemes including the original TCP-Jersey. Numerical results are listed in Table. 7.1

7.5 Summary

In this chapter, an enhanced version of the original TCP-Jersey is proposed and evaluated through simulations. The new TCP-Jersey uses a timestamp-based ARE estimator which computes the achievable rate by the inter-arrival gaps of the data packets at the receiver. It is thus called TS-ARE. This enhancement is implemented using the TCP's timestamps options defined in RFC 1323.

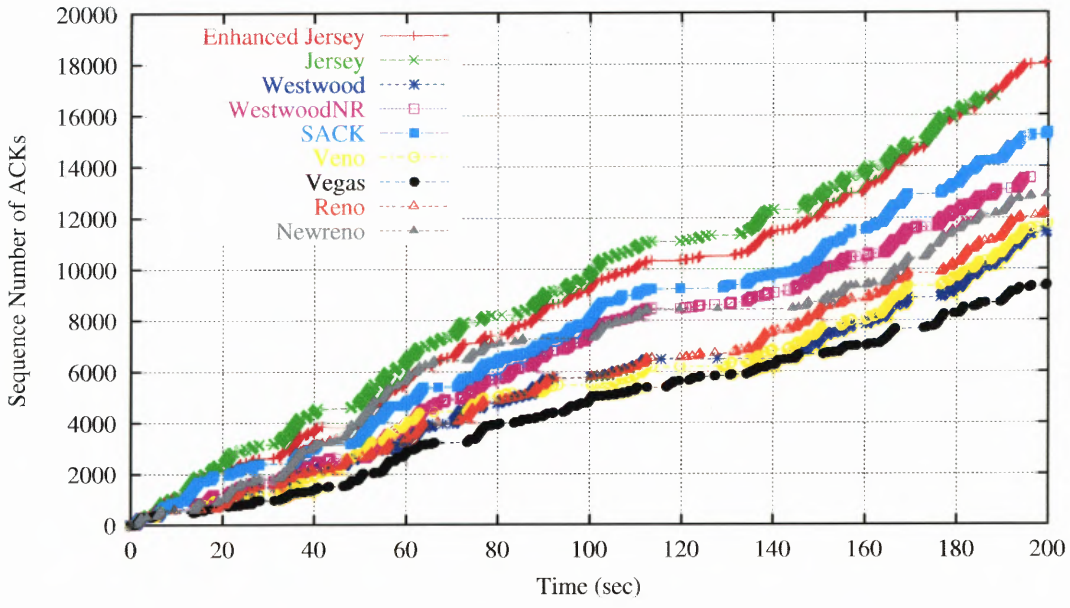
Section 7.3 evaluates the performance of the new TS-ARE and compares it with the original ARE. The evaluations also cover the comparison between the adaptive time-sliding window (ATSW) based rate estimation filter (referred to as filter type 1) and a simple extension of the non-adaptive low-pass filter (called filter type 2) based on the one used in the TCP Westwood's BWE algorithm. As was analyzed in Section 5.2, filter type 1 exhibits superior accuracy than filter type 2 in the simulations.

One of the characteristics of wireless links is that bit errors tend to occur in correlation, instead of independent of each other. A widely used two-state Markov error model is studied and used in the simulations reported in this chapter. The simulation network in this chapter is in close resemblance to the real networks found in many SOHO environments where a wireless AP connects the mobile nodes to the wired Internet. The results from the simulation show that the enhanced TCP-

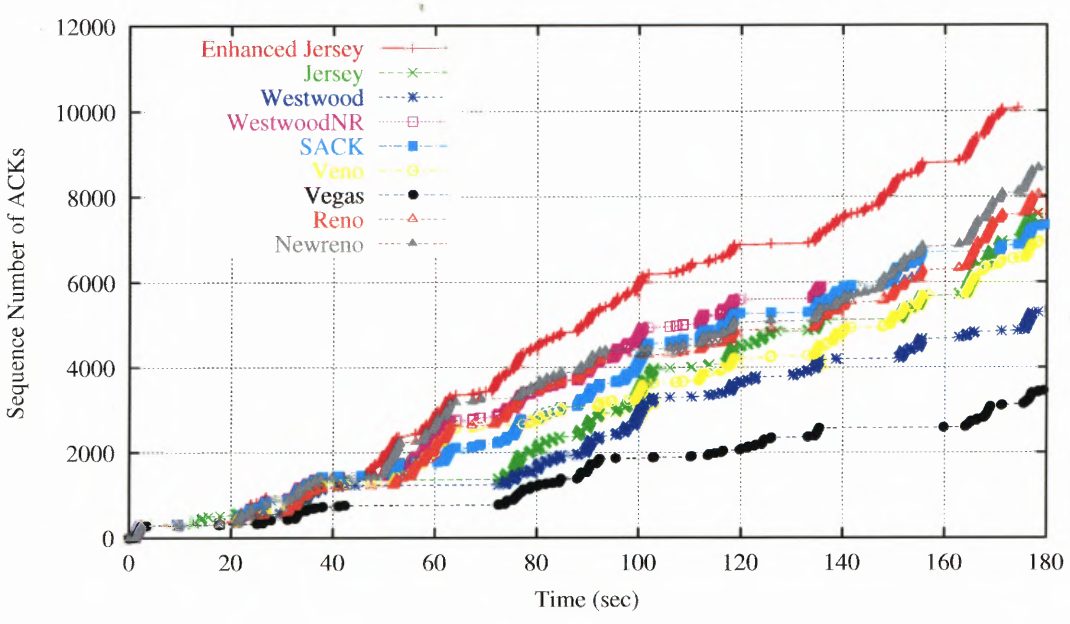
Jersey performs superbly as compared to other TCP schemes when the correlated packet error rate reaches as high as 10%.

Table 7.1 Numerical Results of TCP Goodputs in Network with Correlated Random Packet Losses

	Goodput (Kbps)	
TCP Schemes	5% PER	10% PER
Enhanced TCP-Jersey	722.72	403.44
Original TCP-Jersey	699.28	304.44
TCP Westwood	458.4	212.2
TCP Westwood-NR	543.24	236.24
TCP SACK	613.16	339.64
TCP Vegas	375.8	160.68
TCP Veno	471.48	315.48
TCP Reno	489.92	355.16
TCP New Reno	516.6	384.64



(a) Goodputs at approximately 5% correlated random packet drops.



(b) Goodputs at approximately 10% correlated random packet drops.

Figure 7.7 Performance of the enhanced TCP-Jersey with TS-ARE vs. the original TCP-Jersey and other TCP schemes. The correlated random packet loss rate on the wireless link is a two-state Markov error process with an average packet error rate of 5% and 10%, respectively in (a) and (b).

CHAPTER 8

ACTIVE QUEUE MANAGEMENT

In this chapter, we investigate the role of active queue management (AQM) in the context of TCP performance enhancement for wireless networks. We first review the state-of-the-art of AQM in Section 8.1. In Section 8.2, we present a dynamical model that captures the inter-connection between the TCP source's congestion control algorithm and the AQM's link algorithm. Section 8.3 presents detailed TCP window control dynamics for the standard TCP Reno and the proposed TCP-Jersey. This is followed immediately, in Section 8.4, by analytical studies of the potential performance improvement that the standard TCP Reno and the proposed TCP-Jersey would obtain in case AQM is used to aid the source in packet loss differentiation, i.e., the design framework proposed in this dissertation.

In Section 8.5 and Section 8.5.1, we discuss the issues related to the average queue length computation in the CW algorithm proposed in the original TCP-Jersey (referred to in this chapter as CWAQ), and propose a solution to eliminating the heuristic parameter setting in the EWMA moving average computation by introducing CWPQ, a CW marking scheme using the persistent queue length. Simulation results are also presented and analyzed.

The second issue regarding the heuristic parameter settings in the original CWAQ as well as in the new CWPQ, that is the rather subjective marking threshold setting, is discussed in Section 8.5.2 and Section 8.6. Another new AQM packet marking scheme, namely VCRR, or virtual capacity rate regulation, is then proposed and analyzed. Substantial simulations results are presented and discussed in the subsequent subsections.

8.1 A Brief Introduction to AQM

In this section, we briefly review existing AQM algorithms. A large body of AQM algorithms and architectures has been proposed; we will limit the review to those representatives in connection to TCP congestion control[1].

Networks consist of inter-connected links with which a buffer, or queue, of limited space is associated. A typical architecture of a packet forwarding link is usually illustrated as shown in Figure 8.1, where packets arriving to the link are either served (i.e., forwarded) immediately at the capacity c_l of the link, or are queued in the buffer waiting for the packets in front to be served first. Therefore, from the time a packet arrives to the link to the time it departs from the link, the packet is delayed by the possible waiting time in the queue plus the processing time incurred by the output transmitter.

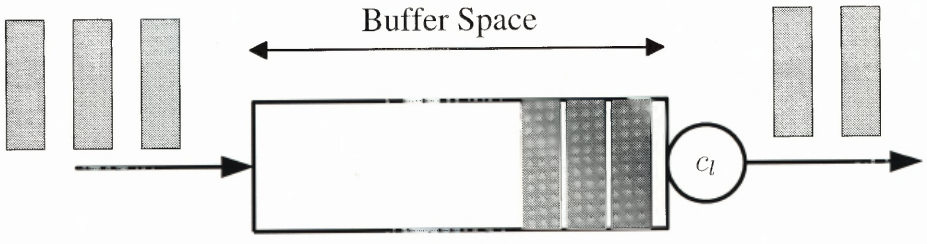


Figure 8.1 A typical link with the associated queue.

The simplest queues in the network, which is still the dominant type of queues deployed in today's Internet, is the unmanaged, or not actively managed, queues that discard all arriving packets once the queue has reached the buffer limit. These are often called drop-tail queues. The problem associated with the simple drop-tail queues in Internet is, firstly, a drop-tail queue entails a large amount of packet drops all at once when its buffer becomes full, resulting in the *global synchronization* problem reported by Floyd and Hashem in [54] and [81], respectively.

[1]To a large extent, Internet congestion control, of which works reported in [79, 80] provide excellent and extended reviews and analysis.

Secondly, drop-tail queues may cause some TCP connections to be "locked out", a phenomenon reported in [82] where some TCP flows receive significantly less throughputs than the established connection.

Random Early Detection, or RED, is the first AQM algorithm proposed in the Internet research community, and is by far the most studied and known AQM scheme. It was developed by Floyd and Jacobson [54] in 1993. Its design was a response to the problems exhibited in the drop-tail queues, particularly related to the TCP performance. The main design philosophy in RED is to drop (by probability) packets *before* the buffer of the bottleneck link overflows. The early dropping scheme works together with the standard TCP congestion control algorithm such as Reno, which halves its transmission rate upon the detection of a packet drop, to prevent the buffer from being overflow. RED drops a packet upon it arrival by a probability that is determined by the average queue length as

$$p_l(t) = \begin{cases} 0 & \tilde{b}_l(t) < b_{min}, \\ \frac{p_{max}}{b_{max}-b_{min}}(\tilde{b}_l(t) - b_{min}) & b_{min} \leq \tilde{b}_l(t) < b_{max}, \\ 1 & \tilde{b}_l(t) \geq b_{max}, \end{cases} \tag{8.1}$$

where p_{max} is the maximum dropping probability, b_{min} is the lower threshold of the average queue length, b_{max} is the higher threshold of the average queue length, and $\tilde{b}_l(t)$ is the average queue length computed by the link upon the arrival of a packet. The computation of average queue length is by exponentially weighted moving average of the instantaneous queue length $b_l(t)$, of which the update rule in discrete steps is

$$\tilde{b}_l[n+1] = w_q b_l[n] + (1 - w_q)\tilde{b}_l[n], \tag{8.2}$$

where $0 < w_q < 1$ is the selected weight. Therefore, RED is designed to maintain a low average queue length at the link by signaling the TCP source to reduce its transmission rate before the link becomes congested.[2]

Explicit Congestion Notification [55,87], or ECN, is a derivative of the original RED. Instead of early dropping packet as the congestion signal, the link marks the packets by setting a designated bit in the packet header. The marked bits are echoed by the receiver in the corresponding ACK packets to the source. These marking bits form a means of in-band signaling mechanism between the link and the source. TCP sources with ECN capability reduce their sending rate according to the congestion control algorithm upon receiving the marked bits in the packet header as if a packet loss was detected. More detailed operation of ECN is summarized in Appendix C. The packet dropping/marking function of RED/ECN is illustrated in Figure 8.2. We want to emphasize that ECN provides a new paradigm of in-band signaling between the link algorithm and the source algorithm which can be applied to Internet congestion control as well as other networking objectives such as fair resource allocation. Its implementation need not be restricted to the mechanisms provided in RED.

The AQM link algorithm such as RED/ECN and the TCP source congestion control algorithm thus form a closed-loop feedback control system, where AQM is the controller and TCP is the plant to be controlled, and the system output is either the source's transmission rate, or the queuing statistics (e.g., queue length, queuing delay, etc.). Therefore, from a control system point of view, the AQM and TCP relationship can be depicted as in Figure 8.3, where the AQM algorithms as

[2]RED does not have signaling effect to flows carried by non-responsive transport protocols such as UDP. In this case, the higher layer protocols would observe large amount of packet losses (on average) and is expected to, by its application design, to throttle the transmission rate. Apparently, this does not stop malicious applications or misbehaving flows from denying other flows for the link resource, since RED does not explicitly distinguish and punish misbehaving flows. This is yet another big research topic of fair queuing which is beyond the scope of this dissertation, see e.g., [83,84,85,86] and the references therein.

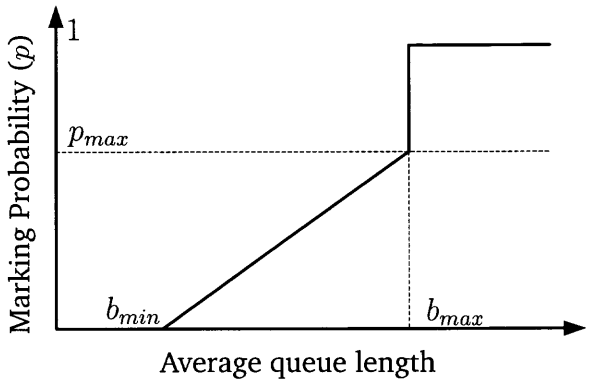


Figure 8.2 Packet marking function of RED/ECN.

the controller of the system is categorized into queue length based and rate based. As shown in Figure 8.3(a), the queue length based AQM takes in some statistics directly measured from the queue length as the error signal and provides control signal to the plant (i.e., TCP source) in the form of packet dropping or marking in order to control the queue length to the desired level. On the other hand, the rate based AQM controller in Figure 8.3(b) acts on the error signal derived from the output transmission rate of the TCP source (or, equivalently the input rate to the link[3]) and attempts to drive the transmission rate to a desired level.

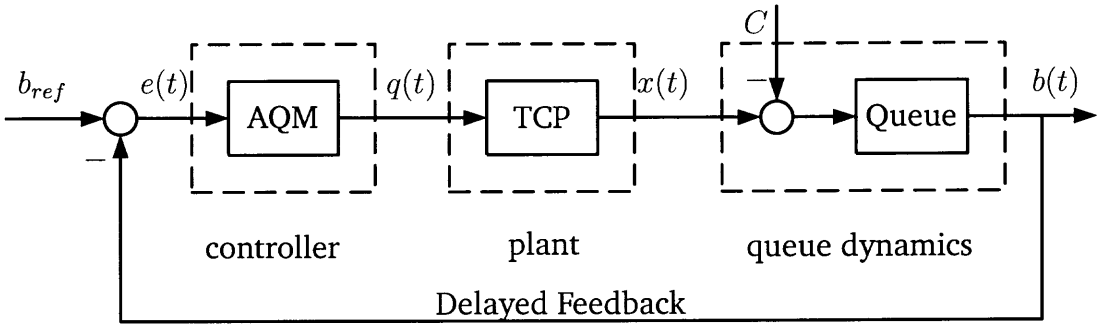
The queue dynamics and the input rate of the link are inter-connected through the following equations

$$\dot{b}(t) = \begin{cases} x(t) - C & 0 < b(t) < B, \\ max(0, b(t) - C) & q(t) = 0, \\ min(0, b(t) - C) & q(t) = B, \end{cases}$$

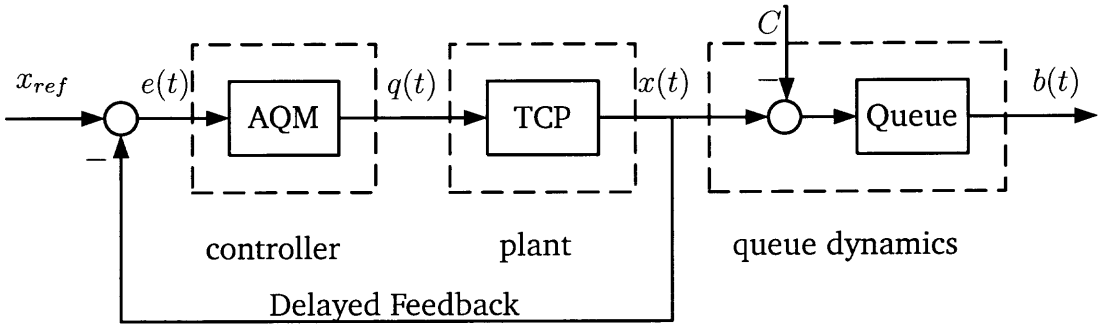
$$\tau(t) = d + \frac{b(t)}{C},$$

where B is the buffer limit of the queue, $\tau(t)$ is the delay, and d is the propagation delay. Therefore, the two types of AQM are also closely related. Queue length based

[3]In reality, the *aggregate* input rate to the link.



(a) Queue length based control.



(b) Rate based control.

Figure 8.3 Block diagram of closed-loop control system of TCP/AQM.

AQMs are easier to implement because the length of the queue can be directly observed by the link, but rate based AQMs have some advantages. The queuing process as in the preceding equations cannot observe the long-term input rate, since if $x(t) < C$ for a long period of time, $b(t)$ remains to be zero and no information of $x(t)$ or its closeness to C can be drawn.

RED has inspired a large body of AQM researches which in turn have produced a wide variety of AQM algorithm proposals. However, RED itself is not widely deployed largely due to the problems of parameter tuning and instability [65, 88, 89, 90, 91].

From the control theoretic point of view, the RED link algorithm is essentially a proportional controller, or P-controller, since the control signal is proportional to the error signal.

An improvement to RED's P-controller, the PI AQM [92] was introduced by Hollot and et al., where the packet dropping/marking rate is updated by

$$p_l[n] = \Big[p_l[n-1] + \alpha(b_l[n] - b_{ref}) - \beta(b_l[n-1] - b_{ref})\Big]^+, \tag{8.3}$$

where α and β are control parameters. Here, the operator $[x]^+$ is defined as $[x]^+ = max(0, x)$. The above can be rewritten in the continuous time domain as

$$\begin{aligned} p_l(t) &= \beta(b_l(t) - b_{ref}) + (\alpha - \beta)\int_0^t (b_l(\tau) - b_{ref})d\tau \\ &= K_P^{PI} e(t) + K_I^{PI}\int_0^t e(\tau)d\tau. \end{aligned} \tag{8.4}$$

As the name suggests, this is a proportional and integral controller, or a PI-controller. It integrates the queue length error in order to make the dropping probability according to the input rate to the link.

Random Exponential Marking [93], or REM, is another AQM scheme where the marking probability is computed by

$$\begin{aligned} p_l[n] &= 1 - \Phi^{-\mu[n]} \\ \mu[n] &= \Big[\mu[n-1] + \gamma\big(\alpha(b_l[n-1] - b_{ref}) + (x[n] - c_l)\big)\Big]^+, \end{aligned} \tag{8.5}$$

where $\Phi > 1$ is a parameter known to all sources and links. It has been shown in [94] that REM is essentially also a queue length based PI-controller in the form of

$$p_l(t) = K_P^{REM} e(t) + K_I^{REM} \int_0^t e(\tau) d\tau, \tag{8.6}$$

where $K_P^{REM} = (1 - \alpha)\gamma ln\Phi$ and $K_I^{REM} = \alpha\gamma ln\Phi$. The $\mu(t)$, or its discrete form $\mu[n]$, is termed as the *price* and the marking probability function can be represented as in Figure 8.4.

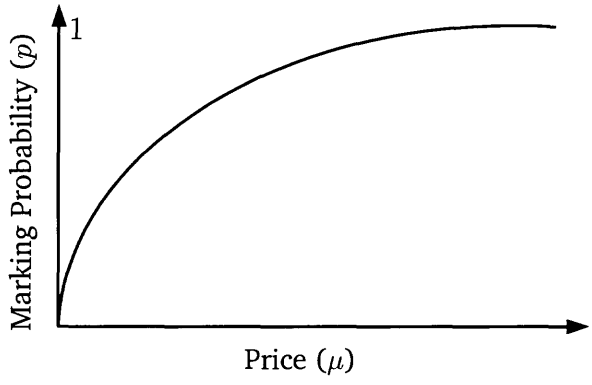


Figure 8.4 Packet marking function of REM.

Adaptive Virtual Queue [95], or AVQ, is a unique rate based AQM algorithm. It is derived from the perspective of network optimization framework [96]. The AVQ algorithm uses a *virtual queue* whose capacity $\tilde{c}_l$ is less than the actual link capacity c_l, i.e., $\tilde{c}_l = \gamma c_l$, and $0 < \gamma < 1$. The virtual queue has the same buffer limit

as the real queue. The capacity of the virtual queue is updated by

$$\dot{\tilde{c}}_l(t) = -\alpha(x(t) - \beta c_l), \tag{8.7}$$

where α and β are the control parameters. Packets in the real queue are dropped or marked when the virtual queue starts overflow. The packet dropping or marking probability is computed by

$$p_l(t) = \left[1 - \frac{\tilde{c}_l(t)}{x(t)}\right]^+. \tag{8.8}$$

AVQ adapts the input rate to achieve the desired level of utilization and keep the real queue length short.

Zhu and Ansari proposed a virtual queue and rate based AQM called VQR in [97] where the virtual queue has a fixed virtual capacity $\tilde{c}_l = \gamma c_l$ and the marking probability is determined by the virtual queue length $\tilde{b}_l(t)$ and input rate $x(t)$ as

$$p_l(t) = \frac{g}{\tilde{c}_l}\left[a(\tilde{b}_l(t) - b_{ref}) + (x(t) - \tilde{c}_l)\right]^+. \tag{8.9}$$

Consider the virtual queue length dynamics $\dot{\tilde{b}}_l(t) = x(t) - \tilde{c}_l$, the above can be expressed as a PI-controller with the error signal being the virtual queue length.

Finally, a proportional integral and derivative, or PID, AQM controller has been proposed in [94] called VRC. The packet marking probability at the link is determined by

$$p_l(t) = \left[\alpha(x(t) - c_l) + \alpha(\beta + \gamma)(b_l(t) - b_{ref}) + \alpha\beta\gamma \int_0^t (b_l(\tau) - b_{ref})d\tau\right]^+, \tag{8.10}$$

which can be expressed as

$$p(t) = K_P^{VRC} e(t) + K_I^{VRC} \int_0^t e(\tau)d\tau + K_D^{VRC} \dot{e}(t), \tag{8.11}$$

where the queue length mismatch $e(t) = b_l(t) - b_{ref}$ is the error signal to the PID controller.

Active queue management is a very active research area with a large body of literature and proposed algorithms. The major concern for AQM in IP networks is for congestion control and fair resource allocation. There are two different approaches to the design of AQM algorithms, one is from the framework of optimization of network utility, e.g., [98, 99, 100, 101], and the other is from the control theoretical perspective. An in-depth discussion of the models and methodologies can be found in [102]. The review in this section provides a brief introduction to the inter-connections between an AQM link algorithm and the TCP's source congestion control algorithm to the degree pertaining to the rest of this dissertation.

8.2 Basic Network Model for TCP/AQM Analysis

In this section, a basic model for analyzing the TCP/AQM dynamics is presented. A network is modeled by a set of bi-directional communication links indexed by $l \in \boldsymbol{L}$, each with a link capacity of $c_l \in \boldsymbol{C}$. Flows are identified by their source-destination pairs indexed by $i \in \boldsymbol{I}$. The end-to-end path of flow i is denoted by a set of links $\boldsymbol{L_i} \subseteq \boldsymbol{L}$. The connectivity matrix or routing matrix $\boldsymbol{R}$ is defined by its elements as

$$R_{li} = \begin{cases} 1 & \text{if source } i \text{ uses link } l \\ 0 & \text{otherwise} \end{cases} . \tag{8.12}$$

In reality, routing matrix $\boldsymbol{R}$ changes as the network load varies, or due to route changes of the network. However, we assume this happens in a much slower time-scale than our analysis, and therefore, $\boldsymbol{R}$ is assumed static throughput the analysis. Aggregate flow rate $y_l(t)$ at each link is defined by the equation

$$y_l(t) = \sum_{i \in \boldsymbol{I}} R_{li} x_i(t - \tau_{li}^f(t)), \tag{8.13}$$

where $x_i(t)$ is the transmission rate associated with flow i, and $\tau_{li}^f(t)$ is the forward delay between the source i [4] and link l.

The feedback mechanism is provided through the congestion indication function $p_l(t)$ at each link (refer to the schematic in Figure 8.3). The function $p_l(t)$ is often referred to as the packet marking probability if the routers implement ECN-like signaling mechanism by setting a designated bit in the packet header in the event of congestion; or, as dropping probability in cases where routers implement no AQM (e.g., tail-drop routers) or RED-like mechanism. As with many literature, we use marking and dropping interchangeably. Sometimes, $p_l(t)$ is also referred to as *price*, e.g., in [99, 93]. In our analysis, we prefer to call $p_l(t)$ as congestion indication or the proportion of packets being marked/dropped at link l.

Following [99, 101, 103, 104], assuming packet marking or dropping occurs independently on different links, the source observes the aggregate marking probability of all links in its path as

$$q_i(t) = 1 - \prod_{l \in \boldsymbol{L_i}} (1 - p_l(t - \tau_{li}^b(t))),$$

where $\tau_{li}^b(t)$ is the backward delay from link l to source i. For the sake of tractable analysis, the above is usually approximated by

$$q_i(t) = \sum_{l \in \boldsymbol{L_i}} R_{li} p_l(t - \tau_{li}^b(t)), \tag{8.14}$$

under the assumption that $p_l(t)$ is small.

The round-trip time (RTT) for flow i at time t is defined by

$$\tau_i(t) = d_i + \sum_{l \in \boldsymbol{L_i}} R_{li} \frac{b_l(t)}{c_l}, \tag{8.15}$$

in which d_i is the round-trip propagation delay and $b_l(t)$ is the backlog, or queue occupancy, at link l at time t. Strictly speaking, this is not the round-trip time

[4] Here, we use i to index both a flow and its source.

experienced by a packet, which visits different links in its path at different times, and hence experiences queuing delays of various links at different times. The expression used here, however, sums the queuing delay of different links at the *same time* t. The forward and backward delays are related to RTT through

$$\tau_i(t) = \tau_{li}^f(t) + \tau_{li}^b(t), \text{ for all } l \in \boldsymbol{L_i}. \tag{8.16}$$

8.3 TCP Dynamic Model

Based on the network model established in the preceding section, the dynamic models of the standard AIMD congestion control found in TCP Reno as well as the proposed TS-ARE based AIAD congestion control are presented in this section.

First, since TCP is a window-based protocol. In particular, the congestion window ($cwnd$) maintained by the TCP source controls (roughly) the number of data packets the sender is allowed to send within one RTT; $cwnd$ advances as positive acknowledgments are received by the source. The congestion window $w_i(t)$ and the source's transmission rate $x_i(t)$ is related as

$$x_i(t) = \frac{w_i(t)}{\tau_i(t)}. \tag{8.17}$$

Second, all rates are assumed in units of packets per second, and all packets are assumed to be of the same size. This simplification has no impact on the analytical results and does not lose its generality either. Since, we are primarily interested in, as far as the modeling process is concerned, the persistent sources which can be controlled. For those persistent flows, i.e., the "elephants", the majority of its packets are sent in the size of the underlying MTU (maximum transmit unit) of the layer two technologies, such as Ethernet. In fact, TCP peers, during their initial three-way handshake, negotiate the "maximum segment size", which is in most cases equal to the smaller of the two end stations' MTUs, or the standard 536 bytes [20]. We remark that besides the "elephants", the network is

also shared by short-lived flows, called "mice", which do not last long enough to be controlled, but are affected by the elephant dynamics, mainly through the queuing delay they experience. This section does not attempt to model the "mice" flows explicitly[5].

Lastly, TCP's transient behavior in the Slow-Start, Fast-Retransmit/Recovery, and Retransmit Timeout [20] are ignored in the models, and only the congestion avoidance phase is concerned.

8.3.1 Model of TCP Reno Source

For TCP Reno, at time t, source i transmits at rate $x_i(t)$. Assuming the receiver acknowledges every packet it received, the source receives the acknowledgments at rate $x_i(t - \tau_i(t))$, of which a proportion of $1 - q_i(t)$ are positive and each results in an increment of $w_i(t)$ by $1/w_i(t)$. Therefore, the congestion window increases on average at the rate of $x_i(t - \tau_i(t))(1 - q_i(t))/w_i(t)$. Similarly, negative acknowledgments are received at an average rate of $x_i(t - \tau_i(t))q_i(t)$, each results in a decrement of the congestion window by half, and hence the window $w_i(t)$ decreases at a rate of $x_i(t - \tau_i(t))q_i(t)w_i(t)/2$. Therefore, the window dynamics of TCP Reno can be expressed in a differential equation as

$$\dot{w}_i(t) = \left[x_i(t - \tau_i(t))(1 - q_i(t))\frac{1}{w_i(t)}\right] - \left[x_i(t - \tau_i(t))q_i(t)\frac{w_i(t)}{2}\right], \tag{8.18}$$

in which $x_i(t)$ and $q_i(t)$ are given by (8.17) and (8.14), respectively. The first term in the above represents the additive increase part, and the second term the multiplicative decrease part of the AIMD algorithm.

[5]In some literature, the Internet mice flows are treated as noise in the dynamic system.

8.3.2 Model of TCP-Jersey Source

TCP-Jersey's congestion control is AIAD in nature, where the adaptive decrease law is determined by the source's active measurement of the achievable rate that the network can sustain without causing congestion. This is carried out by the proposed TS-ARE algorithm discussed in the previous chapter.[6] Therefore, as compared to the TCP Reno's model, the TCP-Jersey's window increase part is the same as that of Reno, and the difference is in the window decrease part.

Upon the reception of a negative acknowledgment, the TCP-Jersey source reduces its congestion window to $r_i(t)d_i$, where $r_i(t)$ is the source's estimation of the achievable rate of the flow. According to the TS-ARE algorithm discussed in Chapter 7, $r_i(t)$ is essentially a low-pass filter output of the rate of the positive delivery of packets at the receiver, which is $x_i(t-\tau_i(t))(1-q_i(t))$. Therefore, the dynamics of the TCP-Jersey's congestion control algorithm can be expressed by following two differential equations,

$$\dot{w}_i(t) = \left[x_i(t-\tau_i(t))(1-q_i(t))\frac{1}{w_i(t)}\right] - \left[x_i(t-\tau_i(t))q_i(t)(w_i(t)-r_i(t)d_i)\right], \quad (8.19)$$

and

$$\zeta_i \dot{r}_i(t) + r_i(t) = x_i(t-\tau_i(t))(1-q_i(t)), \quad (8.20)$$

where ζ_i is the time constant of the low-pass filter of the TS-ARE estimator[7], and $x_i(t)$ and $q_i(t)$ are given by (8.17) and (8.14), respectively.

---

[6]From now on, unless otherwise stated, TCP-Jersey is referred to as the one with the enhanced TS-ARE estimator as opposed to the original one using the non-timestamps based ARE estimator.

[7]Recall the discussion in Chapter 5 that ζ_i in our TS-ARE is in fact time-varying and adaptive to network conditions. For simplicity, here we consider ζ_i as the steady-state value of the time-varying filter parameter.

8.4 Equilibrium Analysis and Relationship to Loss Differentiation Accuracy

Within the concern of this dissertation, TCP throughput degrades in the event of random packet losses, which usually occur in wireless networks. Loss differentiation plays a vital role in improving the performance measure of TCP in lossy environments. The congestion control models derived in the preceding sections for the standard TCP Reno and the proposed TCP-Jersey do not capture the loss differentiation. In other words, the loss probability at each link $p_l(t)$, and henceforth $q_i(t)$, are always considered as congestion indications, and the source reacts accordingly either by the AIMD algorithm as in TCP Reno, or by the AIAD algorithm as in TCP-Jersey. In reality, the fraction of packet loss $x_i(t-\tau_i(t))q_i(t)$ includes packet losses due to congestion as well as wireless transmission errors.

In this section, we attempt to establish a relationship between the improvement on the average throughput of TCP and the accuracy of loss differentiation. The average throughput of TCP is best studied by computing the equilibrium values of the models (8.18) and (8.19)-(8.20).

8.4.1 Equilibrium Properties of TCP Reno and TCP-Jersey Sources

Let $(w_i^*, x_i^*, \tau_i, p_l^*, q_i^*)$ be the equilibrium values of $(w_i(t), x_i(t), \tau_i(t), p_l(t), q_i(t))$. For the TCP Reno source's AIMD model (8.18), the steady-state window size can be derived by setting $\dot{w}_i(t)$ to zero, which yields the following equilibrium

$$\mathfrak{E}_{AIMD} = \begin{cases} w_i^* &= x_i^* \tau_i \\ x_i^* &= \sqrt{\frac{2(1-q_i^*)}{\tau_i^2 q_i^*}} \\ \tau_i &= d_i + \sum_{l \in \boldsymbol{L_i}} R_{li} \frac{b_l^*}{c_l} \end{cases}, \tag{8.21}$$

where b_l^* is the equilibrium queue occupancy at link l.

Similarly, for the TCP-Jersey source's AIAD model (8.19)-(8.20), the equilibrium are derived by setting appropriate derivatives to zero as

$$\mathfrak{E}_{AIAD} = \begin{cases} w_i^* &= x_i^* \tau_i \\ x_i^* &= \sqrt{\frac{1-q_i^*}{\tau_i d_i {q_i^*}^2 + (\tau_i^2 - \tau_i d_i) q_i^*}} \\ r_i^* &= x_i^*(1-q_i^*) \\ \tau_i &= d_i + \sum_{l \in \boldsymbol{L}_i} R_{li} \frac{b_l^*}{c_l} \end{cases} . \tag{8.22}$$

For the sake of notational clarity, consider a single flow case and drop the flow index i in (8.21) and (8.22). Denote, from (8.21) and (8.22), $x^*_{AIMD} = \sqrt{\frac{2(1-q^*)}{q^* \tau^2}}$ and $x^*_{AIAD} = \sqrt{\frac{1-q^*}{q^*[1-(1-q^*)d/\tau]\tau^2}}$ as the equilibrium throughput attainable by the AIMD and AIAD congestion control, respectively. We can define the ratio of the equilibrium throughput between AIMD and AIAD congestion control schemes under the same network conditions as

$$\Delta = \frac{x^*_{AIMD}}{x^*_{AIAD}} = \sqrt{2[1-(1-q^*)\frac{d}{\tau}]}. \tag{8.23}$$

For high-speed networks, it is reasonable to assume that the average queuing delay is very small[8], i.e., b_l^*/c_l is very small, and thus, $d/\tau \approx 1$. This yields

$$\Delta = \frac{x^*_{AIMD}}{x^*_{AIAD}} \approx \sqrt{2q^*}. \tag{8.24}$$

Therefore in high-speed networks, when the steady-state congestion packet loss rate is less than 50%, TCP using the proposed AIAD congestion control law would achieve higher steady-state throughput than TCP employing the standard AIMD congestion control algorithm, i.e., $\Delta < 1$ for $q^* < 0.5$.

[8]For instance, a link of 1Gbps speed can transmit 125 packets of size 1000 bytes per packet in just 1 millisecond.

8.4.2 Potential Performance Improvement by Loss Differentiation

TCP-Jersey is the proposed architecture that incorporates a particular type of packet loss differentiator, which is based on the CW markings as discussed in Chapter 5. In general, suppose a type of packet loss differentiator is employed in TCP Reno and TCP-Jersey with the accuracy of correctly detecting congestion induced loss being A_c and the accuracy of correctly detecting non-congestion induced (or wireless related) loss being A_w, and at each link, the packet loss probability can be expressed as

$$p_l(t) = p_{c,l}(t) + p_{w,l}(t), \tag{8.25}$$

where $p_{c,l}(t)$ and $p_{w,l}(t)$ are the packet loss probabilities of those caused by congestion and those by wireless transmission errors, respectively.[9]

Also, assume that with the deployment of a loss differentiator, the preferred behavior of TCP source's congestion control is to retransmit the lost packet without decreasing the window size (as is proposed in TCP-Jersey), then the effective loss probability that affects the TCP source's congestion control is

$$\hat{p}_l(t) = A_c p_{c,l}(t) + (1 - A_w) p_{w,l}(t). \tag{8.26}$$

In other words, the effective (congestion) loss probability is the result of the actual congestion losses correctly detected by the loss differentiator and the wireless losses misclassified by the differentiator as congestion losses.

For notational clarity, we consider a network with one TCP flow and one wireless link, then the flow index i and link index l in (8.18) and (8.19) can be dropped, and the routing matrix becomes a scalar of constant 1. The TCP source's congestion control law, either AIMD or AIAD, takes $p(t)$ in (8.25) as congestion indication if no loss differentiator is employed, or, otherwise, with loss

[9]With a reasonable assumption that link transmission error and link congestion are independent events.

differentiator, the $\hat{p}(t)$ in (8.26) is used in the window adjustments. The former case is equivalent to the latter with $A_c = 1$ and $A_w = 0$, meaning all packet losses are (mis)identified as congestion losses.

Denote $x_{AIMD}(A_c, A_w)$ as the equilibrium throughput of a TCP flow with AIMD congestion control (e.g., TCP Reno), and $x_{AIAD}(A_c, A_w)$ the equilibrium throughput of a TCP flow with AIAD congestion control (e.g., TCP-Jersey), respectively. We can define the potential TCP throughput improvement by employing a loss differentiator with loss classification accuracy of (A_c, A_w) over a TCP source without loss differentiator as

$$\Delta_{AIMD}(A_c, A_w) = \frac{x_{AIMD}(A_c, A_w)}{x_{AIMD}(1, 0)} - 1, \tag{8.27}$$

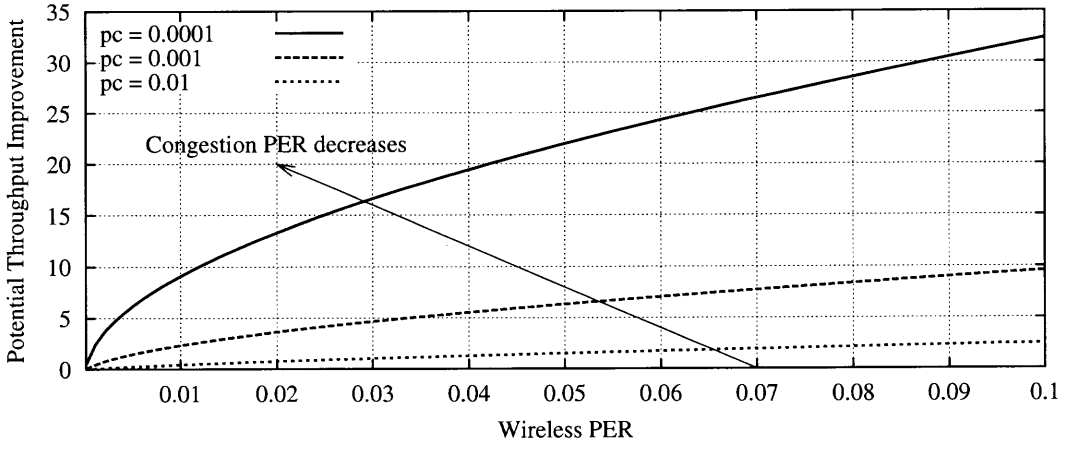
and

$$\Delta_{AIAD}(A_c, A_w) = \frac{x_{AIAD}(A_c, A_w)}{x_{AIAD}(1, 0)} - 1, \tag{8.28}$$

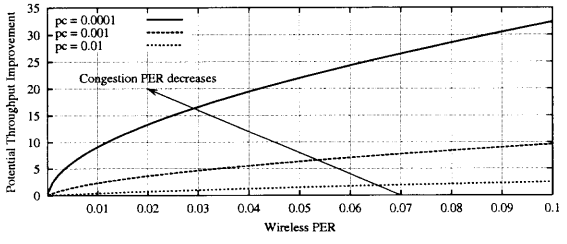
respectively.

By substituting (8.21) and (8.22) into (8.27) and (8.28), respectively, a plot of the potential throughput improvement vs. the actual wireless packet loss rate (wireless PER) can be obtained. Figure 8.5 shows the potential throughput improvement for a *perfect* loss differentiator (i.e., $A_c = 1$ and $A_w = 1$) when the underlying wireless error induced packet loss rate varies from 0.001 to 0.1.

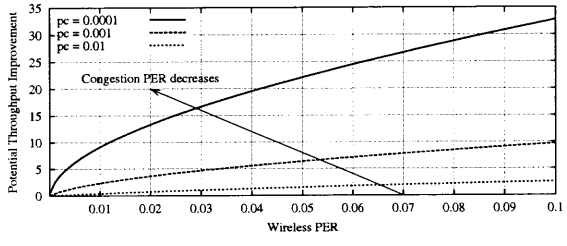
Contrary to what a straightforward reading of Figure 8.5 would indicate, so far there has been no real implementation nor analysis of any loss differentiation algorithm reported that could improve the TCP performance by tens of folds. The inaccuracy of the model's prediction is caused by the over simplification of the models (8.18) and (8.19), where TCP's retransmission timeout and other protocol details are not captured. When packet loss is detected by timeout, TCP source sets the congestion window to 1 regardless of the reason of the loss; then restarts



(a) $\Delta_{AIMD}(1,1)$.



(b) $\Delta_{AIAD}(1,1)$, with $\tau = 0.05$, $d = 0.002$.



(c) $\Delta_{AIAD}(1,1)$, with $\tau = 0.5$, $d = 0.1$.

Figure 8.5 Potential throughput improvement vs. wireless packet loss rate, in case of a perfect loss differentiator. ('pc' in the legend indicates $p_{c,l}$)

the window-based control from the slow-start phase. Furthermore, each timeout event causes the TCP source to exponentially back-off its transmission, meaning the source stops the transmission upon a timeout event for a period that is doubled for each timeout occurrence. This exponential transmission back-off behavior has been illustrated in Figure 3.3. A detailed model of the TCP Reno's source that accounts for the Slow-start as well as Timeout behaviors has been developed by Padhye et al. in [21] and the resulting closed form of TCP Reno's throughput in relation to the packet loss probability is given in (2.4).

When packet loss rate is high, whether due to congestion or wireless link error, losses are often detected by the source via timeouts. Therefore, given a more detailed model, one would expect the potential improvement in Figure 8.5 start to drop as wireless PER becomes higher than some thresholding point. This has been studied by Biaz and Vaidya in [105, 53] using the detailed model of (2.4).

We remark that in order to combat the excessive packet losses due to wireless hand-off process or temporary radio blockage, a persistent mode operation, such as those proposed in Freeze-TCP [38] by Goff et al., and ATCP [37] by Liu and Singh, could be an effective solution. We leave the search of a mechanism to detect wireless hand-off, and a solution to prolonged period of excessive packet loss rate for future endeavor.

A similarly detailed source model of TCP Westwood's AIAD congestion control is studied by Zanella et al. in [106] using Markov Chain analysis. However, the result is not in an explicit closed form.

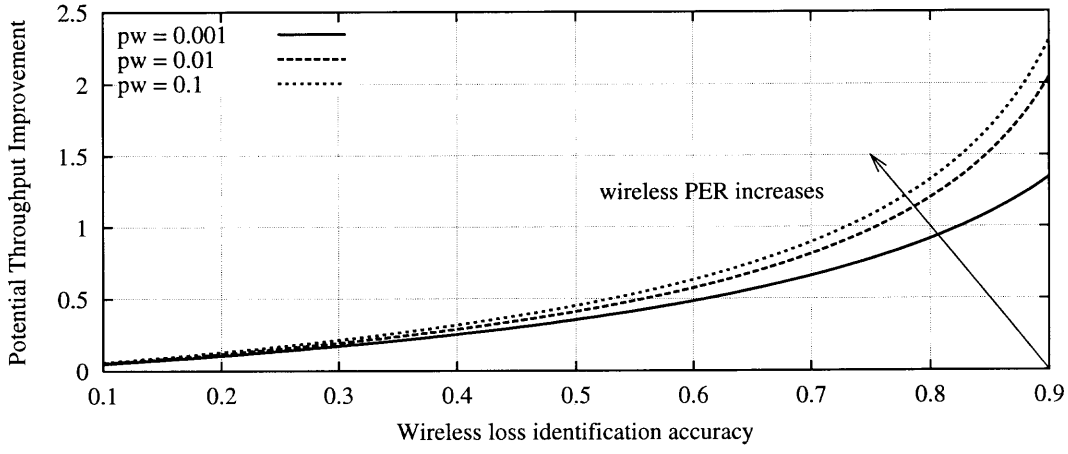
Despite the numerical inaccuracy in the high PER region due to the simplification in the modeling process, one important aspect is clearly observable from the graph (and is in accordance with similar studies using detailed models). That is, a higher wireless loss identification accuracy (A_w) and a lower congestion induced PER ($p_{c,l}$) lead to higher improvement of TCP's performance in lossy wireless

networks. This is true for both the standard AIMD congestion control algorithm as well as for the proposed TCP-Jersey's AIAD congestion control algorithm.

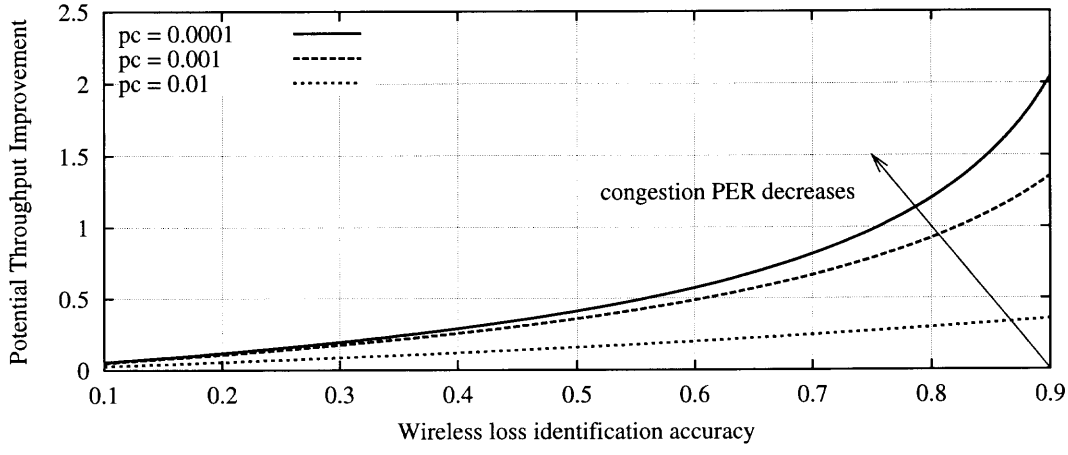
In Figure 8.6, the impact of the wireless loss identification accuracy, A_w, to the potential throughput improvement is shown. In Figure 8.6(a), three curves of $\Delta_{AIMD}(A_c, A_w)$ are plotted for the wireless PER, $p_{w,l}$, of 0.001, 0.01, and 0.1, respectively, all with a fixed congestion PER, $p_{c,l}$ of 0.0001, and A_c of 1. In Figure 8.6(b), three curves of $\Delta_{AIMD}(A_c, A_w)$ are plotted for the congestion PER, $p_{c,l}$, of 0.0001, 0.001, and 0.01, respectively, all with a fixed wireless PER, $p_{w,l}$ of 0.01, and A_c of 1. It is seen from the figures that, first and most obviously, higher A_w always provides more improvement in TCP's throughput. And, as wireless PER increases, the improvement becomes larger. Secondly, A_w has a stronger impact on TCP's performance improvement, especially in the case of a low congestion PER. This can be observed by the relative spacing among the three curves in Figure 8.6(b) as compared to the curve spacing in Figure 8.6(a).

Similarly, Figure 8.7 shows the case for $\Delta_{AIAD}(A_c, A_w)$, which results in the same observations. Therefore, for both AIMD and AIAD congestion control, the accuracy of identifying wireless error induced packet losses has a strong impact on the TCP's performance, particularly when the network is operating in a moderate to low congestion level. It is necessary to emphasize that since the differential equations used in modeling the TCP dynamics ignore many protocol details such as slow-start, fast retransmit, fast recovery, and retransmit timeout, the model presented before is analytically tractable and insightful for the analysis of the long-term behavior of the dynamical system, but is inaccurate in representing its transient behavior. In particular, when the packet loss rate is high, protocol details that have been ignored in the model become the dominating factors of the throughput and other performance metrics. Therefore, the numerical figures in the preceding graphs should be taken as illustrative instead of being used directly.

Nevertheless, the tendency that higher A_w and lower $p_{c,l}$ leads to more performance improvement is clear and has also been verified by other studies using a different (more detailed but less convenient) model.

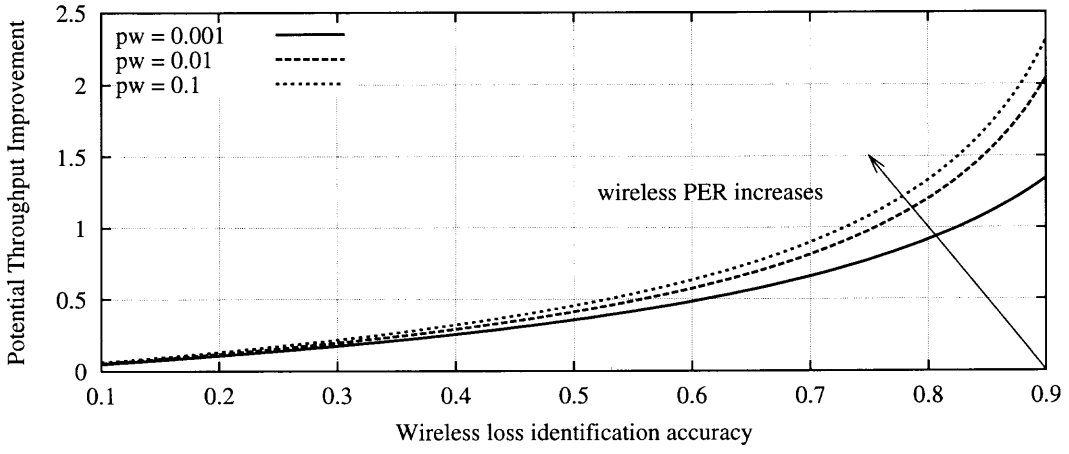


(a) $\Delta_{AIMD}(A_c, A_w)$, varying wireless PER.

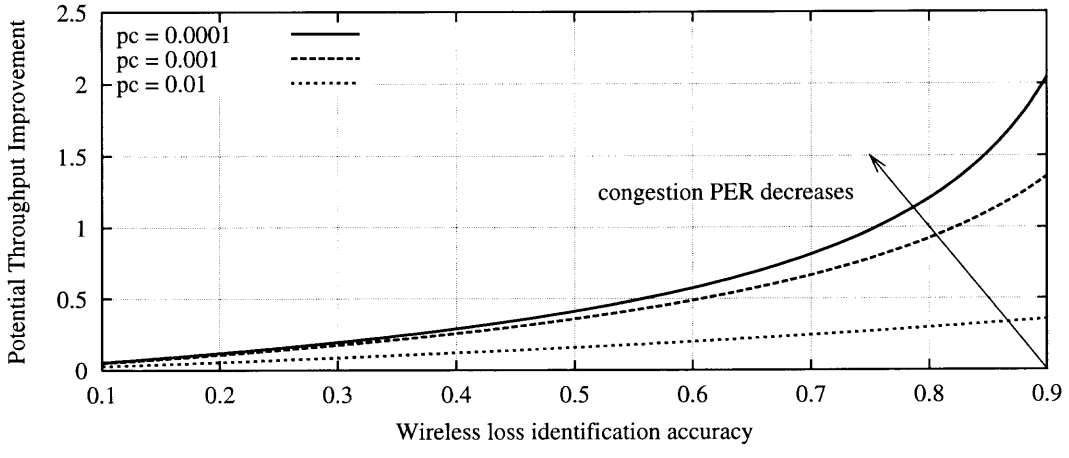


(b) $\Delta_{AIMD}(A_c, A_w)$, varying congestion PER.

Figure 8.6 AIMD potential throughput improvement vs. wireless packet loss identification accuracy A_w. ('pc' in the legend stands for $p_{c,l}$, and 'pw' for $p_{w,l}$)



(a) $\Delta_{AIAD}(A_c, A_w)$, varying wireless PER.



(b) $\Delta_{AIAD}(A_c, A_w)$, varying congestion PER.

Figure 8.7 AIAD potential throughput improvement vs. wireless packet loss identification accuracy A_w. τ and d are 0,05 and 0.002, respectively. ('pc' in the legend stands for $p_{c,l}$, and 'pw' for $p_{w,l}$)

It is also insightful to observe the potential throughput improvement of an AIAD source equipped with a loss differentiator over an AIMD source also equipped with the same loss differentiator. We can define

$$\Delta_{\frac{AIAD}{AIMD}} = \frac{x_{AIAD}(Ac, Aw)}{x_{AIMD}(Ac, Aw)} - 1 \tag{8.29}$$

as the improvement index and plot in a three dimensional graph where the variables are the wireless PER and the accruracy of classifying wireless related packet losses, i.e., Aw, as show in Figure 8.8. In this figure, the solid line plot represents $\frac{x_{AIAD}(1,Aw)}{x_{AIMD}(1,Aw)} - 1$, meaning both algorithms are equipped with the same loss differentiator; the dashed line plot represents $\frac{x_{AIAD}(1,Aw)}{x_{AIMD}(1,0)} - 1$, which is the case of TCP-Jersey vs. the standard TCP Reno

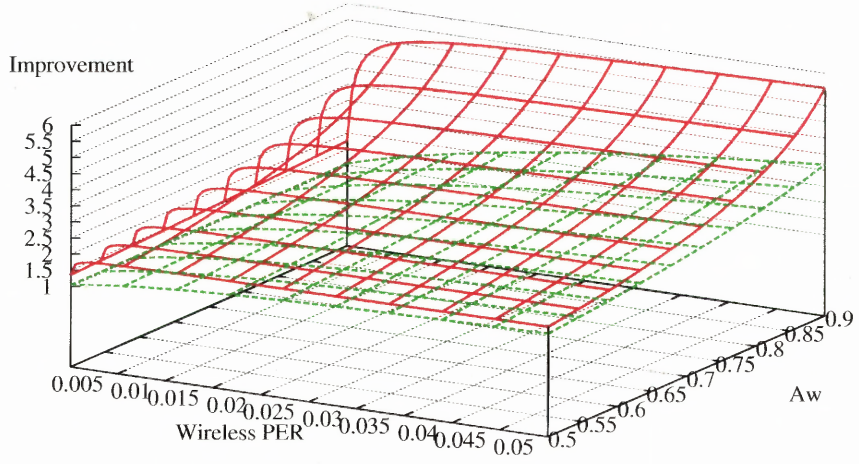


Figure 8.8 Potential throughput improvement, TCP-Jersey vs. TCP Reno when sources are equipped with loss differentiator. The solid line is $\frac{x_{AIAD}(1,Aw)}{x_{AIMD}(1,Aw)} - 1$, and the dashed line is $\frac{x_{AIAD}(1,Aw)}{x_{AIMD}(1,0)} - 1$.

As is seen from the plots, AIAD congestion control algorithm equipped with loss differentiator would outperform AIMD congestion control whether or not it is equipped with loss differentiator.

8.5 Dynamic Thresholding in CW Marking

The second key component of the proposed end-to-end solution architecture to TCP's performance enhancement in wireless networks is the packet loss differentiation through active queue management (AQM), in particular the congestion warning (CW) packet marking using a simple AQM scheme, which is essentially a revised and simplified explicit congestion notification (ECN) [55] link algorithm. The TCP-Jersey's sender utilizes the explicit congestion marks in the DUPACK packets to determine whether the previous packet loss was due to congestion or wireless link errors, and hence achieving the loss differentiation.

There are two heuristic parameters associated with this CW marking scheme. First, the CW marking threshold is based on the average buffer occupancy (or, queue length), which is computed at the link by exponentially weighted moving averaging (EWMA) of the instantaneous queue length at the time of each packet arrival. The update rule of the average queue length $\hat{b}_l$ at time n is

$$\hat{b}_l[n] = \alpha_l \hat{b}_l[n-1] + (1-\alpha_l) b_l[n-1],\ l \in \boldsymbol{L_i}, \tag{8.30}$$

for some constant $1 < \alpha_l < 1$. Or, in a form of differential equation, it is

$$\dot{\hat{b}}_l(t) = -\alpha_l c_l (\hat{b}_l(t) - b_l(t)), \tag{8.31}$$

where the sampling interval is taken to be $1/c_l$ because the computation of the average queue length takes place upon each packet arrival to the link. The setting of the EWMA weight, $1-\alpha_l$, is subjective. In particular, TCP-Jersey uses 0.2, while in [55] the RED/ECN AQM router uses 0.002.

Second, the currently proposed TCP-Jersey's link algorithm, i.e., the CW marking, is essentially a *threshold marking* [71, 107] scheme, where the packets are marked when the objective function (in this case the average queue length at the link) has crossed a pre-determined threshold. The current TCP-Jersey's setting

of the CW threshold to $1/3$ of the buffer capacity is ad hoc at best. In the sections that follow, we investigate ways of reducing the heuristics in the current proposed CW marking scheme.

8.5.1 Use of Persistent Queue Length

The first approach to reduce the heuristics in the CW parameter setting is a simple modification to what has already been implemented in TCP-Jersey. Instead of computing the average queue length by EWMA of the instantaneous queue length, we propose to calculate the *persistent queue length* as our indication of link congestion. We define the persistent queue length as the sustained buffer occupancy of the link during a time interval.

Average queue length by EWMA or any other low-pass filtering of the instantaneous queue occupancy does provide an indication of the underlying link congestion level, since congestion occurs whenever the low-frequency input traffic rate exceeds the link capacity [108]. Link congestion, and hence packet loss, occurs when there is a persistent overage of the aggregate input flow rate over the link's output capacity that cannot be alleviated by the link's limited buffer space. Given that Internet traffic is bursty in nature, short-lived transient input/output rate mismatches do occur. AQM works in concert with TCP congestion control, which in turn forms the decentralized algorithm of today's Internet congestion control. The goal of congestion control is to match up the aggregate rate of input flowing to a link to its output capacity and drain the persistent queue. Too small the EWMA weight produces smoother average queue length but makes it respond sluggishly to sudden increase in input flow rate, and therefore fails to provide congestion warning signals in time and causes unnecessary packet losses. Too large the weight makes the average queue length too close to the instantaneous queue length that

does not represent persistent congestion, and therefore causes unnecessary frequent congestion warning signals that results in network inefficiency.

The persistent queue length can be calculated by the link by measuring its buffer occupancy that does not drain in a given interval (for example, in a round-trip propagation delay). The computation is simpler than EWMA and can easily be implemented at the router's hardware with much less computational demand than EWMA. The use of the estimation of the persistent queue length instead of the average queue length is also proposed in [79]. The implementation of the persistent queue length estimation can be summarized in the following pseudo-code as shown in Figure 8.9. In essence, the persistent queue length is computed by taking the minimum queue length seen by an arriving packet during the last computation period.

```
On packet departure do:
    if(instantaneous_queue < minimum_queue)
        minimum_queue = instantaneous_queue;
    end if
```

```
When the persistent queue computation timer expires, do:
    persistent_queue = minimum_queue;
    minimum_queue = instantaneous_queue;
    reschedule_timer(computation interval);
```

Figure 8.9 Pseudo-code of computing the persistent queue length at the link.

We have replaced the EWMA average queue computation in the original TCP-Jersey's link algorithm with the persistent queue computation listed in Figure 8.9, and conducted simulations with otherwise the same configurations as reported in the previous chapters. In what follows, and throughout the rest of this chapter, we name the original average queue length based congestion marking scheme

as CWAQ, and the persistent queue length based congestion marking scheme as CWPQ. The results of all simulation runs are almost identical to those already been reported in previous chapters. Therefore, instead of repeating all the simulation descriptions and the similar results, Figure 8.11 plots the throughput vs. random wireless error rate (in terms of packet error rate) of various TCP schemes under the network configuration depicted in Figure 8.10, where a long-lived TCP flow is sending data from node S to node D, and the link between the node BS and D is simulated as the wireless link with random bit errors causing packet losses. The CW marking function is implemented at the wireless link. The buffer space on link between nodes S and BS is 500 packets and that of link between nodes BS and D is 50 packets. All data packets are 100 bytes in size. Random packet errors are imposed on both directions of the wireless link.

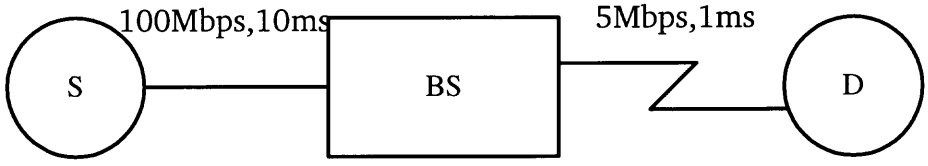


Figure 8.10 Simulation network configuration.

In the figure, TCP-Jersey with average queue length based CW marking is labeled as "Jersey/CWAQ", while the one with persistent queue estimation is labeled as "Jersey/CWPQ". It can be seen from the plot that TCP-Jersey using the persistent queue length estimation at the link performs very closely (almost indistinguishable) with the one using the average queue length computation. The advantage of persistent queue length computation as listed in Figure 8.9 is that it only involves one comparison operation and spares the router any multiplications, and thus can be easily implemented in hardware with minimum impact on the router's performance.

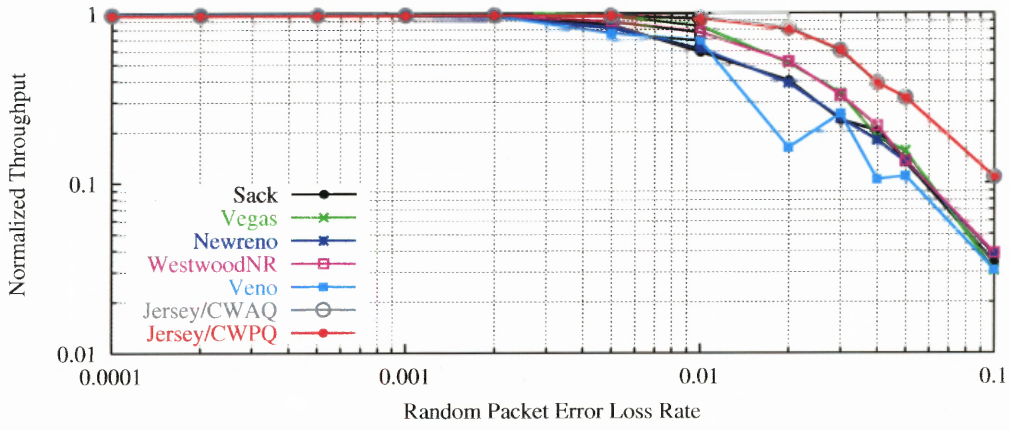


Figure 8.11 Throughput performance of TCP-Jersey using persistent queue length.

8.5.2 Reduce Queue Oscillation by Continuous Packet Marking

Although the congestion warning (CW) packet marking algorithm proposed in the preceding subsection using the persistent queue length estimation results in a very satifactory performance while requiring much less computational power at the link, it is still a threshold marking scheme in nature.

From the perspective of control theory, the aggregate congestion indication [10] $q_i(t)$ is the control signal input from the AQM to the TCP source's window control dynamics, from which the resulting output is the TCP source's congestion window, $w_i(t)$, or, equivelantly the TCP source's transmission rate $x_i(t)$. This relationship can be represented by the diagram as shown in Figure 8.12.

The threshold marking scheme is therefore regarded to as an on-off control scheme, which is inherently oscillatory. This is evident in Figure 8.13, in which the actual trajectory of the packet marking probability (i.e., the measured $q_i(t)$) is plotted. From the figure, it is observed that thresholding marking leads to an oscillatory evolution of the congestion indication signal to its equilibrium point.

The consquence of an oscillatory control, in this case, is the oscillatory queue occupancy at the link. This is shown in Figure 8.14, in which the actual queue

[10] Recall the basic network model presented in Section 8.2.

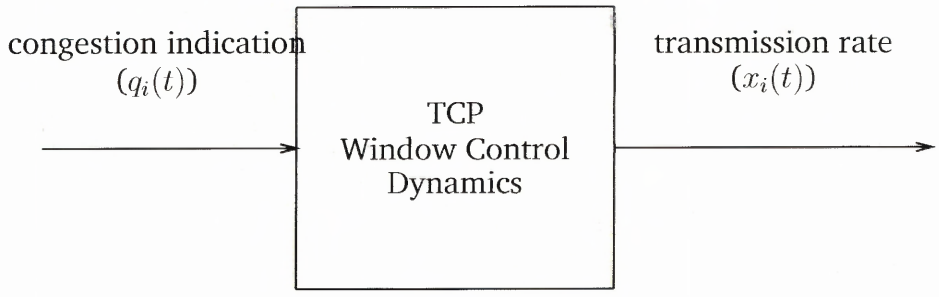


Figure 8.12 Input-Output diagram of TCP congestion control.

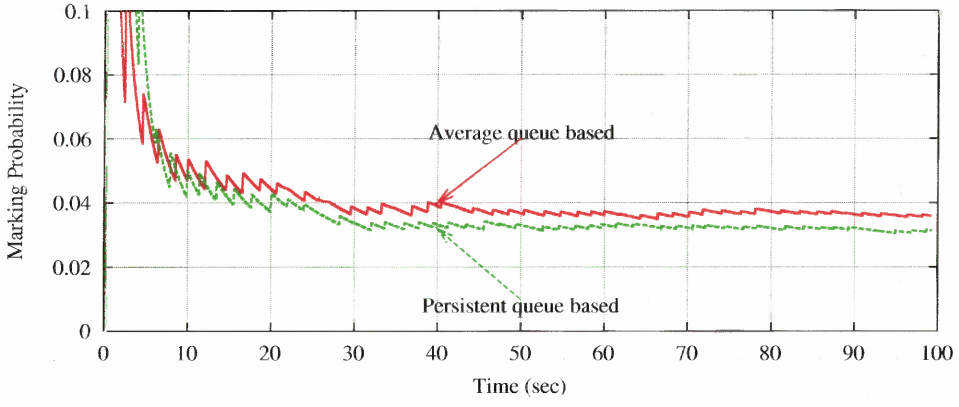


Figure 8.13 Oscillation of marking probability due to on-off control.

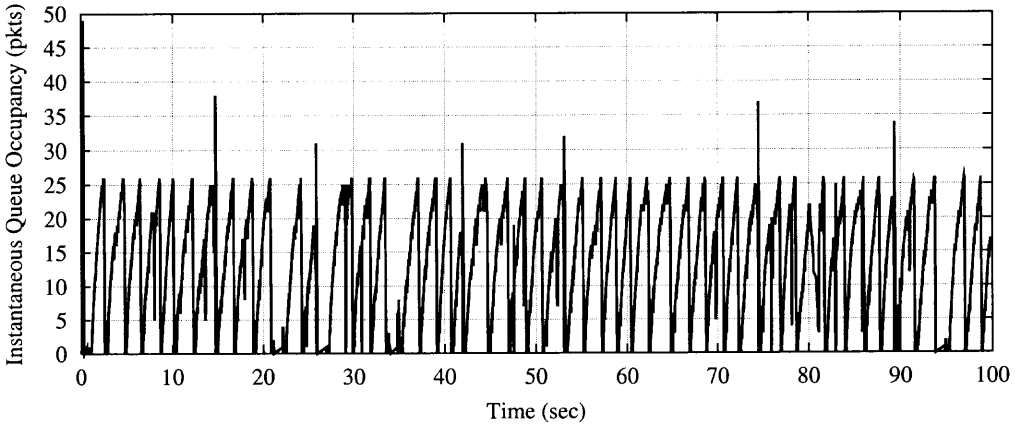
occupancy of the CW router is plotted. This is true for both the average queue length based CW and the persistent queue length based CW marking algorithms.

Queue oscillation is undesirable for two reasons. First, oscillatory queue length leads to a varying end-to-end delay and a large delay jitter, which may severely impair the quality of some of the applications that have stringent requirement on delay and/or jitter bounds. Second, recall that in the local stability conditions derived in the analysis of AIAD congestion control in Chapter 6, the queuing delay is required to be upper bounded by a value that is, in general, inversely proportional to p'_0 (i.e., the rate of change of the marking probability with respect to the aggregate flow rate, see Equations (6.14) and (6.16)). The jumps of the marking probability as observed in Figure 8.13 implies that p'_0 can become infinitely large, and thus making the stability condition extremely difficult to be maintained. A detailed treatment of the theshold marking scheme can be found in [71], where a Markov Chain analysis of the queuing delay under threshold marking[11] for a different congestion control algorithm is discussed.

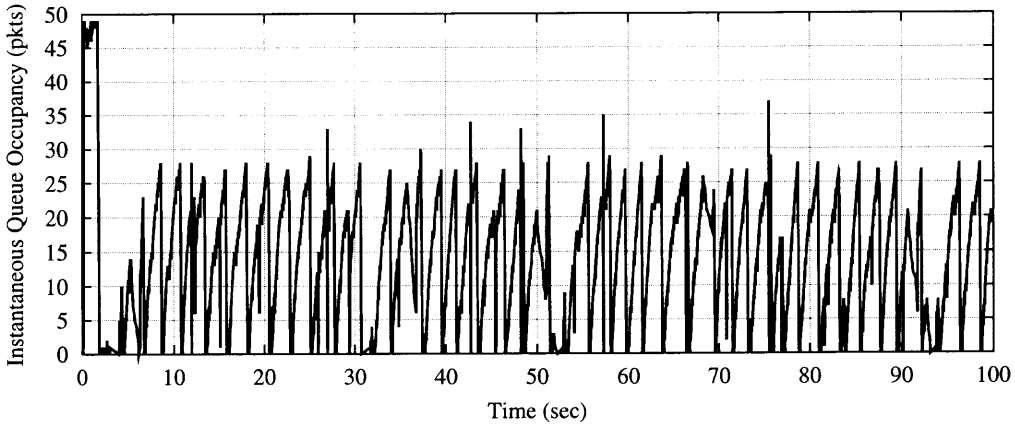
It is concluded that a "smoother" congestion indication function leads to a more stable network. Therefore, instead of devising an algorithm that adaptively changes the marking threshold, as opposed to the current CW solution that fixes the marking threshold as a pre-configured parameter (e.g., from fixed 1/3 to some other value determined dynamically), in subsequent sections, we investigate a different type of congestion indication function (or, marking function) that changes the

[11]Modeling of threshold marking/dropping queue dynamics, a general case of a tail-dropping queue, is very difficult, regardless its simplicity in the queue operation. The difficulty lies in the computation of the tail probability of the queue (i.e., the probability of the queue occupancy being greater than a threshold) for a general class of packet arriving processes. The authors in [71] illustrated the modeling of thresholding marking by assuming the packet arrival process as a Poisson process and derived the probabilities for M/M1 and M/D/1 queuing systems. Other studies of the problem can be found in, e.g., [109] and the references therein.

marking probability directly in a continuous fashion, in place of the less desirable on-off control.



(a) Average queue length based threshold marking.



(b) Persistent queue length based threshold marking.

Figure 8.14 Oscillation of queue occupancy due to threshold marking. In both cases, the marking threshold is set to be at 25 packets for a queue of 50 packets buffer space.

8.6 Virtual Capacity Rate Regulation Active Queue Management

To serve our purpose, that is an end-to-end loss differentiation algorithm for better TCP performance in wireless networks, in this section, we explore and investigate if congestion induced packet losses can be eliminated or drastically reduced to a level such that this type of packet loss only occurs sporadically. The rationale behind this line of thought is that if congestion induced packet loss can be eliminated or reduced to only occur sporadically, then the accuracy of wireless error induced packet loss identification (A_w) can be increased.

Unless the nature of packet loss is reported explicitly, the TCP source has to rely on its own loss differentiator[12] in order to achieve better throughput in a lossy environment such as a wireless network. Explicit loss type notification schemes have been proposed, for example, ETEN in [110], where corruption detection mechanisms are deployed at the link level and the packet corruption information, either as per-packet notification or as cumulative corruption rate or probability, are fed back via either in-band or out-band signaling. Schemes such as ETEN represent a different design space than the one presented in this dissertation, where no wireless-aware link equipment is required. The architecture proposed in this dissertation is based on adaptive congestion control at the source, which itself is beneficial to TCP performance in both wireless and non-wireless environments, and implicit loss type differentiation using the congestion indications (i.e., packet markings) from many routers equipped with AQM.

As it has been discussed in Section 8.4.1, the TS-ARE based AIAD congestion control of TCP-Jersey, as compared to the standard TCP protocol, can handle sporadic congestion-related packet loss better (e.g., see (8.24)). Also in Section

[12]The standard TCP that does not employ a loss differentiator can be regarded as having a loss differentiator that has 100% misclassification rate of non-congestion, or, wireless packet losses.

8.4.2, we concluded that a higher accuracy of identifying wireless error induced packet loss has a stronger positive impact on the performance improvement.

Our goal of this section is to have an packet marking AQM that serves as well as the CW threhold marking in terms of aiding the TCP source in distinguishing loss types, and at the same time results in a much stable queue that would benefit delay and jitter bounded applications. In what follows we investigate an AQM mechanism based on the concept of *virtual capacity* and we call such an AQM a *virtual capacity rate regulation* AQM, or VCRR. We study the performance of VCRR in place of the original CW in TCP-Jersey.

8.6.1 Congestion Indication via VCRR AQM

The VCRR AQM at link l can be considered as a rate regulator that matches its input flow rate to the *virtual capacity* $\tilde{c}_l$, where the relationship between the virtual capacity and the actual capacity of the link is

$$\tilde{c}_l = \gamma c_l, \tag{8.32}$$

in which $0 < \gamma < 1$ is usually chosen to be close to 1, e.g., 0.95. The parameter γ is essentially a target link utilization that the control of the AQM attempts to achieve. The benefit of using the virtual capacity rather than the actual link capacity is that, by sacrificing slightly the achievable link utilization, the steady-state aggregate input rate to the link is regulated to be slightly smaller than the output capacity, and hence avoiding any persistent queuing delays.

Paired with $\tilde{c}_l$, the *virtual queue* $\tilde{b}_l$ can be defined as

$$\dot{\tilde{b}}_l(t) = y_l(t) - \tilde{c}_l, \text{ for } \tilde{b}_l(t) > 0, \tag{8.33}$$

meaning the virtual queue increases with the rate of the aggregate input and decreases with the rate of the virtual capacity. The VCRR link algorithm marks

packets when the aggregate arriving flow rate is exceeding the virtual capacity. The packet marking rate or probability is updated by

$$\dot{p}_l(t) = \frac{(y_l(t) - \tilde{c}_l)^+}{\tilde{c}_l}, \tag{8.34}$$

where $(x)^+ = min(0, x)$. Therefore, in the steady-state, the aggregate input rate equals to the virtual capacity, achieving the target link utilization γ. Since γ is less than 1, the link should experience no persistent congestion except for sporadic congestion-related packet drops due to transient burstiness of the incoming traffic.

Assuming zero initial conditions, integrating (8.34) results in

$$\begin{aligned} p_l(t) &= \frac{1}{\tilde{c}_l} \int_0^t (y_l(\xi) - c_l) d\xi \\ &= \frac{1}{\tilde{c}_l} \tilde{b}_l(t) \end{aligned} \tag{8.35}$$

Therefore, from the control theory perspective, the packet marking control law at the VCRR link is a P-controller with a proportional control gain of $1/\tilde{c}_l$.

Implementation of (8.34) is very simple by discretization of the differential equation as follows

$$p_l[(n+1)\delta t] = p_l[n\delta t] + \frac{y_l[n\delta t]\delta t - \tilde{c}_l \delta t}{\tilde{c}_l}, \tag{8.36}$$

where δt is the control update interval. Since $y_l[n\delta t]\delta t$ is approximately the amount of traffic arrived to the link during the interval, the VCRR link algorithm maintains a counter that increments as packets arrive; at the end of the control interval, the marking probability is updated by the value obtained by subtracting the counter by the amount of traffic drained by the virtual link and dividing the result by the virtual capacity. Thus, the implementation of the VCRR link algorithm does not necessarily involve estimation of the aggregate input traffic rate.

The drawback of this implementation, however, is in its inaccuracy in approximating the aggregate input rate $y_l(t)$, since the implementation implies that $y_l(t)$

does not change during the control period δt. The discretization in (8.36) is a zero-order-hold (ZOH) transformation, which introcudes discretization error for fast changing signals. Thus, there is a trade-off between accuracy and complexity. As a remedy, first-order-hold (FOH) and/or trapezoidal transformation [59] can be used. Other ways to improve the accuracy in updating $p_l(t)$ is to directly estimate the aggregate input flow rate $y_l(t)$ at the link. This can be done through the time-sliding window (TSW) estimation algorithm introduced by Clark and Fang [60] as

$$\hat{y}_l[n+1] = \frac{\hat{y}_l[n] + s_n}{T_w + \delta_n}, \tag{8.37}$$

where s_n is the size of packet n, T_w is the sliding window size, and δ_n is the time lag between packets n and $n+1$ arriving at the link. Or, the link can estimate the flow rate by the moving average algorithm proposed by Stoica et al. [83] in the core stateless fair queuing (CSFQ) scheme as

$$\hat{y}_l[n+1] = \frac{(1 - e^{-\delta_n/\omega})s_n}{\delta_n} + e^{-\delta_n/\omega}\hat{y}_l[n], \tag{8.38}$$

where ω is a constant parameter.

The psuedo-code of the VCRR update rule without the flow rate estimation is listed in Figure 8.15.

On packet arrival do: `counter += packet size;`

When the control period ends, do:
 	
`p += (counter - virtual_capacity x interval) / interval;`
 	
`counter = 0;`

Figure 8.15 Pseudo-code of VCRR updating marking probability at the link.

The TCP/AQM system is a feedback control system with a time delay that is roughly equal to the round-trip delay. The source in such a control system cannot respond any faster than the dominant delay of the system[13]. Therefore, the control interval in the above can be set to the round-trip delay, or a reasonable estimate of such by the link.

8.6.2 Simulation and Performance Evaluation

The VCRR link algorithm described above has been implemented in the TCP-Jersey in place of the threshold marking CW algorithm discussed before. The resulting new TCP-Jersey is tested in the simulation network depicted in Figure 8.10. Figure 8.16 plots the normalized throughputs attained by TCP-Jersey using different packet marking algorithms under the same network settings. Note that in order to magnify the differences between the throughputs, the y-axis is not in log scale, as opposed to previous figures. Since the target utilization of VCRR is set at 0.95, the curve representing the VCRR algorithm has a slightly lower normalized throughput than the other two. If all normalized throughputs are scaled by their respective target utilizations, the three curves will overlap with very little distinguishable differences. In other words, the TCP-Jersey source congestion control combined with the VCRR link packet marking algorithm can achieve the same performance improvement as the original TCP-Jersey.

We repeated the simulation with the terget utilization γ set to 100%, and plotted the queue occupancies, the actual congestion indication trajectory as measured by the ratio of marked packets and arrived packets in time, and the TCP source's end-to-end RTT as well as RTT variation measurements in the following figures.

[13]For studies on the dominant time scale (DTS) in Internet congestion control, see [111] and the references therein.

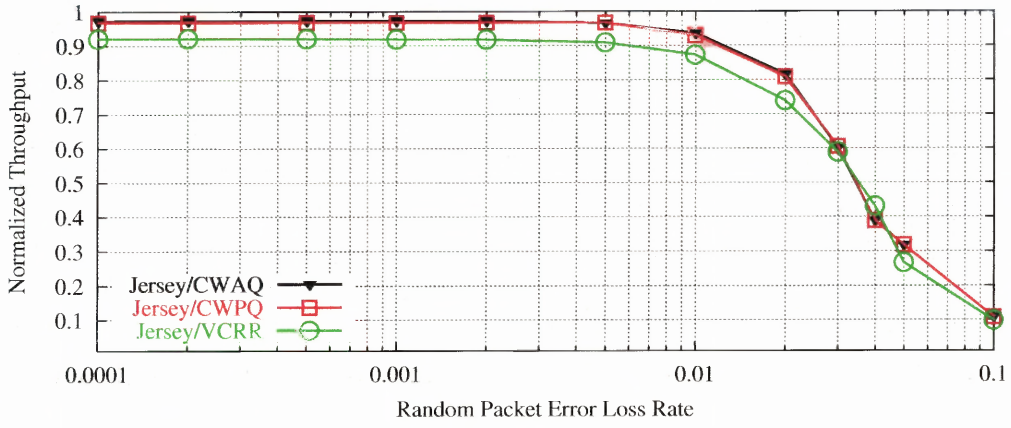
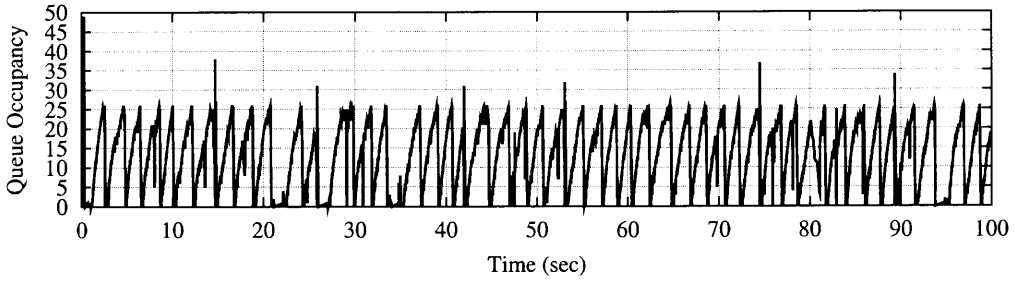


Figure 8.16 Normalized throughputs attained by TCP-Jersey using different packet marking algorithms. Target utilization of VCRR is set at 0.95, and the control interval is 20ms.

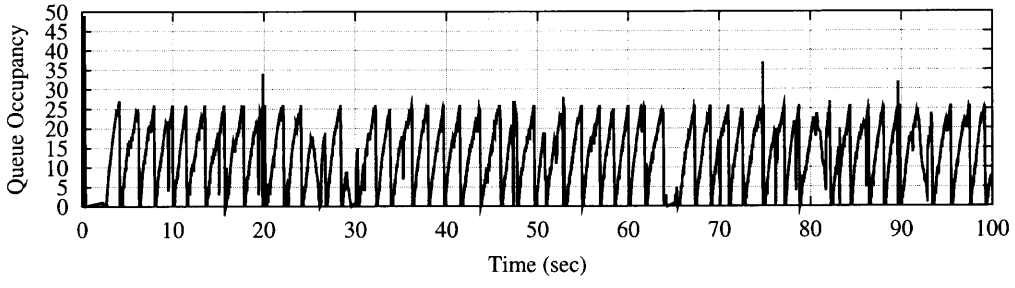
Figure 8.17 shows the instantaneous queue occupancies at the bottleneck wireless link under the different packet marking algorithms used at the link. For a fair comparison, the target link utilization γ is set at 100% for VCRR. It is observed that as compared to the threshold marking schemes of CWAQ and CWPQ, the continuous marking function of VCRR results in a much lower average queue occupancy and a much less oscillatory queue length.

In Figure 8.18, we plot the packet marking probability observed by the TCP-Jersey source when the VCRR link algorithm is used, along with those when CWAQ and CWPQ link algorithms are used. The marking probability approaches to its steady-state in a much less oscillatory fashion as our analysis has predicted.

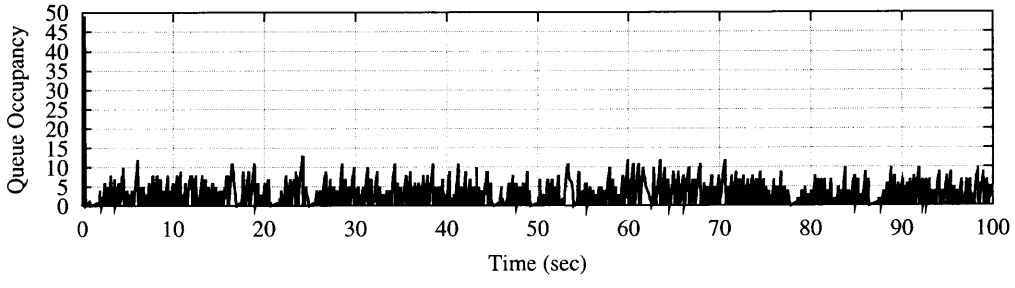
We also plot the statictics of round-trip times measured by the TCP-Jersey source under different packet marking strategies in Figure 8.19 and Figure 8.20. As shown in Figure 8.19, with VCRR, packets experience a smaller and steadier end-to-end delay than with CWAQ and CWPQ. The smaller and steadier delay jitter shown in Figure 8.20 indicates another desirable property of VCRR for delay-sensitive applications.



(a) CWAQ.



(b) CWPQ.



(c) VCRR.

Figure 8.17 Instantaneous queue occupancy by three marking algorithms under the same network configuration, where VCRR has $\gamma = 1.0$, CWAQ and CWPQ both have threshold at 25 packets (50% of the buffer capacity). Wireless PER is 1%.

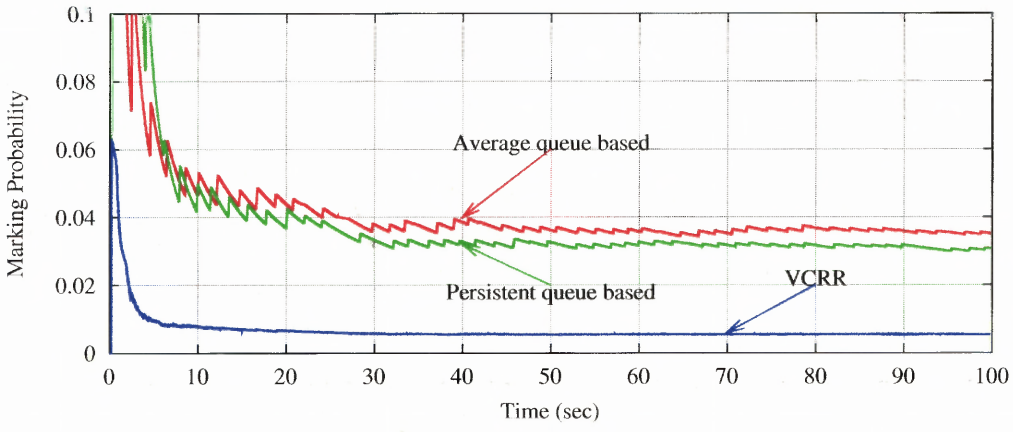
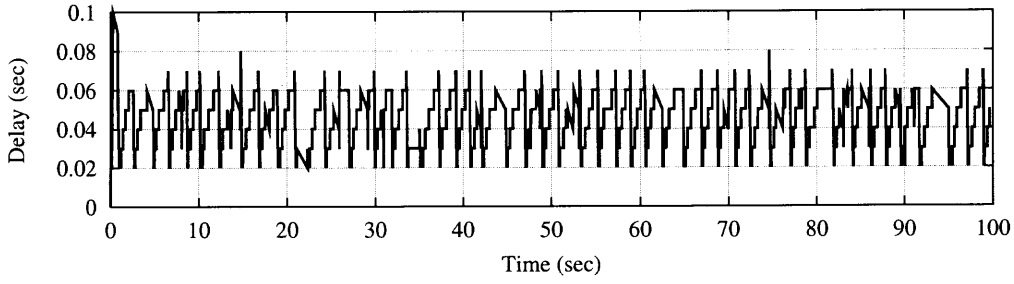
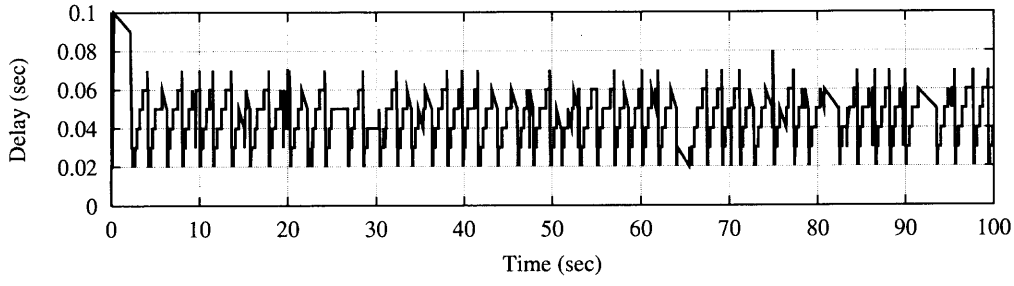


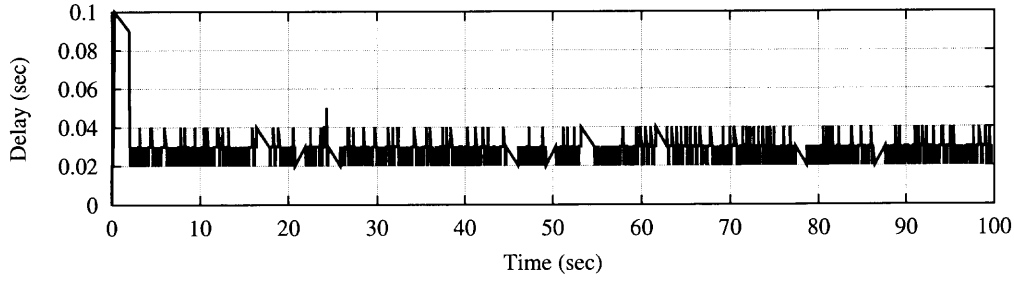
Figure 8.18 Actual congestion indication as measured packet marking probability by three marking algorithms under the same network configuration, where VCRR has $\gamma = 1.0$, CWAQ and CWPQ both have threshold at 25 packets (50% of the buffer capacity). Wireless PER is 1%.



(a) CWAQ.

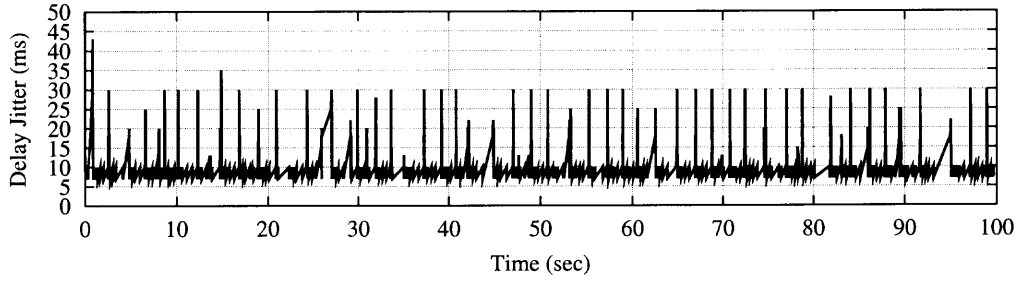


(b) CWPQ.

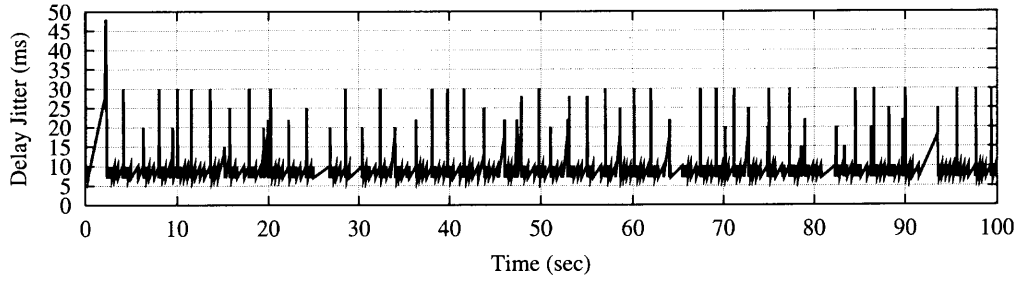


(c) VCRR.

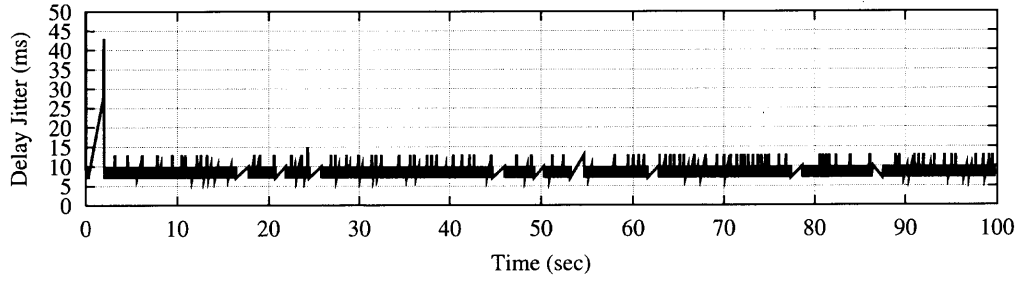
Figure 8.19 Round trip delay measured by TCP source using three marking algorithms under the same network configuration, where VCRR has $\gamma = 1.0$, CWAQ and CWPQ both have threshold at 25 packets (50% of the buffer capacity). Wireless PER is 1%.



(a) CWAQ.



(b) CWPQ.



(c) VCRR.

Figure 8.20 Round trip delay jitter (RTT variation) measured by TCP source using three marking algorithms under the same network configuration, where VCRR has $\gamma = 1.0$, CWAQ and CWPQ both have threshold at 25 packets (50% of the buffer capacity). Wireless PER is 1%.

A second set of simulations have been conducted in the network topology depicted in Figure 8.21 where three long-lived TCP-Jersey flows send data to node D separately from nodes S1, S2, and S3, respectively. Since each source node has an access link capacity of 100Mbps to the base station, the three TCP flows are competing for the bottleneck wireless link of capacity of 5Mpbs. During the 500s simulation run, flow 1 starts the sending at the beginning of the simulation, flow 2 starts at 100s, and flow 3 starts at 200s. Throughout the simulation, the wireless link is configured to randomly drop 1% of the passing packets, i.e., the wireles PER is set at 1%.

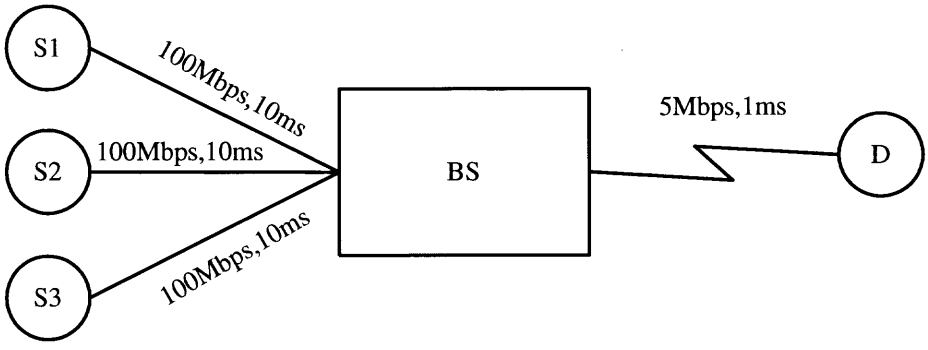
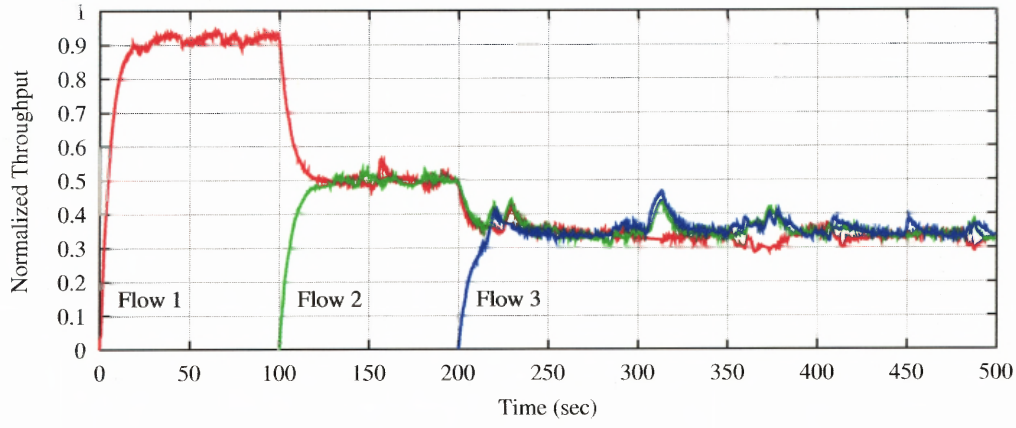


Figure 8.21 Network topology and configuration for simulation of competing TCP flows.

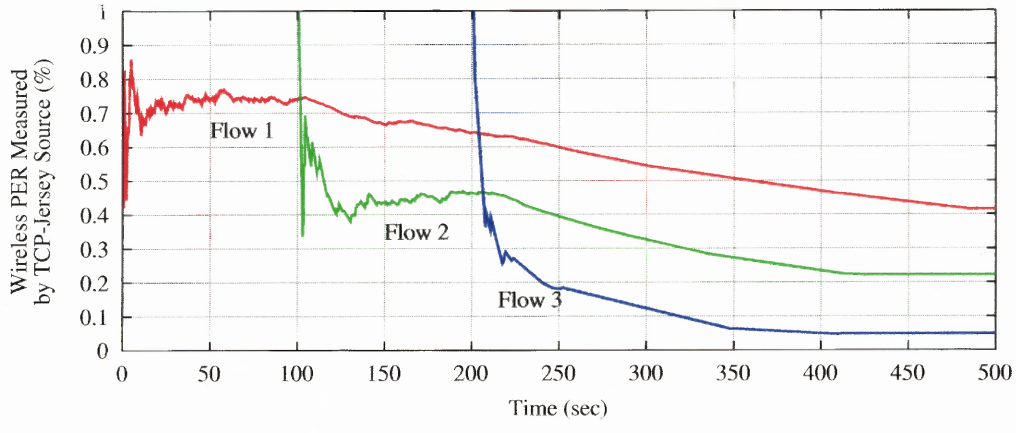
We record each flow's attained throughput over time and plot them in Figure 8.22(a). As the graph shows, the TCP-Jersey's source AIAD congestion control algorithm and the VCRR packet marking algorithm at the wireless link as the congestion indication work in concert in providing a stable and fair resource (i.e., the bottleneck capacity) sharing among the competing flows.

Figure 8.22(b) shows the actual wireless PER measured at the TCP-Jersey source by each flow as the number of detected wireless link error induced packet loss divided by the number of packets the source has sent.

In Figure 8.23, the trajectory of the actual control signal from VCRR and its derivative are plotted in Figure 8.23(a) and Figure 8.23(b), respectively. As



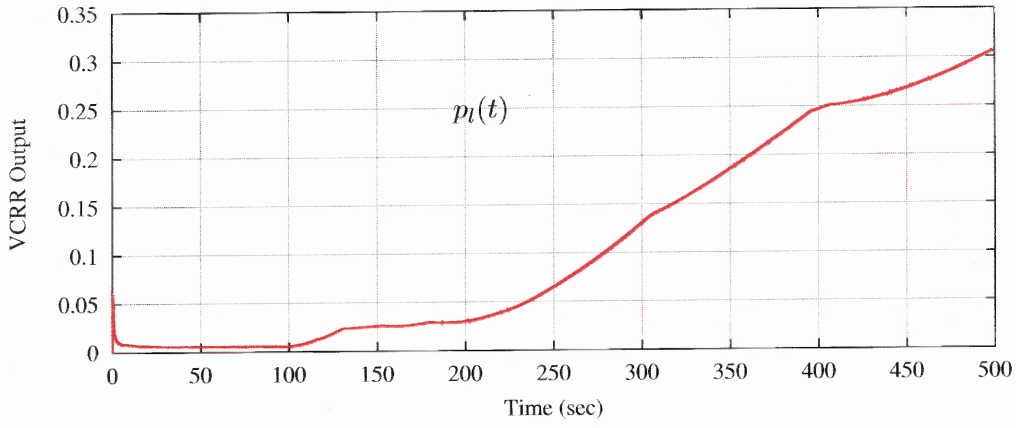
(a) Normalized throughputs.



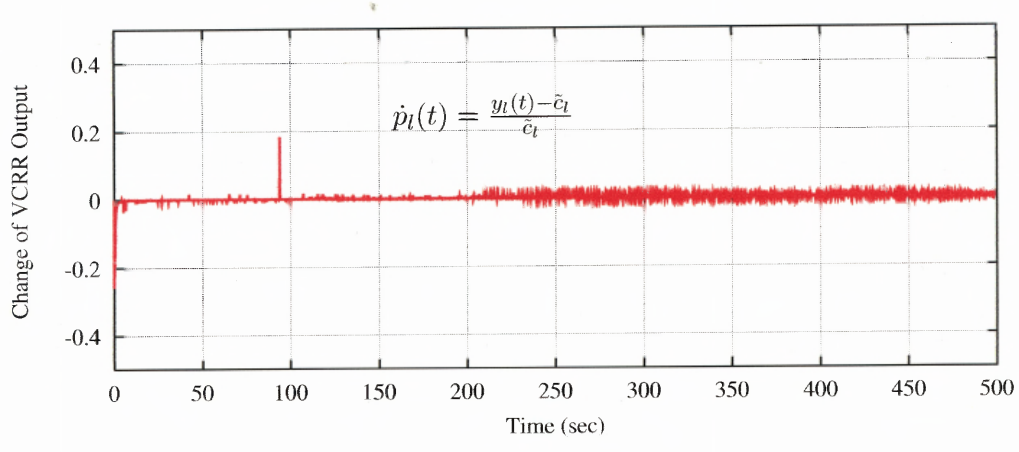
(b) Wireless PER measured by TCP-Jersey sources.

Figure 8.22 End-to-end throughputs and wireless PER measured by sources when 3 TCP-Jersey flows sharing a 5Mbps wireless link. The marking algorithm on the wireless link is VCRR, and the wireless PER inflicted in the link is 1%.

the graphs show, VCRR's output, i.e., the congestion control signal $p_l(t)$ to the soutce, reaches its steady-state fairly quickly and stays at the equilibrium with small variations. It is also observed that during the entire 5000s of simulation, only a total of 21 packets are dropped because of buffer overflow at the bottleneck. These congestion induced packet losses constitute only a neglactable fraction of congestion PER at the order of 10^{-3}%. A closer inspection of these congestion related losses shows that all of them are due to the slow-start phases of each flow, during which the source's congestion window increases exponentially. No other congestion related packet losses occurred during the simulation, implying that the AIAD congestion control at the TCP-Jersey's sources is able to effectively avoid congestion.



(a) Congestion indication.



(b) Derivative of congestion indication.

Figure 8.23 Congestion indication signal output from VCRR and its derivative when 3 TCP-Jersey flows sharing a 5Mbps wireless link. The wireless PER inflicted on the link is 1%.

In a third set of simulations, we increase the random wireless PER from 1% to 5%. At such a high random packet loss rate, TCP-Jersey with all three marking algorithms (i.e., CWAQ, CWPQ, and VCRR) can maintain about 30% of the throughput it would otherwise attain in an error-free link. This can be seen from the data points in Figure 8.16. Note that the graphs in Figure 8.16 were obtained in a simulation scenario where there was only one TCP flow in the network. The following Figure 8.24 shows the throughputs attained by three competing TCP-Jersey flows under the heavy 5% random packet loss rate at the wireless link. It is seen that all three flows are able to maintain a throughput that is about 30% of the total capacity, making the total link utilization at about 90%, which is very close to the target utilization of the VCRR controller. Figure 8.24 also implies that persistent congestion at the bottleneck is effectively avoided by the operations of TCP-Jersey's AIAD congestion control in concert with the VCRR's congestion signaling (i.e., packet marking) mechanism. The loss differentiation algorithm, which is part of the AIAD algorithm, successfully lifts the attainable throughput of TCP-Jersey from all other TCP variants under the heavy random packet loss scenario.

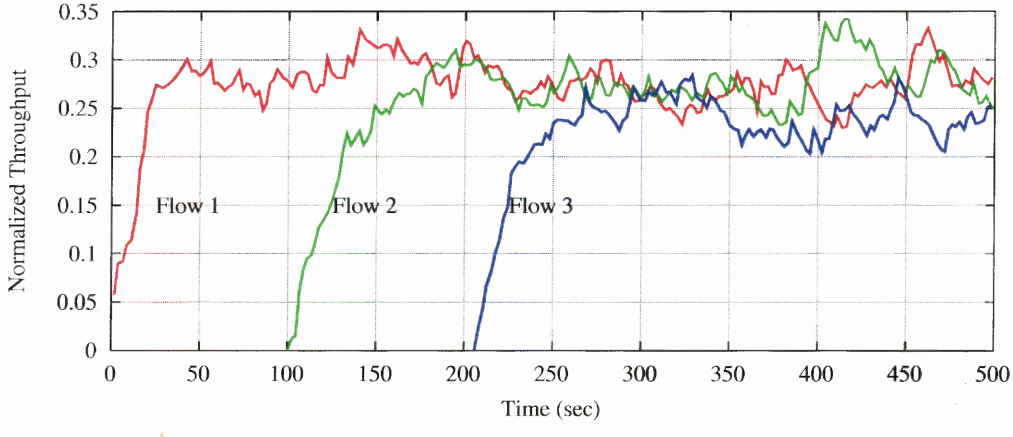


Figure 8.24 Normalized throughputs of three TCP-Jersey flows competing for a 5Mbps wireless link with 5% of random wireless packet loss rate.

Figure 8.25 shows the wireless packet loss rates measured by each flow's TCP-Jersey source when it classifies a detected packet loss as wireles error related. The data points are calculted by the source by dividing the number of detected wireless related packet losses by the total number of packets it has sent so far.

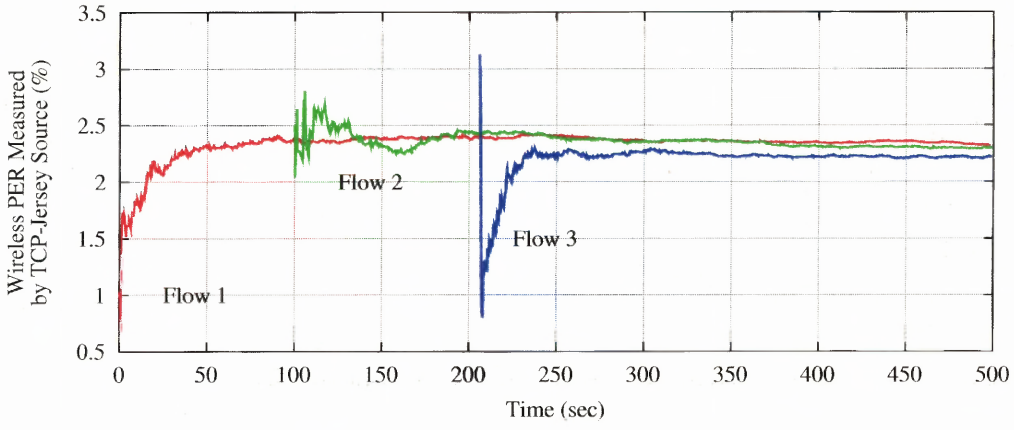


Figure 8.25 Wireless PER measured by sources when 3 TCP-Jersey flows sharing a 5Mbps wireless link with 5% of random packet loss rate.

We summarize that there are three benefits of VCRR packet marking. First, since the input rate to the router is regulated to the virtual capacity, which is slightly smaller than the actual capacity, packet loss due to buffer overflow is eliminated. Therefore, losses due to wireless link errors can be identified with higher accuracy. Second, the real queue will never grow persistently, and thus minimizes the queuing delay and increases the network's stability and responsiveness. Third, heuristic parameter settings in the queue length based threshold marking CW scheme can be eliminated, and thus the scalability and adaptiveness of the resulting system is improved.

8.7 How Close Is TCP-Jersey to a "Perfect" TCP?

According to simulations and analysis presented in preceding sections, we prefer VCRR as the AQM link algorithm component for the end-to-end solution to TCP performance enhancement in wireless networks. Therefore, from this point on, when it is referred to TCP-Jersey, it means a solution that consists of the AIAD congestion control at the source using TS-ARE as the achievable rate estimator, and the VCRR congestion warning packet marking strategy at the link. Appendix B collects the overall implementation of TCP-Jersey as a set of psuedo-codes.

In this section, we investigate what a "perfect" TCP can do in terms of throughput when it is used in a lossy wireles network, and how close TCP-Jersey performs as compared to the "TCP-Perfect".

First we define the "TCP-Perfect" as a modification of the most widely adopted stock TCP protocol with *perfect knowledge* of packet losses caused by wireless link errors, and its congestion window is not decreased when such lost packets are retransmitted.

To construct the perfect TCP, we modified the NS-2 simulator in such a way that the simulator's error process associated with each wireless link reports immediately the sequence number of every corrupted, and thus lost packet to the TCP source through a special type of packet. We pick the TCP NewReno implementation in NS-2 as the stock TCP and modified its source algorithm to recognize the special reporting packet and record the sequence numbers of the corrupted and lost packets in a table for later reference. Therefore, the source has the perfect knowledge as to which packet has been lost due to wireless link errors. When a packet loss is detected by the source receiving 3 duplicate ACKs, whose sequence number indicates the sequence number of the lost packet, the source queries the table for the same sequence number to determine if the loss was due to wireless link errors. The source responds to a wireless related loss by retransmitting

the lost packet and keeping the congestion window intact. Otherwise, the source maintains the normal window adjustment implemented in TCP NewReno, i.e., the AIMD algorithm.

We note that such a perfect TCP scheme is not realistic for several reasons. First, it is not scalable. Every link needs to implement such a special reporting function that can send a report at the TCP layer backwards to the source. Second, the reports will not be realiable in reality. In the simulation environment, we can program the error process, and the error process knows which packet it has just corrupted, since that is exactly it is designed to simulate. While in reality, the link's receiver must be able to detect a corrupted packet and at the same time still be able to extract the TCP sequence number, which is part of the data payload that has been corrupted. Third, the correct delivery of the reporting packets is not guaranteed. Fourth, the TCP source receiving the reports may have to store a fair amount of reported sequence numbers for an accurate query.

Nevertheless, the TCP-Perfect as we have just constructed in a simulation environment can provide a reasonable upper bound of how much a perfect knowledge of all wireless link error induced packet losses can help in improving the throughput of a standard TCP whose reaction is modified according to the perfect knowledge.

To compare TCP-Jersey's performance to that of a perfect TCP, we repeated the simulation that was reported in Figure 8.11 and Figure 8.16 in the simulation network of Figure 8.10, and the results are plotted in Figure 8.26. As shown in the figure, TCP-Jersey's performance approaches to that of the perfect TCP closely up to 3% of random packet loss rate on the wireless link.

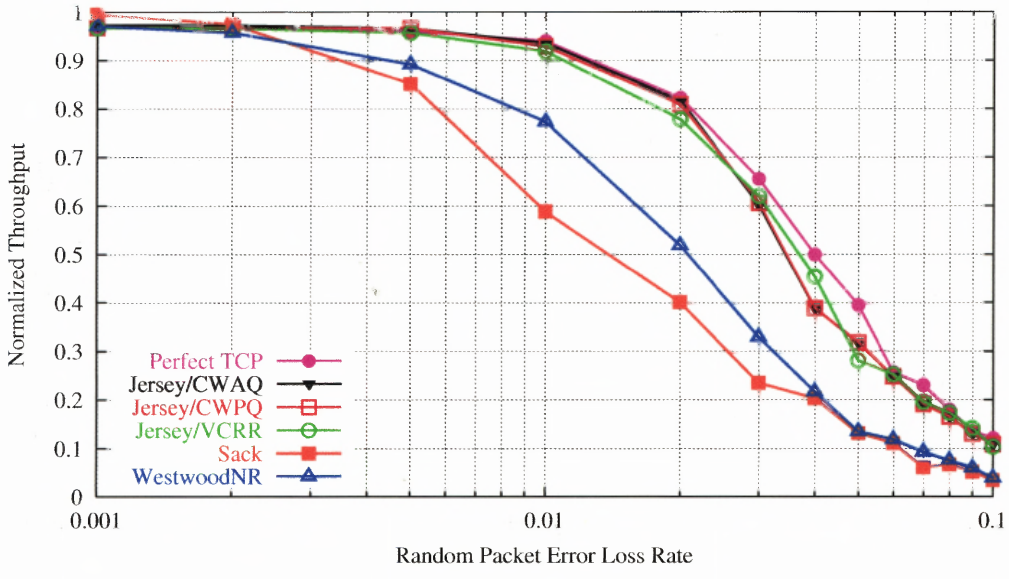


Figure 8.26 Normalized throughput of TCP-Jersey vs. TCP-Perfect and others. The curve representing TCP-Jersey/VCRR is scaled up by the target utilization factor of 0.95%.

8.8 Summary

In this chapter, we have investigated the role the active queue management (AQM) plays in the context of TCP congestion control. We particularly established a basic network model that can be used to study TCP/AQM dynamics. Specifically, we used the model to study the equilibrium properties of the standard TCP Reno and the proposed TCP-Jersey with a focus of the potential performance improvement that a loss differentiator could provide.

In Sections 8.5 and 8.6, we proposed two improved congestion warning (CW) marking schemes, namely, CWPQ and VCRR. CWPQ uses the persistent queue length as the measurement of link congestion level instead of the average queue length as used in the original TCP-Jersey (referred to as CWAQ). CWPQ is very easy to implement and entails much less computational demand than CWAQ. We conducted simulations to compare the performance of the two marking schemes and concluded that both CWAQ and CWPQ performs indistinguishably close to each other and superior to any other TCP schemes in a lossy wireless network. CWPQ eliminates the heuristical setting of the weight parameter in the computation of the average queue length.

VCRR, or virtual capacity rate regulation AQM, is yet another improvement to CWPQ. It addresses the problem that both CWAQ and CWPQ exhibit, that is, queue oscillation. VCRR uses the concept of virtual capacity, which is slightly less than the actual link capacity, and provides a continuous marking function in order to match the aggragate input rate to the virtual capacity. Simulations in this chapter show that the VCRR link algorithm and TCP-Jersey source algorithm perform well in error-prone wireless networks, and at the same time result in much less oscillatory queues, whish is a very desirable proporty for delay and jitter sensitive applications.

In the last section of this chapter, we investigated the performance of a "perfect" TCP which has perfect knowledge of wireless error induced packet losses.

We compare the proposed TCP-Jersey with that of the perfect TCP and showed that TCP-Jersey's performance has approached very close to the upper bound represented by that of the perfect TCP.

CHAPTER 9

GENERALIZATION OF ARE-BASED AIAD CONGESTION CONTROL FOR RATE CONTROL APPLICATIONS

The standard TCP and the proposed TCP-Jersey source algorithms are categorized as the window-based congestion control algorithms, where the source transmits packets according to an elastic window, namely, congestion window, or *cwnd*, that limits the number of outstanding (or, unacknowledged) packets allowed in the network. Because of the acknowledgment mechanism and the additive increase of *cwnd* upon the returning ACKs, the congestion window also determines, by approximation, the source's transmission rate. Therefore, the window-based congestion control such as TCP controls the source's transmission rate *indirectly* through the control of *cwnd* window size. Window-based transmission control tends to produce bursty traffic because, as in TCP, when *cwnd* is increased to the next integral level, back-to-back packets are transmitted.

Rate-based transmission controls, such as TCP-friendly rate control (TFRC) [112], rate adaption protocol (RAP) [113], rate-based AIMD [114], and the rate control framework by Kelly [98], control the source's transmission rate *directly* by adjusting the source rate. Rate-based congestion control has seen many potential applications in real-time streaming where delay sensitive contents are to be delivered. Many of the real-time streaming applications, including voice over IP (VoIP), use UDP as their transport protocol instead of TCP. Congestion controls are done at the application layer.

With the ever-growing demand of real-time multimedia streaming, particularly in wireless or wired/wireless hybrid networks, congestion control and packet loss differentiation remains to be challenges for any rate-based protocols. This

chapter extends the ARE-based AIAD congestion control of the proposed TCP-Jersey to potential applications or implementations of a rate-based transmission control protocol for real-time multimedia content delivery. Specifically, this chapter provides an analytic model of a rate-based congestion control that captures the essence of TCP-Jersey's source algorithm in a discrete-time form. The model developed in this chapter serves a starting point or a stepping stone toward future extension of TCP-Jersey, or more precisely, the design framework of TCP-Jersey, for real-time applications that need an efficient congestion control as well as an effective loss differentiation in wireless networks.

9.1 From Window-based Control to Rate-based Control

Based on the basic network model developed in the previous chapter, the window-based congestion control of TCP-Jersey is expressed as (i.e., (8.19))

$$\dot{w}_i(t) = x_i(t-\tau_i)\big(1-q_i(t)\big)\frac{1}{w_i(t)} - x_i(t-\tau_i)q_i(t)\Big[w_i(t) - r_i(t)d_i\Big], \tag{9.1}$$

where $r_i(t)$ is the achievable rate estimation by the source using the TS-ARE algorithm. In the above, the time-varying delay of $\tau_i(t)$ is replaced by its steady-state value τ_i for the tractability of the analysis.

At the steady-state, the congestion window $w_i(t)$ and the source's transmission rate has the following approximated relationship

$$x_i(t) = \frac{w_i(t)}{\tau_i}. \tag{9.2}$$

Two simplifications are applied to (9.1). First, since the source's congestion window increases roughly by 1 for every RTT [115, 62], we have

$$\frac{w_i(t-\tau_i)}{w_i(t)} \approx \frac{w_i(t)-1}{w_i(t)} = 1 - \frac{1}{w_i(t)} \approx 1, \tag{9.3}$$

where the last approximation is based on the reasonable assumption that in the vicinity of the steady-state, TCP would have $w_i(t) \gg 1$. Second, as with the New Reno TCP, TCP-Jersey only reduces its congestion window no more than once for a RTT, the second term on the right-hand side of (9.1) can be reduced to $\frac{q_i(t)[w_i(t)-r_i(t)d_i]}{\tau_i}$ [103, 90].

By applying the above to (9.1), the window-based control can be converted to a rate-based control as

$$\dot{x}_i(t) = \frac{1 - q_i(t)}{{\tau_i}^2} - \frac{q_i(t)\Big[x_i(t) - \eta_i r_i(t)\Big]}{\tau_i}, \tag{9.4}$$

where $\eta_i = d_i/\tau_i$ is the ratio of the round-trip propagation delay and the steady-state round-trip delay of flow i.

9.2 Generalized Additive Increase and ARE-based Adaptive Decrease Rate Control

The rate-based control of (9.4) which is derived from the additive increase and ARE-based adaptive decrease congestion control of the TCP-Jersey's source algorithm can be generalized as follows.

The additive increase part, i.e., the first term of the right-hand side in (9.4) is multiplied by a control parameter $\hat{\alpha}_i \geq 1$. This is in the same spirit of the HighSpeed TCP (HSTCP) proposed in [116] that aims to increase the speed of convergence of TCP in high-speed networks., since the increase by 1 for each RTT rule results in low link utilization in high-speed networks. The $x_i(t) - \eta_i r_i(t)$ term in (9.4) that controls the magnitude of the adaptive decrease can be naturally generalized to $\hat{\beta}_i[x_i(t) - r_i(t)]$ with a second control parameter $0 < \hat{\beta}_i \leq 1$, which represents a proportion of the mismatch between the source's current sending rate and the estimated achievable rate.

Therefore, a generalized additive increase and ARE-based adaptive decrease congestion control, hence named ARE-based GAIAD, can be expressed as

$$\dot{x}_i(t) = \frac{\hat{\alpha}_i}{{\tau_i}^2}\big(1 - q_i(t)\big) - \frac{\hat{\beta}_i}{\tau_i} q_i(t)\Big[x_i(t) - r_i(t)\Big]. \tag{9.5}$$

9.3 Discrete Realization of ARE-based GAIAD Rate Control

Unlike the window-based control in TCP, where the window size is updated on every ACK, in rate-based control such as (9.5), the source needs to compute and adjust its transmission rate according to the congestion indication $q_i(t)$; therefore, the rate control can be invoked periodically, i.e, once for every δ_i time units. A natural choice of the control interval δ_i is the average RTT measured by the source [113], i.e., $\delta_i \sim \tau_i$. Thus, discretization of (9.5) with control interval $\delta_i = \tau_i$ becomes

$$x_i(n) = x_i(n-1) + \frac{\hat{\alpha}_i}{\tau_i}\big(1 - q_i(n)\big) - \hat{\beta}_i q_i(n)\Big[x_i(n) - r_i(n)\Big]. \tag{9.6}$$

The source rate control as shown in (9.6) is biased since the change of the rate is roughly inverse proportional to the RTT, making the control to exhibit the similar RTT-bias seen in the standard window-based TCP congestion control schemes, where a flow traveling longer distance gets smaller share of the bottleneck bandwidth than a flow close to the bottleneck. To avoid such unfairness, one would fix the control parameters of all flows and establish a uniform set of equations that govern the system. This can be done by a strict protocol specification and request of conformance to the specification in implementations. Thus, a new notion of the control parameters are specified as $\alpha = \hat{\alpha}_i/\tau_i$, and $\beta = \hat{\beta}_i$, with restrictions of $\alpha \geq 1$ and $0 < \beta \leq 1$. Therefore, the discretized generalized model for a rate-based AIAD

congestion control can be expressed as

$$\begin{aligned} x_i(n) &= x_i(n-1) + \alpha\big(1 - q_i(n)\big) - \beta q_i(n)\Big[x_i(n) - r_i(n)\Big] \\ &= \frac{1}{1+\beta q_i(n)}\Big[x_i(n-1) + \alpha\big(1 - q_i(n)\big) - \beta q_i(n) r_i(n)\Big] \end{aligned} \tag{9.7}$$

Recall, in the previous chapter, the TS-ARE is computed based on the time averaging of the instantaneous effective throughput that the source observes, i.e., Equation (8.20). In a discrete implementation, the time averaging can be folded into the computation of the delayed transmission rate $x_i(t - \tau_i)$, and therefore, at each discrete step n, the achievable rate estimation $r_i(n)$ can be expressed as

$$r_i(n) = x_i(n - \tau_i)\big(1 - q_i(n)\big). \tag{9.8}$$

The same can also be derived from (8.20) by simple discretization. Also, recall the aggregate congestion indication $q_i(t)$ that source i receives at time t is a function of the delayed aggregate input rates on the links of the route flow i travels, i.e., Equations (8.13) and (8.14), which can be described in discrete-time domain as

$$q_i(n) = \sum_{l \in \boldsymbol{L_i}} R_{li} p_l(n - \tau_{li}^b), \tag{9.9}$$

$$p_l(n) = p\big(y_l(n)\big), \tag{9.10}$$

and

$$y_l(n) = \sum_{i \in \boldsymbol{I}} R_{li} x_i(n - \tau_{li}^f), \tag{9.11}$$

where τ_{li}^f and τ_{li}^b are the steady-state values of $\tau_{li}^f(t)$ and $\tau_{li}^b(t)$ defined in Section 8.2, respectively; and $\tau_i = \tau_{li}^f + \tau_{li}^b$. Therefore, in (9.7), source rate $x_i(n)$ is adapted based on the most recent rate $x_i(n-1)$ and feedbacks carrying information of $x_i(n - \tau_i)$,

which was in effect RTT time units earlier. The system in (9.6) quickly becomes unstable as the discrepancy between $x_i(n-1)$ and $x_i(n-\tau_i)$ increases. A natural remedy to this problem is to retard the reference rate to be $x_i(n-\tau_i)$ instead of $x_i(n-1)$ so that the feedbacks reflect the actual network conditions with regard to the reference rate. Therefore (9.7) becomes

$$x_i(n) = \frac{1}{1+\beta q_i(n)}\Big[x_i(n-\tau_i) + \alpha\big(1-q_i(n)\big) - \beta q_i(n) r_i(n)\Big]. \tag{9.12}$$

Next, studies have shown that cumulative congestion indication such as (9.9) does not lead to the Max-Min fairness in the network. Instead, congestion indication feedback from the *most-congested* link is preferred [117,79,80]. Feedback from the most-congested link is difficult to realize in the loss-based congestion indication by non-AQM controlled links. With bits in the packet header designated for congestion indication, such as ECN, congestion notification from the most-congested link become possible, e.g., [118]. In such a scheme, instead of (9.9), the congestion indication feedback that the source receives becomes

$$q_i(n) = \max_{l \in \boldsymbol{L_i}} p_l(n-\tau_{li}^b). \tag{9.13}$$

9.4 Stability Analysis

In this section, stability conditions of the discrete dynamical system (9.12)-(9.13) are investigated. Flows can be partitioned into non-overlapping sets based on their bottleneck links. Flows with a common bottleneck link belong to the same set. Here, bottleneck means the most-congested link along a flow's path. Then, system (9.12)-(9.13) for a given set $S = \{x_i : i = 1, 2, \ldots, N\}$ can be simplified by dropping the index of links and expanding $q_i(n)$ in (9.13) and $r_i(n)$ in (9.8) as

$$\begin{aligned} x_i(n) &= \frac{1}{1+\beta p(n-\tau_i^b)}\Big[x_i(n-\tau_i) + \alpha\big(1-p(n-\tau_i^b)\big) - \beta p(n-\tau_i^b) r_i(n)\Big] \\ &= \frac{1-\beta p(n-\tau_i^b)\big(1-p(n-\tau_i^b)\big)}{1+\beta p(n-\tau_i^b)} x_i(n-\tau_i) + \frac{1-p(n-\tau_i^b)}{1+\beta p(n-\tau_i^b)}\alpha, \end{aligned} \tag{9.14}$$

and

$$p(n) = p\big(\sum_{u \in S} x_u(n - \tau_u^f)\big), \tag{9.15}$$

with $\tau_i = \tau_u^f + \tau_i^b$. We assume $p(n)$ is differentiable at the stationary point and has the same first-order partial derivative for all flows, since the link algorithm is assumed to be oblivious.

For systems described by (9.14)-(9.15), the following Theorem 9.1 due to Zhang et al. in [119] can be invoked to study the delay-independent conditions under which the system is locally asymptotically stable.

Theorem 9.1 (Zhang et al. [119]) *Assume an undelayed linear system $\mathcal{L}$ with N flows:*

$$x_i(n) = \sum_{j=1}^{N} a_{ij} x_j(n-1).$$

If the matrix $A = (a_{ij})$ is real-valued and symmetric, then system $\mathcal{L}_D$ with arbitrary directional delays:

$$x_i(n) = \sum_{j=1}^{N} a_{ij} x_j(n - \tau_i^b - \tau_j^f)$$

is asymptotically stable if and only if $\mathcal{L}$ is stable.

In [119], Zhang et al. also showed that the preceding theorem also applies to non-linear systems as stated in the following Corollary 9.2

Corollary 9.2 *Assume an undelayed N-dimensional non-linear system $\mathcal{N}$:*

$$x_i(n) = f_i\big(x_1(n-1), x_2(n-1), \ldots, x_N(n-1)\big),$$

where $\{f_i | f_i : \mathbb{R}^N \to \mathbb{R}\}$ is the set of nonlinear functions defining the system. If the Jacobian matrix $\mathcal{J}$ of the system is symmetric and real-valued, system $\mathcal{N}_D$ with

arbitrary delay:

$$x_i(n) = f_i\big(x_1(n-\tau_i^f-\tau_i^b), x_2(n-\tau_2^f-\tau_i^b), \ldots, x_N(n-\tau_N^f-\tau_i^b)\big),$$

is locally asymptotically stable in the stationary point $\mathbf{x}^*$ *if and only if* $\mathcal{N}$ *is stable in* $\mathbf{x}^*$

To invoke Theorem 9.1, we first study the stability conditions of the undelayed version of system (9.14)-(9.15):

$$\begin{aligned} x_i(n) &= \frac{1-\beta p(n-1)\big(1-p(n-1)\big)}{1+\beta p(n-1)} x_i(n-1) + \frac{1-p(n-1)}{1+\beta p(n-1)}\alpha, \\ p(n) &= p\big(\sum_{u\in S} x_u(n)\big). \end{aligned} \tag{9.16}$$

Since all flows receive the same feedback and activate the same response to the feedback, all flows share the bottleneck link fairly in the steady-state, i.e., $x_i(n) = x^*$ for all i in the steady-state. Simple manipulation of (9.16) gives

$$x^* = \frac{\alpha(1-p^*)}{\beta p^*(2-p^*)}. \tag{9.17}$$

Linearizing system (9.16) in $\mathbf{x}^*$ yields:

$$\frac{\partial f_i}{\partial x_i}|_{\mathbf{x}^*} = \big(\frac{1-\beta p(1-p)}{1+\beta p} + x_i\frac{\partial g}{\partial x_i} + \alpha\frac{\partial v}{\partial x_i}\big)|_{\mathbf{x}^*}, \tag{9.18}$$

$$\frac{\partial f_i}{\partial x_k}|_{\mathbf{x}^*} = \big(x_i\frac{\partial g}{\partial x_k} + \alpha\frac{\partial v}{\partial x_k}\big)|_{\mathbf{x}^*}, k \neq i, \tag{9.19}$$

where

$$\begin{aligned} g &= \frac{1-\beta p(n-1)\big(1-p(n-1)\big)}{1+\beta p(n-1)}, \\ v &= \frac{1-p(n-1)}{1+\beta p(n-1)}, \\ f_i &= g x_i(n-1) + v\alpha. \end{aligned}$$

Since the congestion indication function $p(n)$ depends on the aggregate flow rate on the bottleneck, $p(n)$ has the same first-order partial derivative evaluated at point $\mathbf{x}^*$ for all flows. This implies that for any flows i and k, we have

$$\begin{aligned}
\frac{\partial p}{\partial x_i}|_{\mathbf{x}^*} &= \frac{\partial p}{\partial x_k}|_{\mathbf{x}^*}, \\
\frac{\partial g}{\partial x_i}|_{\mathbf{x}^*} &= \frac{\partial g}{\partial x_k}|_{\mathbf{x}^*}, \\
\frac{\partial v}{\partial x_i}|_{\mathbf{x}^*} &= \frac{\partial v}{\partial x_k}|_{\mathbf{x}^*}.
\end{aligned}$$

Therefore, the Jacobian matrix $\mathcal{J}$ for system (9.16) is in the circulant form as

$$\mathcal{J} = \begin{pmatrix} a & b & \dots & b \\ b & a & \dots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \dots & a \end{pmatrix}, \tag{9.20}$$

where

$$a = \frac{1-\beta(p^*-p^{*2})}{1+\beta p^*} + \left[x^* - \frac{x^*(1-2\beta)-\alpha(1+\beta)}{(1+\beta p^*)^2}\right]\frac{\partial p}{\partial x_i}|_{\mathbf{x}^*}, \tag{9.21}$$

$$b = \left[x^* - \frac{x^*(1-2\beta)-\alpha(1+\beta)}{(1+\beta p^*)^2}\right]\frac{\partial p}{\partial x_i}|_{\mathbf{x}^*}. \tag{9.22}$$

The k-th eigenvalue λ_k of a circulant matrix $\mathcal{J}$ is given by [120]:

$$\lambda_k = a + b\big(\zeta_k + \zeta_k^2 + \zeta_k^3 + \ldots + \zeta_k^{N-1}\big), \tag{9.23}$$

where $\zeta_k = e^{2\pi jk/N}$ $(k = 0, 1, \ldots, N-1)$ is one of the N-th roots of unity. For the case where $N \geq 2$, it is easy to derive that

$$\lambda_k = \begin{cases} a + (N-1)b & \zeta_k = 1, \\ a + b\frac{\zeta_k - \zeta_k^N}{1-\zeta_k} = a - b & \zeta_k \neq 1 \end{cases}, \tag{9.24}$$

since $\zeta_k^N = 1$ for all k.

Since the nonlinear system (9.16) is locally stable if and only if all eigenvalues of its Jacobian matrix $\mathcal{J}$ are within the unit circle [121], the necessary and sufficient conditions for system (9.16) to be locally stable are

$$|a + (N-1)b| < 1 \tag{9.25}$$

$$|a - b| < 1 \tag{9.26}$$

In many implementations of congestion indication functions, p is usually normalized in the range of $[0, 1]$. In such a case, the above conditions translate directly to

$$-1 < \frac{1 - \beta(p^* - p^{*2})}{1 + \beta p^*} + N\left[x^* - \frac{x^*(1-2\beta) - \alpha(1+\beta)}{(1+\beta p^*)^2}\right]\frac{\partial p}{\partial x_i}|_{\mathbf{x}^*} < 1. \tag{9.27}$$

Thus, the undelayed nonlinear system (9.16) is locally stable if and only if (9.27) is satisfied.

According to Theorem 9.1, Jacobian $\mathcal{J}$ of the undelayed system (9.16) is real-valued and symmetric; then according to Corollary 9.2, the following result is obtained.

Theorem 9.3 *The heterogeneously delayed system of the ARE-based generalized AIAD rate control system as described by* (9.14)-(9.15) *is locally asymptotically stable if and only if* (9.27) *is satisfied.*

■

9.5 Summary

Many multimedia applications do not use TCP as the transport protocol, rather, they build the transmission control at the application layer using UDP. In this chapter, the ARE-based AIAD congestion control of the proposed TCP-Jersey is extended and generalized for applications that require direct control of the source's transmission rate instead of the TCP-like window-based control. The generalized system is developed as a discrete-time nonlinear dynamical model. The local stability condition of the generalized system is developed, which can be used in future studies in guiding the choice of control parameters.

CHAPTER 10

CONCLUSIONS

In this dissertation, we have analyzed and illustrated through simulations, in Chapter 3.1, the performance degradation problems exhibited when the unmodified standard TCP protocol is used in data transfer through wireless IP networks. The problem stems from the TCP's inability of differentiating packet losses due to network congestion from those due to wireless link errors, and TCP treats all packet as congestion loss, thus resulting in unnecessary reduction of the transmission rate.

We have reviewed and analyzed by categorizing the current state-of-the-art solutions to the same problem in Section 3.2. We are motivated to provide an end-to-end solution to enhance TCP's performance in wireless IP communications. By an end-to-end solution, we mean a solution that does not rely on either a "wireless-aware" or "wireless-specific" IP router or switch, or on the functions implemented at layers below the IP. The solution proposed by this research, namely TCP-Jersey, is unique such that it combines the achievable rate estimation (ARE) based intelligent congestion control, an additive-increase and adaptive-decrease (AIAD) congestion control algorithm, and the packet marking capability of a simple active queue management (AQM) scheme, the congestion warning (CW) AQM, to combat the two key problems.

The first problem is the rather heuristic rate reduction of the standard TCP protocol, which is not pertaining to wireless IP communications only, but manifests its adverse effect when TCP is not able to distinguish congestion packet losses from random wireless packet losses. Using the sender's estimation of its achievable rate as a guide to the window reduction procedure certainly alleviates, at some level, the severe throughput degradation caused by the high BER of the wireless links,

since, in this case, even though all packet losses are classified as congestion losses, the ARE ensures that the sender's window would not be reduced too far below its achievable rate.

The second key problem is differentiating the packet loss type. We approached to the problem differently from other researches. Instead of devising a split mode connection to shield the wireless links from the TCP connection by a specially designed or augmented router, or relying on the link layer notification of wireless transmission errors to the upper TCP layer, which complicates the protocol design and implementation, we proposed a simple, yet effective, AQM scheme that marks the packets when there is an incipient sign of congestion. The proposed TCP-Jersey source algorithm examines the congestion marks of the DUPACK packets to differentiate the packet loss type. Therefore, the traditional benefits of AQM in congestion control in general has been preserved and the congestion indication (i.e., marks in packet headers) is used for both congestion control and loss differentiation. Therefore, the proposed solution is unique in the application of AQM and is completely compatible to existing technologies. This makes our solution generally applicable to both wired and wireless networks, and most importantly applicable to the wired/wireless hybrid networks.

In Chapter 5, we presented the proposed end-to-end solution to TCP performance enhancement in wireless communications through an integrated implementation named TCP-Jersey. Results from extensive simulations presented in the chapter confirms that the solution is in the right direction and encourages us for further research along this line. In Chapter 6, we generalized the ARE algorithm in a TCP dynamic model and confirmed, through extensive simulations, that the original ARE algorithm can be further improved to become much more robust when the reverse end-to-end path is also experiencing congestion, a reality usually ignored in many theoretical and simulation-based studies.

As a result, an improved achievable rate estimation algorithm, namely, TS-ARE, is proposed in the subsequent Chapter 7. TS-ARE uses the timestamps option available in many standard TCP implementations to construct the estimation of the achievable rate, instead of using the arrival times of the ACK packets. The result is a more robust estimator to ACK compression and ACK losses. The newly improved TS-ARE is then integrated into the TCP-Jersey source algorithm, and further simulations are conducted to confirm its performance. Comparisons are made to the original TCP-Jersey source algorithm in the environment of two-way congestion and correlated error processes.

Chapters 8 provides thorough analysis of the inter-connected TCP source congestion control and active queue management (AQM) link algorithm dynamics in the context of a feedback control system, with a particular interest in improving the TCP-Jersey's link algorithm, i.e., the CW AQM packet marking. In Chapter 8, two CW marking schemes are proposed and extensive simulation results are presented. The first proposal in this chapter is the CWPQ link algorithm. As compared to the original CW marking scheme, which is named as CWAQ, CWPQ uses the estimated persistent queue length as the link's congestion measure instead of computing the average queue length as in CWAQ. The benefit of CWPQ is its extreme simplicity in computation and the elimination of the heuristic parameter setting in CWAQ. Our results show that CWPQ can achieve the same performance as the original CWAQ when combined with the TCP-Jersey's source algorithm.

In an effort to further eliminate the rather oscillatory queue length as the result of threshold marking used in CWAQ and CWPQ, a continuous marking function named VCRR is also proposed and evaluated. It is concluded by experiments in the chapter that VCRR can also achieve the same performance as the previous link algorithms but results a much stable queue.

Many multimedia applications do not use TCP as the transport protocol; rather, they build the transmission control at the application layer using UDP. In Chapter 9, the ARE-based AIAD congestion control of the proposed TCP-Jersey is extended and generalized for applications that require direct control of the source's transmission rate instead of the TCP-like window-based control. The generalized system is developed as a discrete-time nonlinear dynamical model. The local stability condition of the generalized system is developed, which can be used in future studies in guiding the choice of control parameters.

In conclusion, this dissertation has a focused interest in providing an end-to-end solution to the well-known problem that TCP's performance degrades severely in error-prone wireless networks. The body of research work presented in the dissertation includes a comprehensive survey of the currently proposed solutions and our proposed solution framework. In designing the proposed solution, we have followed the philosophy of simplicity and compatibility. As a result, a solution based on an adaptive source algorithm in congestion control and the intelligent use of congestion marks by an AQM link algorithm is proposed and named as TCP-Jersey. Subsequent researches, including analysis and simulations, to improve the original proposal are conducted.

The framework architecture and the actual simulation implementation of the solution can be beneficial not only to further researches in the same direction, but also for researches in reliable and efficient transport protocol designs for future heterogeneous wireless multimedia networks, where congestion and loss recovery will remain to be the major concern for any effective protocol to achieve high performance.

CHAPTER 11

AREAS OF INTEREST FOR FUTURE RESEARCH

TCP-Jersey, a combination of a source initiated estimation of achievable transmission rate, an additive-increase and adaptive-decrease window-based congestion control algorithm, a loss differentiator at the source relying on the single-bit congestion information carried back by the acknowledgment packets, and a simple non-wireless-specific packet marking AQM algorithm at the link provides a simple effective and compatible solution to the performance degradation problem for bulk data transfer in wireless networks. This is the evolutionary path we have taken in solving a practical problem without disturbing the current network and protocol structures, in terms of changes involved in TCP protocol software, link software, and router deployment.

There are limitations as to how far the solution can go as it currently stands. First, the AIAD congestion control at the TCP-Jersey source consists of additive increment of the window size. This slow probing mechanism may become noticeably inefficient in the future high-speed networks. There are researches being conducted in search for solutions with faster convergence speed, such as high-speed TCP (HSTCP) [122, 116] and FAST [123]. Second, the adaptive decrement of window size that is computed from the achievable rate estimation does not take into account many other network conditions, particularly related to bandwidth allocation of the link, service priorities, and QoS parameters. The network has more accurate knowledge of how a flow's transmission rate should be adjusted, and this knowledge needs to be fed back to the source in a more descriptive way. Algorithms such as those proposed in XCP [79] and MaxNet [80] present attractive directions

for future development, although XCP and MaxNet require a complete replacement of the standard TCP protocol and deployment of architecturally different routers.

Loss differentiation in TCP-Jersey is simple, although indirect, but not completely accurate nor in a timely fashion. This may become a problem when packet losses occur on multiple wireless links with different characteristics along the path. This type of wireless IP network is becoming reality with the standards currently being formed by the WiMAX Forum that would integrate heterogeneous wireless access technologies into one inter-connected network.

Solutions to hand-off detection and the appropriate response by TCP source is not pursued by TCP-Jersey. This is certainly an interesting but difficult problem whose solution would bear practical application benefits.

Looking five to ten years into the future, whether TCP as we understand it would remain a dominant data transport protocol in the Internet of that time is an intriguing question. Nevertheless, the need for network congestion control will not diminish, but the mechanism by which it is accomplished may very well be different from today's Internet congestion control, which is basically a sole undertake by the layer-4 protocol in an end-to-end fashion. The loss-based congestion control (i.e., the current TCP) cannot go very far since the transmission rate is tightly coupled with and, in a sense, controlled by the packet loss events. Estimation of the achievable rate as in ARE of TCP-Jersey is in a right direction to a (future) protocol and network architecture that decouples the congestion control from the loss detection and loss recovery completely. To this end, a truly reliable, efficient, and "smart" data transport protocol for the future highly heterogeneous wireless Internet may have to be resulted from researches in cross-layer protocol designs.

APPENDIX A

MATHEMATICAL DERIVATIONS AND JUSTIFICATIONS

In this Appendix, we collect the detailed mathematical derivations and justifications referred to from Chapter 6 and Chapter 8.

A.1 Linear Local Stability of AIAD Congestion Control

This section pertains to the derivations and justifications of Chapter 6

A.1.1 Linearization of AIAD Congestion Control Model

For notational simplicity, we drop the subscript i in the delay differential equation (6.11) describing the AIAD rate adaptation, and rewrite it as follows.

$$f(x, x_\tau) \equiv \dot{x} = \frac{1}{\tau^2} - [x - \eta r]\, x_\tau p, \tag{A.1}$$

where we denote x_τ to be the delayed x, i.e., $x_\tau = x(t-\tau)$, and $p = p(x_\tau)$.

At the equilibrium $x_0 = \frac{C}{N}$, we have $f(x_0, x_0) \equiv 0$. Taking the first-order Taylor expansion of $f(x, x_\tau)$ around the equilibrium x_0, we have

$$\frac{\partial f}{\partial x}|_{x_0} = -\frac{C}{N} p_0 + (\eta \frac{C}{N} p_0) \frac{\partial r}{\partial x}|_{x_0}, \tag{A.2}$$

and

$$\frac{\partial f}{\partial x_\tau}|_{x_0} = -(p_0 + \frac{C}{N} p_0')(\frac{C}{N} - \eta r|_{x_0}), \tag{A.3}$$

where $p_0 \equiv p|_{x_0}$, and $p_0' \equiv \frac{\partial p}{\partial x_\tau}|_{x_0}$, and hence,

$$\begin{aligned}\dot{\delta x}(t) &= \frac{\partial f}{\partial x}|_{x_0}\delta x(t) + \frac{\partial f}{\partial x_\tau}|_{x_0}\delta x_\tau(t) \\ &= \left[-\frac{C}{N}p_0 + (\eta\frac{C}{N}p_0)\frac{\partial r}{\partial x}|_{x_0}\right]\delta x(t) \\ &\quad + \left[-(p_0 + \frac{C}{N}p_0')(\frac{C}{N} - \eta r|_{x_0})\right]\delta x(t-\tau), \end{aligned} \tag{A.4}$$

where $\delta x(t) \equiv x(t) - x_0$. By denoting

$$K_1 = -\frac{C}{N}p_0 + (\eta\frac{C}{N}p_0)\frac{\partial r}{\partial x}|_{x_0}, \tag{A.5}$$

and

$$K_2 = -(p_0 + \frac{C}{N}p_0')(\frac{C}{N} - \eta r|_{x_0}), \tag{A.6}$$

we arrive at (6.12). ■

A.1.2 Local Stability Conditions

Case 1: $r_i(t) = \varphi_i(t)$ In this case, since $\varphi_i(t) = \frac{x_i(t)}{\sum_{i=1}^N x_i(t)}C$, (A.5) becomes

$$K_1 = (\frac{N-1}{N^2}\eta - \frac{1}{N})Cp_0, \tag{A.7}$$

and (A.6) becomes

$$K_2 = (p_0 + \frac{C}{N}p_0')\frac{\eta - 1}{N}C. \tag{A.8}$$

■

To test the stability conditions outlined in Lemma 6.1, we observe that since $K_1 \leq 0$, condition 1) is immediately met. This is because $0 < \eta \leq 1$, and $0 \leq p \leq 1$ is a non-decreasing, non-negative function of the aggregated arrival rate on the bottleneck link, hence, $\frac{\partial p}{\partial x_\tau} \geq 0$.

Condition 2) requires that $K_1 < -K_2$. This is also immediately checked after algebraic simplifications.

Condition 3) requites that

$$-\tau K_2 < \sqrt{a_1^2 + (\tau K_1)^2}, \tag{A.9}$$

where a_1 is a solution of $a = \tau K_1 tan(a)$ for $0 < a < \pi$, and $a_1 = \frac{\pi}{2}$ if $\tau K_1 = 0$. Clearly, no closed form relationship among the parameters can be obtained unless $\tau K_1 = 0$. However, we observe the following. First, (A.7) indicates that unless $p_0 = 0$, K_1 is strictly negative, i.e., $K_1 < 0$. Therefore, for a solution to $a = \tau K_1 tan(a)$ to exist in $(0, \pi)$, we must have $tan(a) < 0$. This implies $a_1 \in (\frac{\pi}{2}, \pi)$. Second, if

$$-\tau K_2 < a_1 \tag{A.10}$$

then we also satisfy the inequality (A.9). We can further relax (A.10) to $-\tau K_2 < \frac{\pi}{2}$, which immediately leads to

$$(1 - \eta)\tau < \frac{\pi N}{2C(p_0 + \frac{C}{N}p_0')}. \tag{A.11}$$

Substituting into the above the relationship $d_q \equiv \tau - d = (1 - \eta)\tau$, we arrive at (6.14). ■

The above approximations can be justified as follows. As the number of sharing flows increases, i.e., N increases, (A.7) becomes

$$\begin{aligned} K_1 &\approx (\eta - 1)\frac{C}{N}p_0 \\ &= (\eta - 1)x_0 p_0. \end{aligned} \tag{A.12}$$

In the above, $x_0 p_0$ is the fraction of transmission rate lost due to congestion at the equilibrium; and it should be very close to zero since at the equilibrium $\sum x_i \to C$ implying no persistent buffer build-up, and hence, no over-flow.

Case 2: $r_i(t) = \kappa_i(t)$ Observe that, according to (6.6), we have

$$\frac{\varphi_i(t)}{\sum_{i=1}^N \varphi_i(t)} = \frac{x_i(t)}{\sum_{i=1}^N x_i(t)}. \tag{A.13}$$

It is thus straightforward to derive from (A.5) and (A.6) that

$$K_1 = (\frac{N-1}{N^2}\alpha\eta - \frac{1}{N})Cp_0, \tag{A.14}$$

and

$$K_2 = \frac{\alpha\eta - 1}{N}C(p_0 + \frac{C}{N}p_0'), \tag{A.15}$$

where $\alpha \equiv \frac{\gamma}{\rho}$. ■

Now, according to condition 1) of Lemma 6.1

$$(\frac{N-1}{N^2}\alpha\eta - \frac{1}{N})Cp_0 < 1. \tag{A.16}$$

This does not always hold depending on the value of $\alpha\eta$, which is affected by the link asymmetry of the end-to-end path.

Condition 2) implies

$$(\frac{N-1}{N^2}\alpha\eta - \frac{1}{N})Cp_0 < -\frac{\alpha\eta - 1}{N}C(p_0 + \frac{C}{N}p_0'). \tag{A.17}$$

Substituting $\eta = \frac{d}{\tau} = \frac{d}{d+d_q}$ into the above inequality it yeilds (6.15). ■

By the same arguments in deriving (6.14) in A.1.2, delay condition of (6.16) can be derived under the Condition 3) of Lemma 6.1. ■

APPENDIX B

TCP-JERSEY IMPLEMENTATION

Implementing TCP-Jersey is fairly simple, and it is best described by the following pseudo-codes. TCP-Jersey contains three modules, namely, the achievable rate estimation (ARE) at the source, the AIAD congestion control with loss differentiation at the source, and the congestion warning (CW) packet marking at the link. The TS-ARE and CW algorithms have been presented in pseudo codes before. In the following, we only list the AIAD congestion control and loss differentiation algorithms. Omitted from the following pseudo codes are the remaining operations implemented in the NewReno version of TCP.

Algorithm 1 Source Response to Loss Detected via DUPACKs

if ECE set in DUPACK **then**

loss may be due to congestion

retransmit lost packet

$cwnd \leftarrow ARE \times D$

else

loss may be due to wireless link error

retransmit lost packet

keep $cwnd$ intact

end if

The ECE bit is explained in the essential operations of ECN in Appendix C. In the above, ARE is the source's estimate of the achievable rate by the TS-ARE algorithm, and D is the source's estimate of the round-trip propagation delay, which is taken to be the minimum RTT it has measured so far. The source also remembers

the state of network congestion by setting the internal state variable *congested* to 1 if an ACK or DUPACK has the ECE bit set, and to 0 otherwise.

Algorithm 2 Source Response to Retransmit Timeout

if $congested == 1$ **then**

timeout may be due to congestion

retransmit lost packet

$cwnd \leftarrow 1$

else

timeout may be due to wireless link error

retransmit lost packet

$cwnd \leftarrow ARE \times D$

end if

Algorithm 3 Source Response to ECE Mark in Normal ACKs

$cwnd \leftarrow ARE \times D$

APPENDIX C

ESSENTIAL OPERATIONS OF ECN

This appendix provides a digest of the essential operations of the Explicit congestion Notification (ECN) proposed in RFC 3168 [87] Most ECN implementations are based on RED, where the average queue length is computed by the link and according to a pair of thresholds and other parameters, the packet marking probability is computed in proportion to the average queue length. Chapter 8 gives the detailed formula for the calculation. We have previously remarked that this is only one of many possible ways of computing the marking probability. TCP-Jersey's link algorithm uses a different one, and there could be a varity of other choices.

In what follows, we summarize the signaling procedure between the TCP source, the ECN enabled link and the TCP receiver. Again, this is also only one of many possible ways of signaling. It is presented here since this is what the RFC proposed and it is also the procedure followed by TCP-Jeresey for compatibility reasons.

Bits 15 and 16 of the IP header, shown in Fig. C.1 are designated for ECN signaling at IP layer. Bit 15 is named ECT, which stands for *ECN Capable Transport*; bit 16 is named CE for *Congestion Experienced*.

Bits labeled as CWR and ECE in the TCP header are designated for ECN signaling at the TCP layer. They are shown in Fig. C.2 CWR stands for Congestion Window Reduced, and ECE stands for ECN Echo.

A TCP flow decides to participate in ECN signaling starts by setting the (ECT, CE) pair to (1, 0) in the packet it sends.

The ECN enabled link takes this pattern of bits as the source's inquiry of its ECN capability, and in turn changes the bits to (0, 1) before it forwards the packet.

The TCP receiver recognizes this pattern as the link's handshake signal and copies the pattern in the ACK packet in addition to setting the (CWR, ECE) pair in the TCP header as (1, 0). When the ACK reaches back to the source, recognizes the patterns of the four bits and understands that both the link and the receiver are capable and willing to participate in ECN signaling. This completes the handshake procedure.

In subsequent the packet it sends, the source sets the ECT bit. When the packet reaches the link, if the link decides that the packet is to be marked, it sets the CE bit indicating it is experiencing congestion.

The TCP receiver sees the CE bit and echoes it back by copying the CE bit to the ECE bit in the corresponding ACK packet. Upon receiving an ACK that carries an ECE bit, the source responds as if a packet loss has occured due to congestion and reduces its congestion window according to its implemented algorithm, after which the source sets the CWR bit in the subsequent packet.

The receiver remembers the state that it has just copied a CE bit to the ECE bit, and sets the ECE bits for all the packets that follow regardless their CE bits' values, until it sees a packet with the CWR bit set that indicates the source has seen its echo and has reduced the window size accordingly.

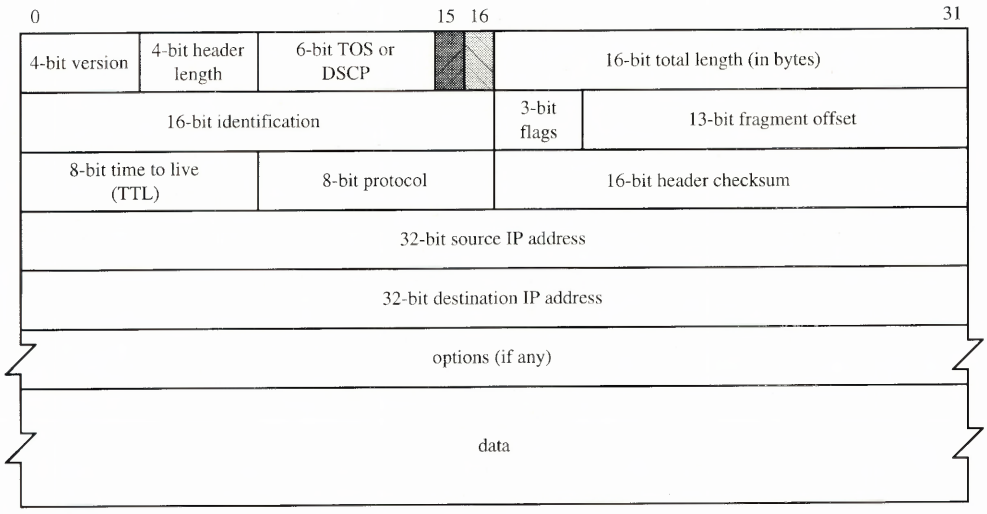


Figure C.1 Structure of IP header showing the two bits assigned for use in ECN.

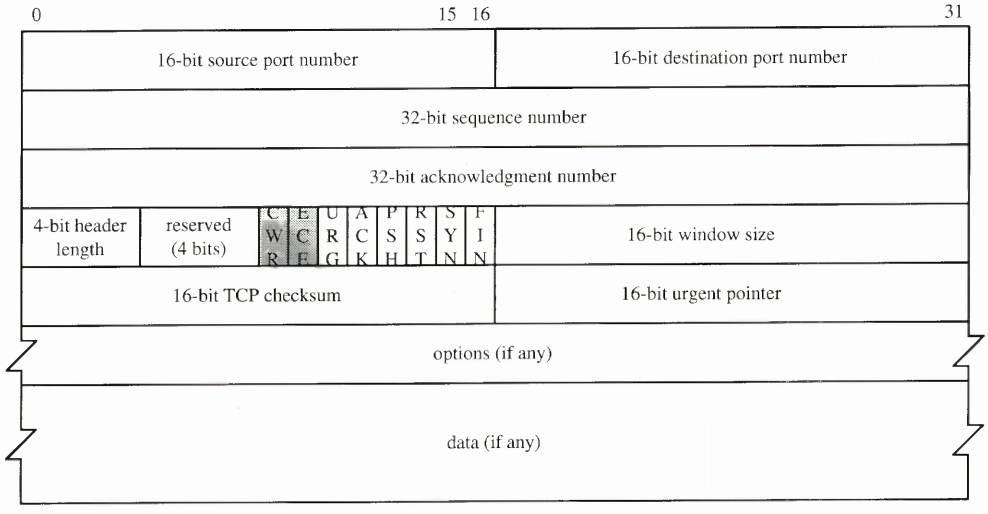


Figure C.2 Structure of TCP header showing the two bits assigned for use in ECN.

REFERENCES

[1] CAIDA, "Traffic Workload Overview (June 1999)," [Available Online] http://www.caida.org/outreach/resources/learn/trafficworkload/tcpudp.xml, June 1999.

[2] S. Floyd, "Bandwidth Used by Different Traffic Types," [Available Online] http://www.icir.org/floyd/ccmeasure.html.

[3] D. D. Clark, C. Partridge, R. T. Braden, B. Davie, S. Floyd, V. Jacobson, D. Katabi, G. Minshall, K. Ramakrishnan, T. Roscoe, I. Stoica, J. Wroclawski, and L. Zhang, "Making the World (of Communications) a Different Place," [Available Online] http://www.ir.bbn.com/ craig/e2e-vision.pdf, March 2005.

[4] V. Tsaoussidis and I. Matta, "Open Issues on TCP for Mobile Computing," *Journal of Wireless Communications and Mobile Computing - Special Issue on Reliable Transport Protocols for Mobile Computing*, vol. 2, no. 1, pp. 3–20, February 2002.

[5] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Link," in *Proc. ACM SIGCOMM 1996*, August 1996, pp. 152–159.

[6] B. P. Crow, J. Kim, I. Widjaja, and P. Sakai, "IEEE 802.11 Wireless Local Area Networks," *IEEE Communications Magazine*, vol. 35, no. 9, pp. 116–126, September 1997.

[7] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP Performance over Wireless Networks at the Link Layer," in *Proc. ACM Mobile Networks and Applications*, vol. 5, no. 1, March 2000, pp. 57–71.

[8] D. Barman and I. Matta, "Effectiveness of Loss Labeling in Improving TCP Performance in Wired/Wireless Networks," in *Proc. of the 10th IEEE International Conference on Network Protocols (ICNP'02)*, 2002.

[9] S. Cen, P. C. Cosman, and G. M. Voelker, "End-to-End Differentiation of Congestion and Wireless Losses," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 703–717, October 2003.

[10] E. H.-K. Wu and M.-Z. Chen, "JTCP:Jitter-Based TCP for Heterogeneous Wireless Networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 4, pp. 757–766, May 2004.

[11] J. Liu, I. Matta, and M. Crovella, "End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment," in *Proc. WiOpt'03: Molding and Optimization in Mobile, Ad Hoc and Wireless Networks*, INRIA Sophia-Antipolis, France, March 2003.

[12] D. Barman and I. Matta, "A Bayesian Approach for TCP to Distinguish Congestion from Wireless Losses," Department of Computer Science, Boston University, Boston, MA 02215, USA, Technical Report BUCS-2003-030, 2003.

[13] ——, "How Well Can TCP Infer Network State?" Department of Computer Science, Boston University, Boston, MA 02215, USA, Technical Report BUCS-2003-011, May 2003.

[14] ——, "Model-Based Loss Inference by TCP over Heterogeneous Networks," in *Proc. WiOpt'04: Molding and Optimization in Mobile, Ad Hoc and Wireless Networks*, University of Cambridge, UK, March 2004.

[15] W. R. Stevens, *TCP/IP Illustrated.* Addison-Wesley, 1994, vol. 1.

[16] J. Postel, "Internet Protocol," IETF RFC 791, 1981.

[17] ——, "Transmission Control Protocol," IETF RFC 793, 1981.

[18] V. Jacobson, "Congestion Avoidance and Control," in *Proc. ACM SIGCOMM 1988*, August 1988, pp. 314–329.

[19] R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," DEC Research, Tech. Rep. TR-301, 1984.

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

[21] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, April 2000.

[22] J. Mahdavi and S. Floyd, "TCP-Friendly Unicast Rate-Based Flow Control," notes sent to end2end-interest mailing list, January 1997.

[23] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet ," *IEEE/ACM Transactions on Networking*, vol. 29, no. 3, pp. 458–472, August 1999.

[24] T. V. Lakshman and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss," *IEEE/ACM Transactions on Networking*, vol. 5, no. 3, pp. 336–350, June 1997.

[25] "NS-2 Network Simulator," [Available Online] http://www.isi.edu/nsnam/ns.

[26] T. S. Rappaport, *Wireless Communications Principles and Practices*, 2nd ed. Upper Saddle River, NJ 07458: Prentice Hall PTR, 2002.

[27] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," in *Proc. of ICDCS'95*, May 1995, pp. 136–143.

[28] K. Brown and S. Singh, "M-TCP: TCP for Mobile Cellular Networks," *ACM Computer Communications Review*, vol. 27, no. 5, pp. 19–43, 1997.

[29] K. Wang and S. K. Tripathi, "Mobile-End Transport Protocol: An Alternative to TCP/IP over Wireless Links," in *Proc. IEEE INFOCOM 1998*, vol. 3, March 1998, pp. 1046–1053.

[30] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," *ACM/Baltzer Wireless Networks Journal*, vol. 1, no. 4, pp. 469–481, December 1995.

[31] S. Keshav and S. Morgan, "SMART Retransmission: Performance with Overload and Random Losses," in *Proc. IEEE INFOCOM 1997*, vol. 3, 1997, pp. 1131–1138.

[32] K. Ratnam and I. Matta, "WTCP: An Efficient Mechanism for Improving TCP Performance over Wireless Links," in *Proc. Int'l Symposium of Computers and Communications*, 1998, pp. 74–78.

[33] H. Balakrishnan and R. H. Katz, "Explicit Loss Notification and Wireless Web Performance," in *Proc. IEEE GLOBECOM Internet Mini-Conf.*, Sydney, Australia, November 1998.

[34] I. F. Akyildiz, G. Morabito, and S. Palazzo, "TCP-Peach: A New Congestion Control Scheme for Satellite IP Networks," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 307–321, June 2001.

[35] I. F. Akyildiz, X. Zhang, and J. Fang, "TCP-Peach+: Enhancement of TCP-Peach for Satellite IP Networks," *IEEE Communications Letters*, vol. 6, pp. 303–305, July 2002.

[36] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," in *ACM MobiCom 2001*, July 2001, pp. 287–297.

[37] J. Liu and S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 7, pp. 1300–1315, 2001.

[38] T. Goff, J. Moronski, D. S. Phatak, and V. Gupta, "Freeze-TCP: A True End-to-End Enhancement Mechanism for Mobile Environments," in *Proc. IEEE INFOCOM 2000*, vol. 3, Tel Aviv, Israel, March 2000, pp. 1537–1545.

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

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

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

[42] C. P. Fu and S. C. Liew, "TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 2, pp. 216–228, February 2004.

[43] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. Prentice Hall, 1992.

[44] S. Biaz and N. H. Vaidya, "Distinguishing Congestion Losses from Wireless Transmission Losses: A Negative Result," in *Proc. IEEE 7th Int'l Conf. on Computer communications and Networks*, Lafayette, LA, October 1998, pp. 390–397.

[45] R. Jain, "A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks," *ACM Computer Communications Review*, vol. 19, pp. 56–71, 1989.

[46] Z. Wang and J. Crowcroft, "A New Congestion Control Scheme: Slow Start and Search (Tri-S)," *ACM Computer Communication Review*, vol. 21, pp. 32–43, January 1991.

[47] N. Samaraweera, "Non-congestion Packet Loss Detection for TCP Error Recovery Using Wireless Links," *IEEE Proceedings of Communications*, August 1999.

[48] T. Kim, S. Lu, and V. Bhargavan, "Improving Congestion Control Performance Through Loss Differentiation," in *Proc. IEEE 8th Int'l Conf. on Computer Communications and Networks*, Lafayette, LA, October 1999.

[49] C. Parsa and J. Garcia-Luna-Aceves, "Differentiating Congestion vs. Random Loss: A Method for Improving TCP Performance over Wireless Links," in *Proc. IEEE WCNC'2000*, 2000, pp. 90–93.

[50] S. Cen, P. C. Cosman, and G. M. Voelker, "End-to-end Differentiation of Congestion and Wireless Losses," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 703–717, October 2003.

[51] S. Biaz, "Heterogeneous data networks: Congestion or corruption?" Ph.D Dissertation, Texas A&M University, College Station, August 1999.

[52] Y. Tobe, Y. Tamura, A. Molano, S. Ghosh, and H. Tokuda, "Achieving Moderate fairness for UDP Flows by Path-Status Classification," in *Proc. 25th Annual IEEE Conf. on Local Computer Networks (LCN 2000)*, Tampa, FL, November 2000, pp. 252–261.

[53] S. Biaz and N. H. Vaidya, "De-randomizing Congestion Losses to Improve TCP Performance over Wired-Wireless Networks," *to appear in IEEE/ACM Transactions on Networking*, 2005, [Available Online: http://www.eng.auburn.edu/users/sbiaz/ton2005.pdf].

[54] S. Floyd and V. Jacobson, "Random Early Detection Gateways for congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.

[55] S. Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communications Review*, vol. 24, pp. 10–23, October 1994.

[56] T. J. Ott, "ECN Protocols and the TCP Paradigm," June 1999, Unpublished, [Available Online] http://web.njit.edu/~ott/Papers/ECN/ECN.pdf.

[57] L. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of IEEE*, vol. 77, no. 2, pp. 257–286, October 1989.

[58] C. Dovrolis, P. Ramanathan, and D. Moore, "What Do Packet Dispersion Techniques Measure?" in *Proc. IEEE INFOCOM 2001*, 2001, pp. 9905–914.

[59] C.-T. Chen, *Analog and Digital Control System Design: Transfer-Function, State-Space, and Algebraic Methods*. Saunders College Publishing, 1993.

[60] D. D. Clark and W. Fang, "Explicit Allocation of Best Effort Packet Delivery Service," *IEEE/ACM Transactions on Networking*, vol. 6, pp. 362–373, August 1998.

[61] W. Fang, N. Seddign, and B. Nandy, "A Time Sliding Window Three Color Marker (TSWTCM)," IETF RFC2859, June 2000.

[62] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "A Control Theoretic Analysis of RED," in *IEEE INFOCOM 2001*, 2001, pp. 1510–1519.

[63] F. Agharebparast and V. C. M. Leung, "A New Traffic Rate Estimation and Monitoring Algorithm for the QoS-Enabled Internet," in *Proc. IEEE GLOBECOM 2003*, December 2003, pp. 3883–3887.

[64] T. Bonald, M. May, and J. C. Bolot, "Analytic Evaluation of RED Performance," in *Proc. IEEE INFOCOM 2000*, March 2000, pp. 1145–1424.

[65] M. May, J. Bolot, C. Doit, and B. Lyles, "Reasons Not To Deploy RED," in *Proc. Int'l Workshop Quality-of-Service (IWQoS)*, 1999, pp. 260–262.

[66] T. V. Lakshman, U. Madhow, and B. Suter, "TCP Performance with Random Loss and Bidirectional Congestion," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 541–555, October 2000.

[67] A. Misra and T. Ott, "Jointly Coordinating ECN and TCP for Rapid Adaptation to Varying Bandwidth," in *Proc. IEEE MILCOM 2001*, vol. 1, 2001, pp. 719–725.

[68] D. E. Lapsley and S. H. Low, "Random Exponential Marking for Internet Congestion Control," in *Proc. IEEE GLOBECOM 1999*, December 1999, pp. 66–74.

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

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

[71] R. Johari and D. K. H. Tan, "End-to-End Congestion Control for the Internet: Delays and Stability," *IEEE/ACM Transactions On Networking*, vol. 9, no. 6, pp. 818–832, December 2001.

[72] N. D. Hayes, "Roots of the Transcendental Equation Associated with a Certain Differential-Difference Equation," *Journal of London Mathematics Society*, vol. 25, pp. 226–232, 1950.

[73] L. Zhang, S. Shenker, and D. D. Clark, "Observations and Dynamics of a Congestion Control Algorithm: The Eff ects of Two-Way Traffic," in *Proc. ACM SIGCOMM 1991*, 1991, pp. 133–147.

[74] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Performance of two-way tcp traffic over asymmetric access links," in *Interop'97 Engineers' Conference*, 1997.

[75] J.-P. Ebert and A. Willig, "A gilber-elliot bit error model and the efficient use in packet level simulation," Technical University Berlin Tecommunications Network Group, Tech. Rep. TKN-99-002, March 1999.

[76] A.-H. A. Abouzeid, S. Roy, and M. Azizoglu, "Stochastic Modeling of TCP over Lossy Links," in *Proc. IEEE INFOCOM 2000*, 2000, pp. 1724–1733.

[77] ——, "Comprehensive Performance Analysis of a TCP Session Over a Wireless Fading Link With Queueing," *IEEE Transactions on Wireless Communications*, vol. 2, no. 2, pp. 344–356, 2003.

[78] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," IEEE 802.11 Standard Part II, 1997.

[79] D. Katabi, "Decoupling Congestion Control and Bandwidth Allocation Policy with Application to High Bandwidth-Delay Product Networks," Ph.D. Dissertation, Massachusetts Institute of Technology, March 2003.

[80] B. P. Wydrowski, "Techniques in Internet Congestion Control," Ph.D. Dissertation, University of Melbourne, 2003.

[81] E. Hashem, "Analysis of Random Drop for Gateway Congestion Control," Laboratory of Computer Science, MIT, Cambridge, MA, Tech. Rep. LCS TR-465, 1989.

[82] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on Queue Management and congestion Avoidance in the Internet," *IETF RFC 2309*, April 1998.

[83] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High-Speed Networks," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 33–46, 2003.

[84] G. Cheng, K. Xu, and N. Ansari, "Core-Stateless Proportional Fair Queuing for AF Traffic," in *Proc. IEEE GLOBECOM 2004*, Dallas, Texas, USA, DEcember 2004.

[85] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Stochastic Fair Blue: A Queue Management algorithm for Enforcing Fairness," in *Proc. IEEE INFOCOM 2001*, 2001.

[86] R. Pan, B. Prabhaker, and K. Psounis, "CHOKE, A Stateless Active Queue Management Scheme for Approximating Fair bandwidth Allocation," in *Proc. IEEE INFOCOM 2000*, 2000, pp. 942–951.

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

[88] M. May, T. Bonald, and J. Bolot, "analytic Evaluation of RED Performance," in *Proc. IEEE INFOCOM 2000*, 2000, pp. 1415–1424.

[89] V. firoiu and M. Borden, "A Study of Active Queue Management for Congestion Control," in *Proc. IEEE INFOCOM 2000*, 2000, pp. 1435–1444.

[90] S. H. Low, F. Paganini, J. Wang, and J. C. Doyle, "Linear Stability of TCP/RED and a Scalable Control," *Elsevier Journal of Computer Networks*, vol. 43, pp. 633–647, 2003.

[91] M. Christiansen, K. Jeffay, D. Ott, and F. Smith, "Tuning RED for Web Traffic," in *Proc. ACM SIGCOMM 2000*, 2000, pp. 139–150.

[92] C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "On designing Improved Controllers for AQM Routers Supporting TCP Flows," in *Proc. IEEE INFOCOM 2001*, vol. 3, 2001, pp. 1726–1734.

[93] S. Athuraliya, S. H. Low, V. H. Li, and Q. Yin, "REM: Active Queue Management," *IEEE Network*, vol. 15, no. 3, pp. 48–53, 2001.

[94] E.-C. Park, H. Lim, K.-J. Park, and C.-H. Choi, "Analysis and Design of the Virtual Rate Control Algorithm for Stabilizing Queues in TCP Networks," *Elsevier Journal of Computer Networks*, vol. 44, pp. 17–41, January 2004.

[95] S. Kunniyur and R. Srikant, "Analysis and Design of an Adaptive Virtual Queue (AVQ) algorithm for Active Queue Management," in *Proc. ACM SIGCOMM 2001*, 2001, pp. 123–134.

[96] ——, "End-to-end Congestion Control Schemes: Utility Functions, Random Losses and ECN Marks," in *Proc. IEEE INFOCOM 2000*, 2000, pp. 1323–1332.

[97] L. Zhu and N. Ansari, "Local Stability of a New Adaptive Queue Management (AQM) Acheme," *IEEE Communications Letters*, vol. 8, no. 6, pp. 406–408, June 2004.

[98] F. P. Kelly, "Charging and Rate Control for Elastic Traffic," *European Transactions on Telecommunications*, vol. 8, pp. 33–37, 1997.

[99] F. P. Kelly, A. Maulloo, and D. Tan, "Rate Control for Commuication Networks: shadow Prices, Proportional Fairness and Stability," *Journal of Operation Research Society*, vol. 49, no. 3, pp. 237–252, March 1998.

[100] F. P. Kelly, "Fairness and Stability of End-to-end Congestion Control," *European Journal of Control*, vol. 9, pp. 149–165, 2003.

[101] S. H. Low and D. E. Lapsley, "Optimization Flow Control, I:Basic Algorithm and Convergence," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 861–874, December 1999.

[102] R. Srikant, "Models and Methods for analyzing Internet Congestion Control Algorithms," [Available Online]: http://comm.csi.uiuc.edu/~srikant.

[103] S. H. Low, F. Paganini, J. Wang, S. Adlakha, and J. C. Doyle, "Dynamics of TCP/RED and a Scalable Control," in *Proc. IEEE INFOCOM 2002*, 2002, pp. 239–248.

[104] F. Paganini, Z. Wang, J. C. Doyle, and S. H. Low, "Congestion Control for High Performance, Stability, and Fairness in General Networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 43–56, February 2005.

[105] S. Biaz and N. H. Vaidya, "De-randomizing Congestion Losses to Improve TCP Performance over Wired-Wireless Networks," Computer Science and Software Engineering Dept., Auburn University, Technical Report CSSE03-01, October 2003.

[106] A. Zanella, G. Procissi, M. Gerla, and M. Y. Sanadidi, "TCP Westwood: Analytic Model and Performance Evaluation," in *Proc. IEEE GLOBECOM 2001*, Sant Antonio, Texas, December 2001, pp. 1698–1702.

[107] G. Vinnicombe, "Robust Congestion Control for the Internet," [Available Online]: http://www-control.eng.cam.ac.uk/gv/internet/sigcomm.pdf, February 2002.

[108] S. Q. Li and C. Hwang, "Link Capacity Allocation and Network Control by Filtered Input Rate in High-speed Networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 1, pp. 10–25, February 1995.

[109] H. S. Kim and N. B. Shroff, "Loss Probability Calculations and Asymptotic Analysis for Finite Buffer Multiplexers," *IEEE/ACM Transactions on Networking*, vol. 9, no. 6, pp. 755–768, December 2001.

[110] R. Krishnan, J. P. G. Sterbenz, W. M. Eddy, G. Partridge, and M. Allman, "Explicit Transport Error Notification (ETEN) for Error-prone Wireless and Satellite Networks," *Elsevier Journal of Computer Networks*, vol. 46, pp. 343–362, July 2004.

[111] D. Y. Eun and N. B. Shroff, "A Measurement-Analytic Framework for QoS Estimation Based on the Dominant Time Scale," *IEEE/ACM Transactions on Networking*, vol. 11, no. 2, pp. 222–235, April 2003.

[112] M. Handley, J. Padhye, and S. Floyd, "TCP Friendly Rate Control (TFRC): Protocol Specification," Work in Progress (IETF Internet Draft), [Available Online]: http://citeseer.ist.psu.edu/handley01tcp.html, 2001.

[113] R. Rejaie, M. Handley, and D. Estrin, "An End-to-end Rate-based Congestion Control for Realtime Streams in the Internet," in *Proc. IEEE INFOCOM 1999*, March 1999.

[114] S. Ramakrishnan, S. Kalyanaraman, J. Wen, and H. Ozbay, "Effect of Time Delay in Network Traffic Control," in *Proc. Automatic Controls Conference (ACC) 2001*, 2001.

[115] M. Vishal, W.-B. Gong, and D. Towsley, "Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED," in *Proc. ACM SIGCOMM 2000*, 2000.

[116] S. Floyd, "HighSpeed TCP for Large Congestion Windows," Internet Draft Work in Progress, http://www.icir.org/floyd/hstcp.html, February 2003.

[117] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth Delay Product Networks," in *Proc. ACM SIGCOMM 2002*, vol. 32, no. 4, August 2002, pp. 89–102.

[118] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One More Bit Is Enough," in *Proc. ACM SIGCOMM 2005*, 2005.

[119] Y. Zhang, S.-R. Kang, and D. Loguinov, "Delayed Stability and Performance of Distributed Congestion Control," in *Proc. ACM SIGCOMM 2004*, vol. 34, no. 4, Portland, Oregon, USA, August 2004, pp. 307–318.

[120] R. Bronson, *Schaum's Outline of Theory and Problems of Matrix Operations*. McGraw-Hill, 1988.

[121] W. G. Kelley and A. C. Peterson, *Difference Equations*. Harcourt/Academic Press, 2001.

[122] T. Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," Submitted fo publication, http://www-lce.eng.cam.ac.uk/~ctk21/scalable, December 2002.

[123] C. Jin, D. Wei, and S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," California Institute of Technology, Pasadena, CA, Tech. Rep. CaltechCSTR:2003:010, 2003, http://netlab.caltech.edu/FAST.