



**The Design Analysis and Implementation of a Novel TCP
Congestion Control Scheme**

Wei Xu

A thesis submitted for the degree of Doctor of Philosophy

Department of Electrical and Electronic Engineering

Adelaide University

August 2001

Contents

List of Figures	v
List of Acronyms	xi
Abstract	xiii
Declaration	xv
Acknowledgments	xvii
1 Introduction.....	1
2 TCP Background	7
3 A Novel Approach to TCP Congestion Control.....	13
3.1 Congestion Control in Packet Switched Networks	14
3.2 TCP Congestion Control	16
3.2.1 Fundamentals of TCP Congestion Control Mechanism	16
3.2.2 TCP Tahoe	17
3.2.3 TCP Reno	19
3.2.4 TCP Vegas and Other TCP Flavours	20
3.3 Limitations of Existing TCP Congestion Control Schemes	22
3.4 A Self-Tuning PD Based TCP Congestion Control Scheme	25
3.4.1 Main Features of ST-PD TCP	25
3.4.2 Formulation	26
3.4.3 Design Aspects of ST-PD TCP	27
3.4.3.1 Design of PD Controller	29
3.4.3.2 Design of Fuzzy Logic Supervisory Controller.....	30
3.4.3.3 Flow-Chart of ST-PD TCP Congestion Control Algorithm	32

3.4.3.4 Stability Discussions of ST-PD TCP Congestion Control Scheme.....	34
3.5 Analysis and Simulation Results.....	35
3.5.1 A Simple TCP Connection Case.....	35
3.5.1.1 The Behaviour of TCP under Sufficient Buffer Conditions.....	37
3.5.1.2 The Behaviour of TCP under Insufficient Buffer Conditions.....	40
3.5.2 Multiple TCP Connections.....	43
3.5.2.1 A Fixed Bottleneck Link Case.....	44
3.5.2.2 A Moving Bottleneck Link Case.....	48
3.5.3 The Impact on Round-Trip Delays of TCP Connections.....	51
3.6 Summary	56
4 Performance Analysis of ST-PD TCP over Wireless links	59
4.1 Background	60
4.1.1 Overview of Wireless Data Networks	60
4.1.2 Characteristics of Wireless Channels.....	63
4.1.3 TCP Performance Degradations over Wireless Links	64
4.1.4 Existing Performance Enhancing Techniques and Limitations	65
4.2 Decoupling Error Control and Congestion Control in ST-PD TCP	68
4.3 Performance Analysis.....	71
4.3.1 The Impact of Wireless Losses on the Congestion Window of ST-PD TCP	71
4.3.2 Transient Response Analysis of ST-PD TCP under Disturbance.....	73
4.3.3 Numerical Results	77
4.4 Simulation Results.....	80
4.4.1 TCP Performance over Terrestrial Wireless Links.....	80
4.4.1.1 TCP Performance under Uniformly Distributed Wireless Loss Conditions.....	81
4.4.1.2 TCP Performance under Bursty Wireless Loss Conditions.....	85
4.4.2 TCP Performance over GEO Satellite Links	89
4.5 Summary	96
5 The Implementation of ST-PD TCP and Experiment Results	99
5.1 Data Structures of TCP/IP Implementation in Linux Kernel	100
5.2 Implementation of ST-PD TCP in Linux Kernel	103

5.2.1 Software Implementation of Floating-point Arithmetic Operations	105
5.2.2 Refining Timestamp Resolution	106
5.3 Experiment Environments	108
5.3.1 Testbed Network Configuration.....	108
5.3.2 Methods for Experiment Data Collection	109
5.3.3 Implementation of Data Link Emulator	110
5.4 Experiment Results.....	117
5.4.1 Results of Experiments over Lossless Links.....	117
5.4.2 Results of Experiments over Lossy Links.....	133
5.5 Summary	139
6 The Coexistence of ST-PD TCP with Existing TCP Deployment	141
7 Conclusions.....	151
Appendix. The Kernel Architecture of 4BSD Unix Networking Subsystem	157
REFERENCES	159

List of Figures

Fig. 2.1 DoD four-layer network model	8
Fig. 3.1 A TCP connection in an arbitrary network	26
Fig. 3.2 Closed-loop control diagram	26
Fig. 3.3 ST-PD control structure.....	29
Fig. 3.4 Fuzzification membership function.....	31
Fig. 3.5(a) Defuzzification function for K_p	32
Fig. 3.5(b) Defuzzification function for K_d	32
Fig. 3.6 Flow-chart of ST-PD TCP congestion control algorithm	33
Fig. 3.7 A simple TCP connection.....	35
Fig. 3.8(a) Congestion window trace of TCP Reno under a sufficient buffer condition.....	37
Fig. 3.8(b) Congestion window trace of ST-PD TCP under a sufficient buffer condition.....	37
Fig. 3.9(a) Buffer occupancy of node R under the congestion control of TCP Reno under a sufficient buffer condition.....	39
Fig. 3.9(b) Buffer occupancy of node R under the congestion control of ST-PD TCP.. under a sufficient buffer condition.....	39
Fig. 3.10 Throughput of TCP Reno and ST-PD TCP under a sufficient buffer condition.....	40
Fig. 3.11(a) Congestion window trace of TCP Reno under an insufficient buffer condition..	41
Fig. 3.11(b) Congestion window trace of ST-PD TCP under an insufficient buffer condition	41
Fig. 3.12(a) Buffer occupancy of node R under the congestion control of TCP Reno under an insufficient buffer condition.....	42
Fig. 3.12(b) Buffer occupancy of node R under the congestion control of ST-PD TCP.. under an insufficient buffer condition.....	42
Fig. 3.13 Throughput of TCP Reno and ST-PD TCP under an insufficient buffer condition.	43
Fig. 3.14 Network configuration of a fixed bottleneck link case	44
Fig. 3.15(a) Queuing behaviour of TCP Reno.....	45

Fig. 3.15(b) Throughput of TCP Reno	45
Fig. 3.16(a) Queuing behaviour of TCP Vegas	46
Fig. 3.16(b) Throughput of TCP Vegas.....	46
Fig. 3.17(a) Queuing behaviour of ST-PD TCP	47
Fig. 3.17(b) Throughput of ST-PD TCP.....	47
Fig. 3.18 Network configuration of a moving bottleneck link case	48
Fig. 3.19(a) Buffer occupancy at R_1 and R_2 under the congestion control of ST-PD TCP	49
Fig. 3.19(b) Throughput of the three ST-PD TCP connections.....	49
Fig. 3.20(a) Buffer occupancy at R_1 and R_2 under the congestion control of TCP Reno.....	50
Fig. 3.20(b) Throughput of the three TCP Reno connections	50
Fig. 3.21 Network configuration of FTP and Telnet connections	52
Fig. 3.22(a) Buffer occupancy of node R_1 under the congestion control of ST-PD TCP	52
Fig. 3.22(b) RTT trace of Telnet packets under the congestion control of ST-PD TCP	53
Fig. 3.23(a) Buffer occupancy of node R_1 under the congestion control of TCP Reno	54
Fig. 3.23(b) RTT trace of Telnet packets under the congestion control of TCP Reno.....	54
Fig. 3.24 Acknowledged sequence number of FTP packets.....	55
Fig. 4.1 Flow-chart of ST-PD TCP error control and congestion control	70
Fig. 4.2(a) Buffer occupancy before a wireless loss.....	71
Fig. 4.2(b) Buffer occupancy after a wireless loss	71
Fig. 4.3 The effect of a packet loss on ST-PD TCP	72
Fig. 4.4 Queuing dynamics at node R.....	73
Fig. 4.5 Normalised packet arrival rate	78
Fig. 4.6 Throughput of ST-PD TCP under various packet error probabilities	79
Fig. 4.7 A TCP wireless connection	80
Fig. 4.8 Throughput of different versions of TCP	81
Fig. 4.9 Congestion window trace of TCP Tahoe	83
Fig. 4.10 Congestion window trace of TCP Reno	83
Fig. 4.11 Congestion window trace of TCP Vegas	84
Fig. 4.12 Congestion window trace of ST-PD TCP	84
Fig. 4.13 Clustered error distribution in respect of packet index	86
Fig. 4.14 Congestion window trace of TCP Reno in respect of packet index.....	86

Fig. 4.15 Congestion window trace of ST-PD TCP in respect of packet index	86
Fig. 4.16 Congestion window trace of TCP Reno in respect of time	88
Fig. 4.17 Congestion window trace of ST-PD TCP in respect of time	88
Fig. 4.18 A TCP connection over GEO satellite links.....	91
Fig. 4.19(a) Congestion window trace of TCP Reno over GEO satellite links	92
Fig. 4.19(b) Throughput of TCP Reno over GEO satellite links.....	92
Fig. 4.20(a) Congestion window trace of TCP Vegas over GEO satellite links	93
Fig. 4.20(b) Throughput of TCP Vegas over GEO satellite links	93
Fig. 4.21(a) Congestion window trace of ST-PD TCP over GEO satellite links	94
Fig. 4.21(b) Throughput of ST-PD TCP over GEO satellite links	94
Fig. 4.22 A comparison of acknowledged segment sequence traces.....	95
Fig. 5.1 Socket data structures.....	101
Fig. 5.2 Structure of tcp_opt.....	102
Fig. 5.3 Experiment setting.....	108
Fig. 5.4 A sample TCP traffic trace collected by tcpdump	109
Fig. 5.5 Block diagram of data link emulator.....	111
Fig. 5.6 Address area map of internal RAM and Flash ROM of M30620FP	112
Fig. 5.7 Link propagation delay emulation.....	113
Fig. 5.8 Circular delay buffer structure	113
Fig. 5.9(a) ACF of $LCG(16807,0,2^{31}-1,1)$ sequence.....	115
Fig. 5.9(b) ACF of uniformly distributed sequence generated by Matlab.....	115
Fig. 5.10 Pair plot of the $LCG(16807,0,2^{31}-1,1)$ sequence	116
Fig. 5.11(a) Sequence traces of TCP Reno over 115.2kbps 0ms-delayed serial link.....	118
Fig. 5.11(b) Sequence traces of ST-PD TCP over 115.2kbps 0ms-delayed serial link.....	118
Fig. 5.12(a) RTT trace of TCP Reno over 115.2kbps 0ms-delayed serial link	119
Fig. 5.12(b) RTT trace of ST-PD TCP over 115.2kbps 0ms-delayed serial link	119
Fig. 5.13(a) Sequence trace of TCP Reno over 38.4kbps 80ms-delayed serial link	121
Fig. 5.13(b) Sequence trace of ST-PD TCP over 38.4kbps 80ms-delayed serial link	121
Fig. 5.14(a) RTT trace of TCP Reno over 38.4kbps 80ms-delayed serial link	122
Fig. 5.14(b) RTT trace of ST-PD TCP over 38.4kbps 80ms-delayed serial link	122
Fig. 5.15(a) Sequence trace of TCP Reno over 38.4kbps 250ms-delayed serial link	123

Fig. 5.15(b) Sequence trace of ST-PD TCP over 38.4kbps 250ms-delayed serial link	123
Fig. 5.16(a) RTT trace of TCP Reno over 38.4kbps 250ms-delayed serial link	124
Fig. 5.16(b) RTT trace of ST-PD TCP over 38.4kbps 250ms-delayed serial link	124
Fig. 5.17(a) Throughput of TCP Reno over 115.2kbps 0ms-delayed serial link.....	126
Fig. 5.17(b) Throughput of ST-PD TCP over 115.2kbps 0ms-delayed serial link.....	126
Fig. 5.18(a) Throughput of TCP Reno over 38.4kbps 80ms-delayed serial link.....	127
Fig. 5.18(b) Throughput of ST-PD TCP over 38.4kbps 80ms-delayed serial link.....	127
Fig. 5.19(a) Throughput of TCP Reno over 38.4kbps 250ms-delayed serial link.....	128
Fig. 5.19(b) Throughput of ST-PD TCP over 38.4kbps 250ms-delayed serial link.....	128
Fig. 5.20(a) Arrived packet sequence traces of two competing TCP Reno connections.....	130
Fig. 5.20(b) Arrived packet sequence traces of two competing ST-PD TCP connections....	130
Fig. 5.21(a) Acknowledged packet sequence traces of two competing TCP Reno connections	131
Fig. 5.21(b) Acknowledged packet sequence traces of two competing ST-PD TCP connections	131
Fig. 5.22(a) Throughput of two competing TCP Reno connections.....	132
Fig. 5.22(b) Throughput of two competing ST-PD TCP connections.....	132
Fig. 5.23(a) Comparison of acknowledged packet sequence traces between Reno and ST-PD TCP over 38.4kbps 250ms-delayed serial link with packet error ratio of 10^{-1}	134
Fig. 5.23(b) Comparison of throughput between Reno and ST-PD TCP	over 38.4kbps 250ms-delayed serial link with packet error ratio of 10^{-1} 134
Fig. 5.24(a) Comparison of acknowledged packet sequence traces between Reno and ST-PD TCP over 38.4kbps 250ms-delayed serial link with packet error ratio of 10^{-2}	135
Fig. 5.24(b) Comparison of throughput between Reno and ST-PD TCP	over 38.4kbps 250ms-delayed serial link with packet error ratio of 10^{-2} 135
Fig. 5.25(a) Comparison of acknowledged packet sequence traces between Reno and ST-PD TCP over 38.4kbps 80ms-delayed serial link with packet error ratio of 10^{-1}	136
Fig. 5.25(b) Comparison of throughput between Reno and ST-PD TCP over 38.4kbps 80ms- delayed serial link with packet error ratio of 10^{-1}	136
Fig. 5.26(a) Comparison of acknowledged packet sequence traces between Reno and ST-PD TCP over 38.4kbps 80ms-delayed serial link with packet error ratio of 10^{-2}	137

Fig. 5.26(b) Comparison of throughput between Reno and ST-PD TCP over 38.4kbps 80ms- delayed serial link with packet error ratio of 10^{-2}	137
Fig. 6.1 Network configuration of two mixed-version TCP connections.....	142
Fig. 6.2(a) Throughput of two TCP connections.....	143
Fig. 6.2(b) Throughput of three TCP connections.....	144
Fig. 6.3(a) Throughput of TCP Reno connection S_1D_1 and ST-PD TCP connection S_2D_2 (link R_1R_2 with packet error ratio of 10^{-2})	145
Fig. 6.3(b) Throughput of TCP Reno connection S_1D_1 and TCP Reno connection S_2D_2 (link R_1R_2 with packet error ratio of 10^{-2})	145
Fig. 6.4 A comparison of acknowledged sequence traces of Reno and ST-PD TCP connection S_2D_2 (link R_1R_2 with packet error ratio of 10^{-2}).....	146
Fig. 6.5(a) Throughput of TCP Reno connection S_1D_1 and ST-PD TCP connection S_2D_2 (link S_2R_1 with packet error ratio of 10^{-2}).....	147
Fig. 6.5(b) Throughput of TCP Reno connection S_1D_1 and TCP Reno connection S_2D_2 (link S_2R_1 with packet error ratio of 10^{-2}).....	147
Fig. 6.6 A comparison of acknowledged packet sequence traces of Reno and ST-PD TCP connection S_2D_2 (link S_2R_1 with packet error ratio of 10^{-2}).....	148

List of Acronyms

ABR	Available Bit Rate
AF	Address Family
ARDIS	Advanced Radio Data Information Service
ARP	Address Resolution Protocol
ARPANET	Advanced Research Project Agency NETwork
ARQ	Automatic Repeat reQuest
ATM	Asynchronous Transfer Mode
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BSD	Berkeley Software Distribution
CAC	Connection Admission Control
CARD	Congestion Avoidance using Round-trip Delay
CDPD	Cellular Digital Packet Data
ECN	Explicit Congestion Notification
ELN	Explicit Loss Notification
FEC	Forward Error Correction
FNC	Federal Networking Council
FPU	Floating-point Processing Unit
FTP	File Transfer Protocol
GEO	Geosynchronous Earth Orbit
GII	Global Information Infrastructure
GPRS	General Packet Radio Service
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IMP	Interface Message Processor
IMT2000	International Mobile Telecommunications 2000

IP	Internet Protocol
IPng	IP next generation
ISI	Inter-Symbol Interference
LCG	Linear Congruential Generator
LEO	Low Earth Orbit
LER	Link Error Rate
LMDS	Local Multipoint Distribution System
MMDS	Multipoint Multichannel Distribution Service
NCP	Network Control Protocol
NWG	Network Working Group
OSI	Open Systems Interconnection
PDN	Public Data Network
PID	Proportional-Integral-Derivative
POSIX	Portable Operation System for unIX
PPP	Point-to-Point Protocol
RARP	Reverse Address Resolution Protocol
RFC	Request For Comments
RMD	Ram Mobile Data
RNG	Random Number Generator
RTT	Round-Trip Time
SMTP	Simple Mail Transfer Protocol
SNR	Signal Noise Ratio
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
USAT	Ultra Small Aperture Terminal
VC	Virtual Circuit
VSAT	Very Small Aperture Terminal
WLL	Wireless Local Loop

Abstract

The thesis presents the design, analysis and implementation aspects of a new TCP congestion control scheme.

To overcome the shortfalls associated with existing TCP congestion control schemes, which are heuristic based, a novel Self-Tuning Proportional and Derivative control based TCP (ST-PD TCP) congestion control scheme is proposed, which approaches the congestion control problem from a control-theoretic perspective. In this scheme, the TCP congestion window is adjusted by a classic PD controller aiming to keep the buffer occupancy of the bottleneck node on the connection path at an ideal operating level. The control gains of the PD controller are tuned on-line by a fuzzy logic controller based on the perceived bandwidth-delay product of the TCP connection. This scheme gives ST-PD TCP several advantages over current TCP implementations. These include rapid response to bandwidth variations, insensitivity to buffer sizes, and significant improvement of TCP throughput over lossy links by decoupling congestion control and error control functions of TCP.

In addition to simulations and experiments, analytical studies are conducted to reveal ST-PD TCP's performance over lossy links. To quantify the throughput of ST-PD TCP over wireless links, the effects of wireless losses on the congestion window of ST-PD TCP are modelled as disturbances on the control errors estimated by the PD controller. The throughput of ST-PD TCP over lossy links is derived by transient response analysis of the PD controller under such disturbances.

In order to demonstrate that the design of ST-PD TCP does not make any unrealistic assumptions, ST-PD TCP has been implemented into the kernel of Linux-2.2.17 and its performance was measured in a testbed network. The experiment results are consistent with the analytical and simulation studies conducted in both wired and wireless networking environments.

The issues of ST-PD TCP's coexistence with current TCP deployment are also explored in this thesis.

Declaration

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of my thesis, when deposited in the University Library, being available for loan and photocopying.

Signed: _____

Date: 22/01/02

Acknowledgments

I wish to express sincere gratitude to my research supervisors Dr A. G. Qureshi, Dr K. W. Sarkies and Prof L. White for their intelligent guidance and constant encouragement. I would like to especially thank Dr K. W. Sarkies, who supervised my research after Dr A. G. Qureshi's departure, for his continuous advice, his availability, and his extensive review and comments on this thesis.

I would also like to thank Dr C. C. Lim for his advice on the issue of transient response analysis of control systems.

I am grateful to every member of the Department who has helped me during my stay here.

Chapter 1

Introduction

Perhaps the most influential social event in the last decade of the past century was the popularisation of the Internet. Evolved from a research network used by a group of people with special interests in computer communications, Internet has become a ubiquitous information infrastructure in ordinary people's daily life. E-mail and on-line chat offer speedier and less expensive alternatives to traditional ways people communicate with each other. World Wide Web (WWW) becomes another important information source where people can get their daily news besides traditional media. Despite suffering a few setbacks, perhaps due to being ahead of its time, e-commerce will certainly revolutionise the way business is conducted in the future. Most importantly, the fact that no one can control Internet may change the political landscape in some countries where traditional media are under tight control and freedom of press is stifled.

Since its inception three decades ago, Internet has been expanding at an unprecedented pace in terms of both its scale and its applications, accompanied by revolutionary changes in the computer and telecommunications industries. Today's personal computers have more processing power than the astronomically priced mainframes thirty years ago, and the 50kbps leased lines that constituted the backbone of Internet's predecessor have been replaced by optical fibre links with capacities of gigabits per second. The only thing that has remained relatively stable is the core protocol suite TCP/IP, upon which Internet is built. The following definition of Internet by the Federal Networking Council (FNC) reflects the TCP/IP-centric nature of Internet.

"Internet" refers to the global information system that — (i) is logically linked together by a global unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ons; (ii) is able to support communications using the

Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions/follow-ons, and/or other IP-compatible protocols; and (iii) provides, uses, or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure described herein.

From this definition, it can be seen that the whole Internet is built upon TCP/IP[†]. According to the Open Systems Interconnection (OSI) 7-layer reference model, IP is a network layer (Layer 3) protocol. Due to robustness considerations, IP offers very basic datagram forwarding services. At each intermediate node, IP datagrams are forwarded independent of each other, and no state information regarding the passing flows is kept in the intermediate node. Because of this, redundant paths can be exploited in the presence of unreliability or unavailability of some parts in the network. Also because of this, end-to-end sequenced data delivery cannot be guaranteed in the IP layer. TCP, a transport layer (Layer 4) protocol operating only at end-systems, offers reliable sequenced end-to-end data delivery services on top of the IP layer. Complex networking functions, such as connection setup, error control and congestion control, are implemented in TCP. Among those functions that TCP carries out, congestion control is the most important. As Internet is a loose interconnection of heterogeneous physical networks, it has no centralised traffic control mechanism. Congestion control in Internet relies on each individual TCP source regulating its own emitted traffic. The Internet expansion history has shown that TCP congestion control is so effective that it has coped very well with the exponential growth of the Internet scale and the traffic it accommodated. In fact technically, Internet's success is, to a large extent, contributed by two factors: (i) the stateless datagram based IP services make Internet very resilient; and (ii) the end-to-end based TCP congestion control makes Internet very elastic.

Despite the enormous success, no one pretends that TCP/IP protocol suite is perfect. Since its inception in the late 1970s, efforts aimed to improve its performance and functionality have never stopped. However, restrained by the fact that TCP/IP is a widely deployed product, and that it has been running on computers in government and military organisations, business

[†] The definition does not exclude TCP/IP subsequent extensions/follow-ons (e.g. IPv6) and other IP-compatible protocols (such as UDP). However the overwhelming majority of Internet traffic is carried by TCP/IP.

sectors, research and scientific institutions around the world to conduct non trivial day-to-day work, the progress of TCP/IP technologies is everything but revolutionary.

For example, to deal with the inadequacies of current IP (known as IPv4), such as IP address shortages, insufficient security mechanisms, lack of flexibility in network configuration and management, the Internet Engineering Task Force (IETF) set out to draft the IP next-generation (IPng) proposal in July 1992. During 1995-1996 a set of Request For Comments (RFC) documents finalised the specifications of the new network layer protocol, now officially known as IPv6 [1]. However the transition from IPv4 to IPv6 is extremely slow despite all the advantages of IPv6 over IPv4. Currently there is a small-scaled IPv6 testing network worldwide known as 6Bone. Measures such as deploying IPv4/IPv6 dual-stack routers at the edge of 6Bone and adopting tunnelling techniques inside 6Bone are taken to enable the interoperability between IPv4 and IPv6 during the transition phase.

Compared with IP, the performance of TCP has more direct impact on Internet users' experiences. As a result, more efforts have been devoted to improve the efficiency of TCP operations. Among them, the most active research area focused on TCP congestion control is finding an ideal congestion control strategy that can maximise network resource utilisation without overloading it in a decentralised and uncoordinated way. This is not a trivial task. Further, the prerequisite that any newly developed TCP performance enhancing measures must be able to coexist with legacy systems, without requiring wholesale changes, makes research in this area more challenging. Since the early 1980s, only a small number of proposals for TCP performance enhancements have gained wide acceptance, among them, *slow start/congestion avoidance*, *fast retransmission* and *fast recovery* officially became the key congestion control mechanisms in current TCP implementations.

However, emerging technologies and applications continue to pose challenges to TCP. As terrestrial wireless and satellite networks are increasingly being used for data communications, the inadequacy of current TCP implementations become more apparent. Further, the efficient operations of current TCPs require intermediate nodes to have sufficient buffer space, and existing TCP congestion control schemes lack the mechanism to restrain the queue size growth at intermediate nodes. As a result, packets transmitted by current TCPs

might incur excessive queuing delays, which will have negative effects on a number of interactive-oriented applications, such as Telnet and IP telephony's signalling transmission, which rely on TCP to transport their data across networks.

The work presented in this thesis is another attempt to improve TCP congestion control performance while maintaining the newly devised TCP's coexistence with existing TCP deployment.

The thesis is organised as follows. In chapter 2, the history of TCP is outlined first, followed by a brief review of TCP functionality in IP-based networks.

Chapters 3, 4 and 5 form the core of the thesis, which cover the design, analysis and implementation aspects of the new TCP congestion control scheme respectively.

Chapter 3 begins with a general discussion of the role that congestion control plays in packet switched networks. Then existing TCP congestion control schemes are examined in detail. After analysing the limitations of congestion control mechanisms adopted by current TCP implementations, a Self-Tuning Proportional and Derivative control based TCP (ST-PD TCP) congestion control scheme is proposed, which addresses the congestion control problem from a control-theoretic perspective. Details of the design of the new TCP congestion control scheme and its general behaviour in wired networking environments are presented in this chapter.

Chapter 4 is focused on TCP performance analysis in wireless networking environments. After a brief introduction of wireless channel characteristics and their adverse effects on achievable throughput of existing TCP implementations, ST-PD TCP's solution to this problem is described. A performance analytical model is developed to quantify the impact of wireless losses on the congestion window of ST-PD TCP, from which the achievable throughput of ST-PD TCP is derived. To confirm the numerical analysis, simulation results are presented.

Chapter 5 provides a detailed description of the implementation of the new TCP congestion control scheme in a Linux kernel. To test the performance of ST-PD TCP in a wide range of networking environments, a data link emulator is constructed to emulate the propagation

delay and transmission error characteristics of a physical link in real-time. The comparison experimental results of TCP Reno and ST-PD TCP over a range of lossless and lossy links are presented.

Chapter 6 is devoted to discussing the issues of ST-PD TCP's coexistence with existing TCP deployment. After analysing the adverse effects on ST-PD TCP caused by sharing a same bottleneck queue with TCP connections that use Reno-style congestion control schemes, a line is drawn between situations where the performance of ST-PD TCP is not affected at the presence of TCP Reno connections and the situations where TCP Reno connections do affect the normal operations of ST-PD TCP. Further scenarios are outlined where ST-PD TCP still outperforms current TCP implementations even if its normal operations are adversely affected at the presence of TCP Reno connections.

The thesis is concluded in Chapter 7.

Chapter 2

TCP Background

The origin of TCP can be traced back to the US Department of Defence (DoD) project — Advanced Research Project Agency NETwork (ARPANET) in the 1960s.

In 1961, a paper by L. Kleinrock then at MIT established the packet switching theory for information networks [2], which demonstrated the efficiency of packet switching methods over then dominant circuit switching methods. The packet switching theory was further developed into a conceptual computer network in 1967, when L. Roberts at DARPA proposed ARPANET in a conference [3].

After years of theoretical and conceptual preparation, DARPA funded Bolt Beranek and Newman (BBN) to develop the core packet switching equipment of ARPANET, called Interface Message Processors (IMPs) in 1968. In September 1969, BBN installed the first IMP at the Network Measurement Centre at University of California at Los Angeles (UCLA), which is the first node on the ARPANET. One month later, a second IMP was installed at the Network Information Centre at Stanford Research Institute (SRI), and the first message of ARPANET was sent from UCLA to SRI. By the end of 1969, ARPANET expanded to a four-node network [7].

The concept of layered networking architecture was also pioneered in the design of ARPANET. The idea behind the layered networking architecture is that total communication functions could be divided into several layers, each building upon the services of the one below. This approach facilitates the decoupling of complex networking functions, such that replacing or upgrading one layer does not affect the functionality of the adjacent layers. The architecture of ARPANET is divided into four layers [4], as shown in Figure 2.1. In December 1970, the initial ARPANET Host-to-Host layer protocol known as Network Control Protocol (NCP), which is the direct ancestor of TCP, was finalised by the Network

Working Group (NWG). By the end of 1972, all sites at ARPANET had implemented NCP, and network applications built on top of NCP began to emerge. Electronic mail (email) was first developed in 1972 as a communication tool that enabled ARPANET developers to exchange ideas, and remains one of the most popular network applications decades later [7].

Process/application
Host-host layer
Internet layer
Network access layer

Fig. 2.1 DoD four-layer network model

Being built specifically for ARPANET, NCP was not based on an open-architecture networking principle. This made it difficult for other types of networks available at that time, to be connected to ARPANET. To address the shortfalls of NCP, R. Kahn at DARPA began to formulate an open networking architecture for ARPANET. The following principles regarding to the open networking architecture was documented in his 1972 memorandum [5]:

- ❑ There should be no constraints on the types of networks that can be interconnected, and no internal changes should be required for such internetworking.
- ❑ Data transfer should be on a best effort basis. It is end nodes' responsibility to ensure the integrity of the data transfer.
- ❑ Intermediate nodes should not maintain the state of each individual flow of packets passing through them, and thereby ensure the resilience of a network to various failures.
- ❑ There would be no global control.

These ground rules have had significant influence on computer networking architecture for decades. In 1973, R. Kahn and V. Cerf decided to develop a network protocol based on the open architecture. Their work was published in a paper, which specified the key mechanisms in that protocol [6]:

- ❑ Communication between two processes would logically consist of a stream of octets. The position of any octet in the stream would be used to identify it.
- ❑ Flow control would be done by using sliding windows and acknowledgments. Each acknowledgment would be cumulative for all packets received to that point.
- ❑ A 32-bit IP address was used for global addressing.

In that paper, the protocol was called TCP, which provided both transport and packet forwarding services. It was soon split into two protocols: TCP for reliable transport services and IP for addressing and forwarding services. Another transport protocol UDP was added later to provide connectionless transport services.

Shortly after the paper was published, DARPA funded BBN and two other institutions to implement TCP/IP independently. This was the beginning of long term experimentation and development to evolve and mature the Internet concepts and technology [7].

Since Unix from the Berkeley Software Distributions (BSD) was officially chosen for all DARPA-sponsored projects [8], DARPA decided to fund the Computer System Research Group (CSRG) at University of California at Berkeley (UCB) to incorporate the TCP/IP implementation developed at BBN into BSD Unix in 1979. Until then, computer networking approaches focused on implementing communication protocols as applications rather than an integral part of operating systems.

4BSD was the first Unix operating system that had the networking subsystem in its kernel. Followed by 4.1aBSD alpha release in 1981 and 4.1cBSD beta release in 1982[†], the first stable version of 4BSD, 4.2BSD, was released in 1983. At the same year, DARPA mandated the transition from NCP to TCP/IP on all sites of ARPANET [7].

Because 4.2BSD was government funded, its source code was available to anyone for the cost of distribution. The networking subsystem of 4.2BSD provided a free reference implementation of TCP/IP protocol suite [9], which facilitated the widespread adoption of the protocol suite among research communities and computer industries.

[†] 4.1bBSD is a internal release.

While adhering to TCP/IP specifications, efforts aimed to improve the performance of TCP/IP have never stopped. *Fast retransmission* mechanism was added to the TCP implementation in 4.3BSD Tahoe released in 1988, and *Fast recovery* mechanism was incorporated into the TCP implementation in 4.3BSD Reno released in 1990.

Although ARPANET ceased to exist a long time ago, the key concept of layered networking architecture in ARPANET design has influenced the direction of computer internetworking design. Among several protocol standards specified within the DoD four-layer model, IP (MIL-STD-1777 Internet Protocol) [10] and TCP (MIL-STD-1778 Transmission Control Protocol) [11] have great significance not only to the functionality of the ARPANET, but also to the success of today's global Internet.

In the DoD four-layer model, TCP/IP operates at host-host layer/internet layer respectively, which is equivalent to transport layer/network layer in OSI seven-layer reference model. Since the OSI reference model has gained wide acceptance[†], TCP is now regarded as a transport layer protocol and IP a network layer protocol.

Higher priority on robustness has led ARPANET to adopt a connectionless datagram-based network layer protocol IP. Besides providing a common data format that enables interoperability among heterogeneous computers, IP specifies a global network addressing mechanism. In this layer, packets are forwarded independently of one another, and routing decisions are made on a per packet basis. This is quite different from the Public Data Network (PDN) available at that time — X.25. Although also a packet switched network, X.25 is connection-oriented in the sense that routing decisions are made during the Virtual Circuit (VC) setup phase. Once a VC is established, packets belonging to the same flow always follow a predetermined path. The connectionless datagram-based network layer makes ARPANET very resilient. If nodes and links in some parts of a network fail, packets can be routed around the affected areas. However, since packets are treated independently on a hop-by-hop basis, reliable sequenced end-to-end data delivery cannot be guaranteed in this layer.

[†] DoD 4-layer reference model preceded the OSI 7-layer reference model.

The networking functions of ARPANET were assigned to its protocol layers on the following philosophical basis. The fewer functions that are put into the network layer, the more resilient the network is. Complex network functions should be implemented on higher layer protocols that are run at end systems.

TCP, a transport layer protocol operating at end systems, adds a great deal of functionality to basic IP services by providing reliable stream-oriented end-to-end data delivery. To make up for the deficiencies of IP layer services, TCP performs three major functions:

- ❑ Connection control: set up, maintain and tear down end-to-end connections over the connectionless IP layer.
- ❑ Error control: use sequence numbers and acknowledgments to ensure the integrity of data transfer and retransmit lost or corrupted packets.
- ❑ Congestion control: probe network load conditions and adjust emitted traffic to maximise throughput without overloading the network.

Among the three functions, congestion control is the most important one, since there is no resource reservation nor connection admission control in TCP/IP based services. The objectives of TCP congestion control are two-fold: maximising the utilisation of network resource; and congestion prevention. As far as congestion prevention is concerned, TCP congestion control mechanisms are very effective: TCP has been successfully coping with the exponential growth of Internet from 562 registered ARPANET hosts in August 1983 to tens of millions of hosts in today's Internet [7]. However, in terms of network resource utilisation, the performance of TCP congestion control mechanisms is far from satisfactory [12][13]. This is due to the heuristic and overly conservative approach adopted by the existing TCP congestion control algorithms. Considering the fact that the majority of Internet application layer protocols, such as HTTP, FTP, SMTP, Telnet etc., rely on TCP to transport their data across networks, the congestion control mechanisms in current TCP implementations need to be revisited.

Chapter 3

A Novel Approach to TCP Congestion Control

In packet switched networks, resources such as bandwidth and buffer space are shared through statistical multiplexing. For each connection in such networks, there is no minimum bandwidth guarantee if networking resources in intermediate nodes cannot be reserved beforehand. This is the case for TCP connections, as intermediate nodes in IP-based networks do not keep the state of passing flows. As a result, TCP sources must exercise congestion control in order to adapt to network load variations.

As mentioned earlier in chapter 2, TCP/IP was conceived on an open-architecture networking principle, and one of the motives of replacing NCP with TCP is that TCP should impose no constraints on the types of networks that can be interconnected and it should require no internal changes for such internetworking. Thus it would be impractical for TCP to anticipate any assistance from intermediate nodes while conducting congestion control. To satisfy the open-architecture networking requirement, TCP congestion control is conducted strictly on an end-to-end basis. Each TCP source node relies only on the acknowledgments returned from its corresponding node to infer current network load conditions. Those stringent conditions pose significant challenges in finding an efficient TCP congestion control scheme.

In this chapter, after a brief introduction of congestion control in packet switched networks, various TCP congestion control schemes proposed so far are reviewed in detail. Based on the analysis of the existing TCP congestion control schemes, a self-tuning PD based TCP congestion control scheme, which addresses the congestion control problem from a control-theoretic perspective, is proposed. Details of the formulation of the new TCP congestion control scheme and its general behaviour in wired networking environments are presented in this chapter.

3.1 Congestion Control in Packet Switched Networks

Congestion control is not a critical issue in circuit switched telephone networks. Once a circuit is established, fixed network resources are allocated along the path for its exclusive use, and there is no uncertainty about the bandwidth and delays [14]. To ensure enough network resources are available, call admission control is carried out during the call setup phase.

However, the voice-centric circuit switched networking paradigm is not suitable for computer networks. The data traffic generated by computers tends to be very bursty and unpredictable, therefore preserving resources would lead to inefficient network resource utilisation. Packet switching technology improves the efficiency of network resource utilisation by sharing resources among network users through statistical multiplexing. Due to the lack of resource reservation and call admission control, congestion control becomes extremely important in making participating sources adjust their emitted traffic to prevent the network from congestion.

Since the emergence of packet switching technology, dozens of congestion control schemes have been proposed:

- 1) Depending on where the network traffic is regulated, current congestion control schemes can be classified into link-by-link based versus end-to-end based. In link-by-link based congestion control schemes, source nodes as well as intermediate nodes adjust their outgoing traffic according to the feedback information from their immediate downstream nodes. In end-to-end based congestion schemes, only source nodes adjust traffic according to the feedback information from their destination nodes, whereas the intermediate nodes and links are treated as if they were a single transmission medium. The advantage of link-by-link based congestion control schemes is that the congestion control is carried out locally. Thus they respond more promptly to network congestion than end-to-end based congestion control schemes do. Each individual link's resource can be more efficiently utilised under link-by-link based congestion control schemes. The limitation of hop-by-hop based congestion control schemes is that all nodes in the network must have some sort of

homogeneity, ie. there must be a set of rules and conventions that all networking nodes agree upon on how to issue feedback messages to their immediate upstream nodes and how to respond on receiving feedback information from their immediate downstream nodes.

- 2) Depending on how the network traffic is regulated, current congestion control schemes can be classified into rate based versus credit based. In rate based congestion control schemes, source nodes are instructed to adjust their transmission speed; whereas in credit based congestion control schemes, source nodes are given a certain amount of credit that limits the number of data the source nodes can transmit. Source nodes can keep on transmitting until they run out of credit. Credit is renewed after the network can accept more data. Compared with credit based congestion control schemes, the network traffic regulated by rate based congestion control schemes is less bursty. However, rate based congestion control schemes require intermediate nodes to compute each competing connection's share of the available bandwidth, which not only requires the homogeneity of all networking nodes, but also increases the functional complexities of intermediate nodes.
- 3) Depending on how feedback information notifies source nodes about network load conditions, current congestion control schemes can be classified into explicit notification based versus implicit notification based. In explicit notification based congestion control schemes, source nodes are instructed by incoming feedback messages either to start/stop transmission or to increase/decrease their transmission rates. In implicit notification based congestion control schemes, traffic source nodes can only infer network congestion through experiencing a sharp increase of round-trip delays or detecting packet losses. Like end-to-end based and credit based congestion control schemes, implicit notification based congestion control schemes trade the effectiveness of congestion control for the functional simplicity of congestion control.

As stated before, functionality in the IP layer has been reduced as much as possible for robustness considerations. Therefore, TCP adopts an end-to-end, implicit credit based

approach to congestion control. The next section reviews existing TCP congestion control mechanisms.

3.2 TCP Congestion Control

3.2.1 Fundamentals of TCP Congestion Control Mechanism

The specific credit that TCP sources use to regulate their emitted traffic is called “congestion window”. Each TCP source maintains a congestion window of size *cwnd* as a measure of the network capacity for that TCP connection, and *cwnd* limits the number of packets that a TCP source can send to the network without receiving further acknowledgments. The congestion window size *cwnd* varies dynamically in response to packet acknowledgments and detection of packet losses. Depending on how *cwnd* changes in response to acknowledgments, the generic TCP congestion control algorithm consists of *slow start* and *congestion avoidance* phases [15]. A parameter called slow start threshold *ssthresh* is maintained at TCP sources to distinguish between the two phases. TCP sources start transmission in the *slow start* phase by sending one packet[†]. When a TCP source receives an acknowledgment from its corresponding node asking for new data, the source increases *cwnd* by one. The *slow start* phase continues until the congestion window *cwnd* reaches the slow start threshold *ssthresh*, and then the *congestion avoidance* phase begins. During the *congestion avoidance* phase, the source increases its window size by $1/cwnd$ every time a packet is acknowledged. After a whole window of packets are acknowledged, the window size *cwnd* is increased by one. The basic TCP congestion control algorithm updates congestion window size *cwnd* and the slow start threshold *ssthresh* as follows:

if a new acknowledgment is received

if $cwnd < ssthresh$

$cwnd = cwnd + 1$

else

[†] Strictly, the Data Protocol Unit (DPU) in transport layer is called a segment.

```
         $cwnd = cwnd + 1 / cwnd$   
  
    end if  
  
    else if a packet loss is detected  
  
        retransmit the lost packet  
  
         $ssthresh = cwnd / 2$   
  
         $cwnd = 1$   
  
    end if
```

To detect packet losses, different versions of TCP use different techniques. In the original version of TCP (Old Tahoe), packet loss is detected by maintaining a timer based on an estimate of the connection's Round-Trip Time (RTT). When a packet is sent, the timer is started. If no acknowledgment is received before the timer expires, the packet is assumed to be lost. Besides the timeout mechanism, the later versions of TCP, namely Tahoe and Reno, adopt the *fast retransmission* technique to speed up packet loss detection, furthermore TCP Reno also uses the *fast recovery* technique to quickly restore congestion window size [16].

3.2.2 TCP Tahoe

The granularity of the retransmit timer in TCP (Old Tahoe) is very coarse, and an exponential backoff algorithm is used to set the timeout value [17]. As a consequence, a large amount of time is wasted in waiting for a timeout before lost packets can be retransmitted. TCP Tahoe introduced a mechanism called *fast retransmission* to speed up the retransmission process. The *fast retransmission* mechanism relies on receiving a number of duplicate acknowledgments to trigger retransmission without waiting for the expiration of the retransmission timer. In TCP specifications, the sequence number included in an acknowledgment is the index of the segment that the receiver is expecting to receive next, so a TCP receiver would send back duplicate acknowledgments if it received out of order segments.

Once the number of duplicate acknowledgments a TCP Tahoe source receives, exceeds a threshold known as $DupAck_{th}$, it will retransmit the lost packet immediately, followed by shrinking congestion window to one. Choosing an appropriate value for $DupAck_{th}$ must consider the following two factors: 1) Packet reordering can happen in IP-based networks, therefore TCP senders may receive duplicate acknowledgments even though no packet loss occurs. 2) If a TCP source has a small congestion window size, there may not be enough duplicate acknowledgments generated by its receiver. In current implementations of TCP, the value of $DupAck_{th}$ is set to 3. TCP Tahoe[†] with *fast retransmission* mechanism was first implemented in 4.3 BSD Unix in 1988 [16].

TCP Tahoe congestion control algorithm updates congestion window size $cwnd$ and the slow start threshold $ssthresh$ as follows:

```

if a new acknowledgment is received

    if  $cwnd < ssthresh$ 

         $cwnd = cwnd + 1$ 

    else

         $cwnd = cwnd + 1 / cwnd$ 

    end if

else if a duplicate acknowledgment is received

     $N_{DupAck} = N_{DupAck} + 1$ 

    if  $N_{DupAck} > DupAck_{th}$ 

        retransmit the lost packet

         $ssthresh = cwnd / 2$ 

         $cwnd = 1$ 

    end if

```

[†] Tahoe is actually the release name of the 4.3BSD distributed in 1988

else if the retransmission timer expires

retransmit the lost packet

$ssthresh = cwnd / 2$

$cwnd = 1$

end if

3.2.3 TCP Reno

Besides retaining *slow start*, *congestion avoidance* and *fast retransmission* mechanisms of TCP Tahoe, TCP Reno adopted a new technique called *fast recovery* to quickly restore congestion window size.

In TCP Tahoe, after receiving a number of duplicate acknowledgments, the sender performs *fast retransmission*, reduces the congestion window size to one segment and enters the *slow start* window growing phase. However, the fact that duplicate acknowledgments can get through means that the network is not severely congested. Therefore, cutting congestion window size to one segment after *fast retransmission* is a very conservative approach.

The *fast recovery* mechanism sets congestion window size to $cwnd / 2 + DupAck_{th}$ and forces the congestion window growing process to enter into a *congestion avoidance* phase when a packet loss is detected by duplicate acknowledgments. Hence *fast recovery* technique prevents the “packet pipe” from going empty after *fast retransmission* and restore congestion window size without going through the *slow start* phase. TCP Reno[†] with *fast recovery* mechanism was first implemented in 4.3 BSD Unix in 1990 [16].

TCP Reno congestion control algorithm updates congestion window size $cwnd$ and the slow start threshold $ssthresh$ as follows:

if a new acknowledgment is received

[†] Reno is actually the release name of the 4.3BSD distributed in 1990

```

if  $cwnd < ssthresh$ 
     $cwnd = cwnd + 1$ 
else
     $cwnd = cwnd + 1 / cwnd$ 
end if

else if a duplicate acknowledgment is received
     $N_{DupAck} = N_{DupAck} + 1$ 
    if  $N_{DupAck} > DupAck_{th}$ 
        retransmit the lost packet
         $ssthresh = cwnd / 2$ 
         $cwnd = ssthresh + DupAck_{th}$ 
    end if
else if the retransmission timer expires
        retransmit the lost packet
         $ssthresh = cwnd / 2$ 
         $cwnd = 1$ 
    end if

```

3.2.4 TCP Vegas and Other TCP Flavours

Clearly both TCP Tahoe and TCP Reno adopt a reactive strategy to congestion control. By the time TCP Tahoe and TCP Reno cut their congestion window, packet losses have already occurred.

TCP Vegas tries to reduce packet losses by taking a proactive approach to congestion control. It calculates a parameter called *actual throughput* by counting the number of bytes

transmitted every RTT. The difference δ between *expected throughput* and *actual throughput* is compared with two parameters α and β ($0 < \alpha < \beta$). If the difference is less than α , TCP Vegas increases the congestion window by $1 / cwnd$. If the difference is greater than β , TCP Vegas decreases the congestion window by one, and if the difference lies between α and β , TCP Vegas leaves the congestion window unchanged [18]. During *congestion avoidance* phase, TCP Vegas congestion control algorithm updates congestion window size $cwnd$ as follows:

if $\delta > \beta$

$cwnd = cwnd - 1$

else if $\delta < \alpha$

$cwnd = cwnd + 1 / cwnd$

else

$cwnd$ unchanged

end if

The root of the proactive congestion control scheme adopted by TCP Vegas can be traced back to Wang and Crowcroft's Tri-S [19] and Jain's CARD (Congestion Avoidance using Round-trip Delay) [20] proposals. In the Tri-S scheme, the window size is increased by one segment every RTT, and the throughput is compared with the throughput when the window is one segment smaller. If the difference is less than a threshold, the window size is decreased by one segment in the next RTT. In Jain's CARD proposal, the window size is adjusted once every two RTTs based on the product $(Cwnd_{current} - Cwnd_{old}) \cdot (RTT_{current} - RTT_{old})$ as follows: if the result is positive, decrease the window size by one-eighth. If the result is negative or zero, increase the window size by one segment.

Besides TCP Tahoe, TCP Reno and TCP Vegas, dozens of other TCP flavours have been proposed, including TCP New-Reno [21], which tries to overcome the deficiency of TCP Reno when multiple losses occur in a single window.

Currently most TCP implementations adopt the Reno-style congestion control scheme. Despite other TCP variations claiming to achieve better performance than TCP Reno, they mainly exist in research communities.

3.3 Limitations of Existing TCP Congestion Control Schemes

From the congestion control schemes described in Section 3.2, it can be seen that TCP Tahoe and TCP Reno keep increasing their congestion window sizes until buffer overflows occur over the intermediate nodes. On experiencing packet losses, TCP Tahoe and TCP Reno cut their congestion windows back immediately to mitigate the congestion. When TCP Tahoe and TCP Reno start to shrink their congestion windows to reduce the network load, congestion has already occurred. TCP Tahoe and TCP Reno constantly cause buffer overflows in their processes to probe the available bandwidth.

Due to the reactive nature of TCP Tahoe and TCP Reno congestion control schemes, large buffer space must be allocated in intermediate nodes, so that the bottleneck link will not be starved immediately after these congestion window reductions. However, buffering a large amount of packets in intermediate nodes is not desirable, as it increases queuing delays and it also exaggerates the handover overhead in mobile data networks.

Another problem induced by the reactive congestion control mechanisms of TCP Tahoe and TCP Reno is that packet losses are interpreted as indications of network congestion. When a TCP source node detects packet losses, it must shrink its congestion window to slow down its transmission. This kind of congestion control scheme works very well in wired networking environments, where buffer overflows are the primary cause of packet losses. However, in situations where a TCP connection comprising unreliable transmission media such as wireless links, packet losses caused by transmission errors cannot be avoided. TCP source nodes cannot distinguish whether packet losses are caused by congestion or corruption. As a result, TCP source nodes misinterpret transmission error caused packet losses as indications of network congestion and thereby reduce their congestion window sizes, which leads to poor throughput of TCP connections over lossy links. As wireless links are becoming a fast

growing part in the global internetworking infrastructure, improving TCP performance over wireless links becomes an important issue.

Besides the problems mentioned above, the current TCP congestion window updating mechanisms are heuristic based. For example, during the *congestion avoidance* phase, when a congestion window needs to be increased, Tri-S, TCP Tahoe, Reno and Vegas increase the congestion window size by one segment every RTT, and the CARD method increases the congestion window size by one segment every two RTTs. There are no essential reasons why TCP should not increase its congestion window size by more than one segment every RTT. In fact, when the bandwidth-delay product of a TCP connection is large and the network is under-utilised, increasing the congestion window size by more than one segment every RTT can help the TCP connection reach its maximum throughput more quickly.

To overcome the shortfalls associated with existing TCP congestion control schemes, a novel Self-Tuning Proportional and Derivative control based TCP (ST-PD TCP) congestion control scheme is proposed in Section 3.4, which addresses the congestion control problem from a control-theoretic perspective. In ST-PD TCP, the congestion control routine is carried out by a conventional PD controller; a fuzzy logic controller is employed as a supervisory controller to tune the gains of the underlying PD controller online. The PD controller improves TCP congestion control performance by keeping the buffer occupancy of the TCP connection's bottleneck node around an ideal operating point, thereby ST-PD TCP is able to probe available bandwidth without causing buffer overflows. A number of benefits result directly from this. Firstly the performance of ST-PD TCP is insensitive to the dimensions of buffers at intermediate nodes. Secondly ST-PD TCP congestion control scheme no longer relies on the detection of packet losses to adjust its congestion window size. This feature greatly improves the performance of ST-PD TCP over wireless links. Finally, compared with the heuristic congestion control mechanisms used in TCP Tahoe, TCP Reno and TCP Vegas, the PD-based congestion control scheme can respond more rapidly and accurately to bandwidth variations.

It is worth a mention that a number of PD based and fuzzy logic based congestion control schemes for ATM networks have appeared in recent literature. In [22], an explicit rate based

congestion control scheme for ATM ABR service is proposed. In this scheme, ATM switches employ PD controllers to compute each flow's fair share of bandwidth. The allocated bandwidth information is explicitly fed back to source nodes, which then adjust their transmission rates accordingly. Another congestion control framework for ATM networks is described in [23], which uses fuzzy logic controllers at switches to compute the explicit rate for each flow. However the following differences distinguish the ST-PD TCP congestion control scheme from existing PD based or fuzzy logic based congestion control schemes proposed for ATM networks:

- 1) the ST-PD TCP congestion control operates on a purely end-to-end basis without any involvement of intermediate nodes, while in the ATM congestion control paradigm, end nodes as well as intermediate nodes participate in the congestion control;
- 2) in the ST-PD TCP congestion control scheme, the major control entity resides at source nodes where the networking resource information, such as the bandwidth of each link, the buffer size of each node and the number of competing connections, is unavailable, while in the ATM congestion control paradigm [22][23], the major control entity resides at intermediate nodes, which have explicit knowledge about the bandwidth, buffer size and the number of active flows;
- 3) in the ST-PD TCP congestion control scheme, the only feedback information is acknowledgments of received data, while in the ATM congestion control paradigm, the explicit rates allocated to each flow are conveyed to source nodes.

To the best of our knowledge, applying a PD control structure to end-to-end TCP congestion control has never been proposed before, and will have practical relevance to the networking environments currently dominated by TCP/IP.

3.4 A Self-Tuning PD Based TCP Congestion Control Scheme

3.4.1 Main Features of ST-PD TCP

The following innovative features and important mechanisms are introduced into the new TCP congestion control scheme for performance enhancements:

- A dual-controller structure forms the base of ST-PD TCP congestion control: a PD controller performs the routine congestion control function and a fuzzy logic controller is used as a supervisory controller to adjust the gains of the underlying classic PD controller online. The self-tuning PD based congestion control scheme leads to a faster and more accurate congestion control result.
- In ST-PD congestion control schemes, the functions of congestion control and error control are decoupled. This mechanism ensures that TCP users would not experience unnecessary performance deterioration over error-prone links. The behaviour of ST-PD TCP over wireless links is investigated in Chapter 4.

In addition to performance enhancements, the following measures are taken to ensure its deployability when designing ST-PD TCP:

- No additional information is required from the network layer (unlike TCP ECN [24]). The only information that ST-PD TCP requires to perform congestion control is acknowledgments sent back from any legitimate current versions of TCP.
- Existing TCP implementations can communicate with ST-PD TCP without any modification.
- The computational complexity of ST-PD TCP congestion control algorithm is very low.

3.4.2 Formulation

Consider an arbitrary IP-based network in which a TCP connection S-D traverses a set of routers $R_1, R_2, \dots, R_n$ as shown in Figure 3.1. Each link between adjacent routers is shared by other TCP connections. Existing TCP connections can leave and new TCP connections can join in at anytime without being subjected to connection admission control (CAC) or resource reservation. Along its path, connection S-D has only one bottleneck link (if more than one link has the same lowest speed, the bottleneck link is the one closest to the source node). The service discipline of the bottleneck link is First-In First-Out (FIFO). Each router has a finite buffer and arriving packets are simply dropped when the buffer is full.

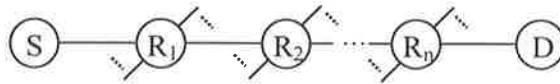


Fig. 3.1 A TCP connection in an arbitrary network

Assume the link between routers R_j and R_{j+1} is the bottleneck link for the TCP connection S-D. If the packet arrival rate is larger than the packet departure rate at router R_j , a queue will build up and eventually arriving packets will be dropped due to buffer overflow. On the other hand, if the packet arrival rate is smaller than the packet departure rate at router R_j , the existing queue will be emptied and the bottleneck link is at risk of under-utilisation.

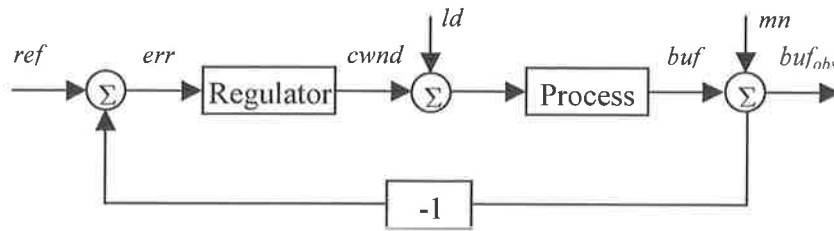


Fig. 3.2 Closed-loop control diagram

To fully utilise the bottleneck link without causing buffer overflows, a regulator is needed at every TCP source node to adjust its congestion window size so that the buffer occupancy at

the bottleneck node can stay around an ideal level. The closed-loop control diagram of the regulator and the process that the regulator tries to control is shown in Figure 3.2.

In Figure 3.2, the regulator is the congestion control entity that resides in TCP source node S. The regulator output $cwnd$ is TCP congestion window size. The load disturbance ld is contributed by:

- 1) other connections change their transmission rates,
- 2) new connections join in, and
- 3) some existing connections are terminated.

The buffer occupancy dynamics of the bottleneck node is chosen as the process that the regulator tries to control. The process output buf is the buffer occupancy. Since the source node cannot measure the bottleneck node's buffer occupancy directly, the difference between the observed buffer occupancy buf_{obs} and the actual buffer occupancy buf at the bottleneck is the measurement noise mn , ie., $buf_{obs} = buf + mn$. The reference point ref is the desired buffer occupancy. The regulator adjusts congestion window size $cwnd$ so that the observed buffer occupancy buf_{obs} can converge to the reference point ref . The difference between the desired buffer occupancy ref and the observed buffer occupancy buf_{obs} forms the control error err , which is fed into the regulator's input.

3.4.3 Design Aspects of ST-PD TCP

Classical model-referenced control methods require that underlying processes be modelled mathematically. The buffer occupancy dynamics can be expressed as,

$$buf(n) - buf(n-1) = \max\{(\sum_i r_i(n) - S(n)), 0\} \quad (3.1)$$

where buf is the buffer occupancy, r_i is packet arrival rate at the bottleneck link from connection i and S is the service rate of bottleneck link.

However, from the regulator's point of view, the information expressed in equation (3.1) is totally unavailable, since the regulator, which resides at an end node does not know what the service rate of the bottleneck link is, how many connections share the bottleneck link and what their arrival rates are. Therefore, conventional model-referenced control schemes cannot be applied here, and that is the fundamental difference between ST-PD TCP congestion control scheme and other control-theoretic based congestion control schemes such as the ones described in [22][23], which are primarily applied in ATM networks.

On the other hand, it has been observed that, in control engineering practice, PID controllers have been successfully applied in many cases where the underlying process is unknown or is too complicated to be modelled mathematically [25]. The strength of PID controller is that there are only three gains to be tuned and how the three gains affect the control performance such as rise time, overshoot and settle time is well known [26]. Thus a human operator can tune a PID controller online based on experience. The ST-PD TCP congestion control scheme described in this chapter is based on the supervised PID control concept. Two adaptations have been made in order to meet the special requirements as a congestion control entity in a transport layer protocol:

- 1) The human operator is replaced by a fuzzy logic supervisory controller. Since congestion control should be carried out automatically without any human involvement, the controller that performs congestion control must have a self-tuning property. A major feature of fuzzy logic is that it is linguistic based, which makes fuzzy logic controllers ideal candidates to replace human operators of PID controllers. It is reasonably straightforward to transform the knowledge base of an experienced human operator into the linguistic rule base of a fuzzy logic controller.
- 2) A PD instead of PID control structure is adopted. There are two reasons for this change. Firstly, the integral part in a PID controller is used mainly to eliminate the residual control error when the system reaches a steady state. However, in a real network, a large number of connections can join or leave the bottleneck link at anytime, hence the system may never reach a steady state. Therefore in the congestion control application considered here, eliminating residual control error is

not an objective. Secondly, the congestion control process must be carried out in real time. Removing the integral part of the PID controller will reduce the computational complexity.

The detailed structure of the regulator in Figure 3.2 is shown in Figure 3.3. The input i to the fuzzy logic supervisory controller is the regulator perceived bandwidth-RTT product. The output of the fuzzy logic supervisory controller is the gains K_p and K_d of the underlying PD controller.

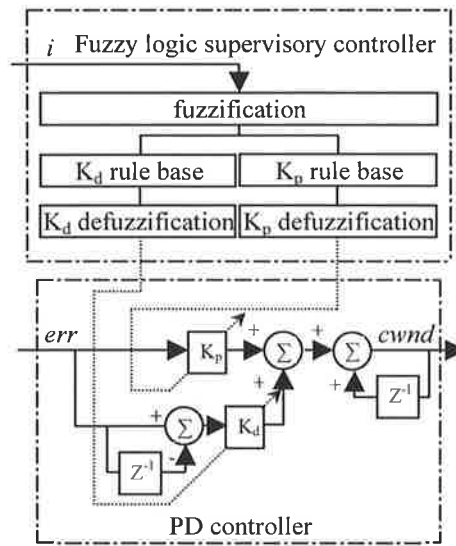


Fig. 3.3 ST-PD control structure

3.4.3.1 Design of PD Controller

The input to the PD controller shown in Figure 3.3 is the control error err , which is the difference between the desired buffer occupancy ref of the bottleneck node and the observed buffer occupancy buf_{obs} .

$$err(n) = ref - buf_{obs}(n) \quad (3.2)$$

The observed buffer occupancy buf_{obs} can be expressed as,

$$buf_{obs}(n) = (RTT(n) - RTT_{min}) \cdot S(n) \quad (3.3)$$

where $RTT(n)$ is the current round-trip time and RTT_{min} is the minimum round-trip time experienced so far. The perceived service rate is,

$$S(n) = \frac{\text{the number of packets transmitted during } RTT(n)}{RTT(n)} \quad (3.4)$$

To avoid starving the bottleneck link and also to avoid excessive queuing delay, we choose to set the desired buffer occupancy ref to 1 packet.

After computing the control error err , the PD controller regulates the congestion window $cwnd$ as follows.

$$cwnd(n) = cwnd(n-1) + K_p \cdot err(n) + K_d \cdot (err(n) - err(n-1)) \quad (3.5)$$

TCP connections with different bandwidth-RTT products require different gains K_p and K_d in order to optimise the performance of the PD controller. Before designing the fuzzy logic supervisory controller, a range of values for K_p and K_d corresponding to TCP connections with different bandwidth-RTT products, are established through a series of Ziegler-Nichols frequency response tuning and hand-tuning procedures [26]. These are transformed into two sets of fuzzy logic rule bases, on which the fuzzy logic inference engine is based. This is done once only in the design stage.

3.4.3.2 Design of Fuzzy Logic Supervisory Controller

The input i to the fuzzy logic supervisory controller shown in Figure 3.3 is the regulator perceived bandwidth-RTT product, which is the amount of data transmitted during the current round-trip time. The input i is used as a linguistic variable, which is fuzzified into five linguistic terms: ES(Extremely Small), S(Small), M(Medium), L(Large) and EL(Extremely Large). Figure 3.4 shows the fuzzification membership function μ .

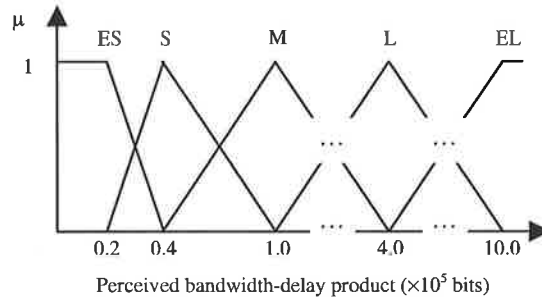


Fig. 3.4 Fuzzification membership function

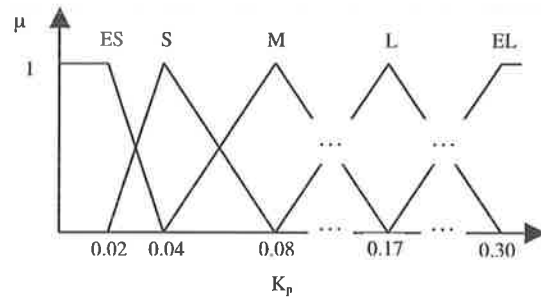
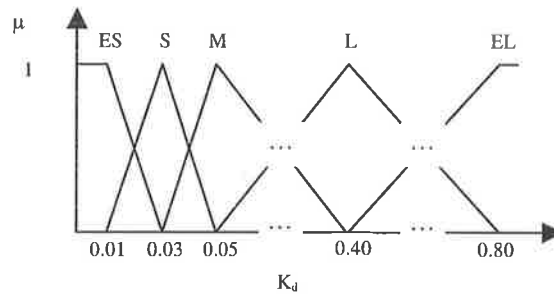
The rule base for K_p is,

- If bandwidth-delay product is ES, then K_p is EL.
- If bandwidth-delay product is S, then K_p is L.
- If bandwidth-delay product is M, then K_p is M.
- If bandwidth-delay product is L, then K_p is S.
- If bandwidth-delay product is EL, then K_p is ES.

The rule base for K_d is,

- If bandwidth-delay product is ES, then K_d is ES.
- If bandwidth-delay product is S, then K_d is S.
- If bandwidth-delay product is M, then K_d is M.
- If bandwidth-delay product is L, then K_d is L.
- If bandwidth-delay product is EL, then K_d is EL.

Figure 3.5(a) and (b) show defuzzification functions for K_p and K_d respectively.

Fig. 3.5(a) Defuzzification function for K_p Fig. 3.5(b) Defuzzification function for K_d

CoM (Centre-of-Maximum) [27] is used as the defuzzification method.

3.4.3.3 Flow-Chart of ST-PD TCP Congestion Control Algorithm

The flow-chart of ST-PD TCP congestion control algorithm is shown in Figure 3.6.

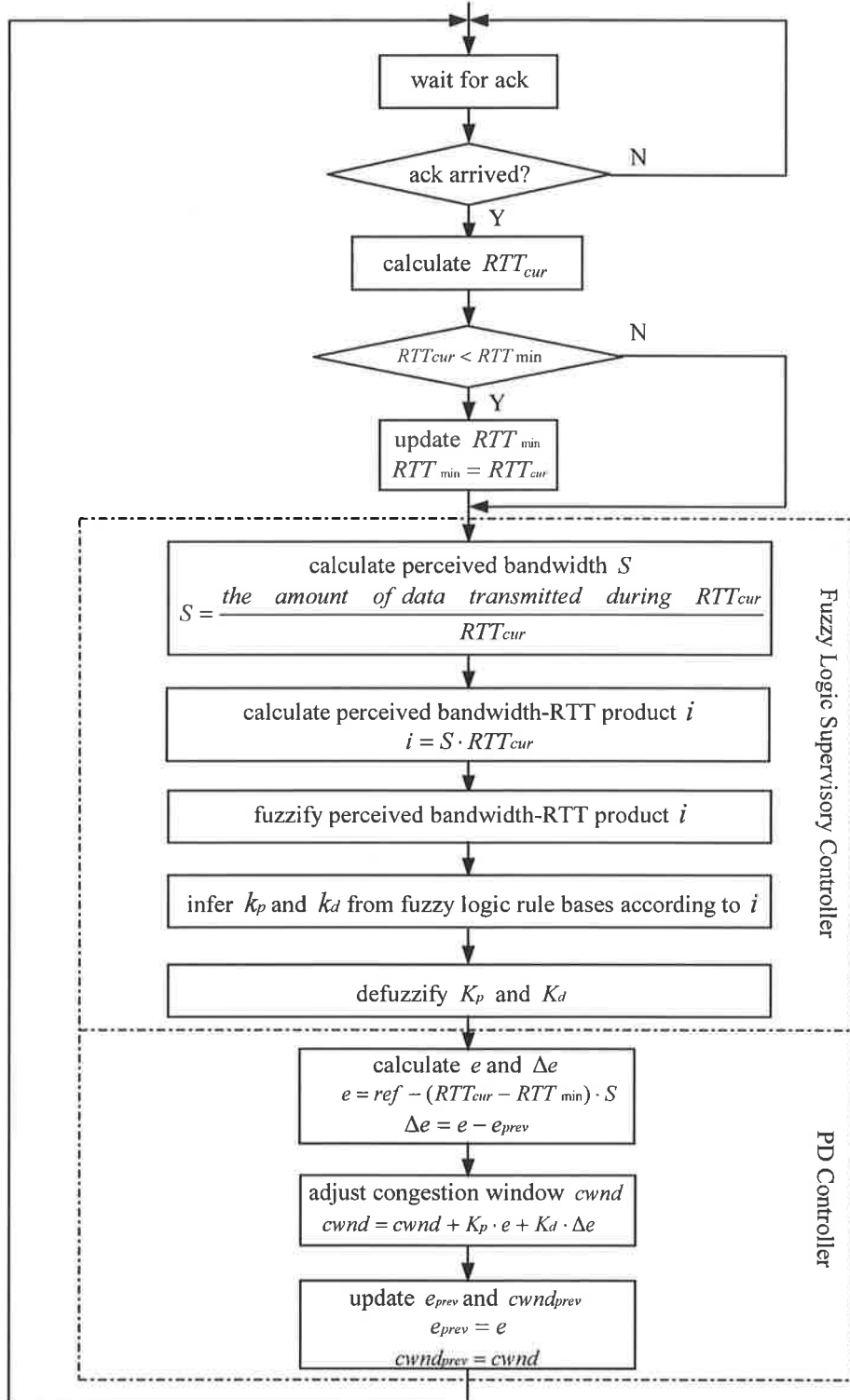


Fig. 3.6 Flow-chart of ST-PD TCP congestion control algorithm

3.4.3.4 Stability Discussions of ST-PD TCP Congestion Control Scheme

Presently there is no systematic procedure for the design and stability analysis of fuzzy logic controllers. The design of a fuzzy logic controller is, to a large extent, dependent on experience. Optimisation and stability proof of a fuzzy logic controller is primarily obtained through a series of trial-and-error procedures. Despite the fact that a substantial amount of research activities have been devoted to the stability theory of fuzzy logic controllers, the analysis methods established so far are limited to a very narrow range of systems, which are usually simple or special control cases created for pure academic research purposes rather than originated from the real world. Therefore, current stability theory of fuzzy logic controllers has little practical relevance.

The problem related to stability analysis also exists for conventional control systems. In order to perform stability analysis for a conventional control system, the mathematical model of the process under control must be derived, and the reliability of stability analysis depends on the accuracy of the mathematical model. However, in control engineering practices, most processes cannot be accurately described in mathematical expressions or have significant uncertainties that cannot be modelled, and an inaccurate model does not suffice for a convincing stability proof. Hence, in most engineering applications, stability analysis plays a minor role, regardless of whether these applications use fuzzy logic or conventional control techniques [27]. A stability proof must rely on final tests.

Finally, considering the specific application here — TCP congestion control, a stability requirement is not essential. Existing TCP congestion control schemes, such as TCP Tahoe and TCP Reno, are inherently unstable, ie., their congestion window growing processes exhibit an oscillation pattern, and the buffer of the bottleneck node is periodically overflowed under the congestion control of existing TCP implementations. ST-PD TCP aims to improve this by confining the buffer occupancy oscillation into a narrow range. Forcing a bottleneck node's buffer occupancy to be stabilised at a particular point is not an objective, and probably an unrealistic goal in a decentralised and uncoordinated networking environment such as the Internet.

3.5 Analysis and Simulation Results

The ST-PD TCP is implemented on the infrastructure of an open source network simulation software *ns-2* [28]. TCP Tahoe, TCP Reno, TCP Vegas and other versions of TCP are included in the release of *ns-2*. Simulation results are presented in this section to demonstrate the general behaviour of ST-PD TCP over a range of wired networking environments.

3.5.1 A Simple TCP Connection Case

Consider a TCP connection SD consisting of links SR and RD as shown in Figure 3.7. The transmission speed and propagation delay of link SR are 10Mbps and 0.2ms. The transmission speed and propagation delay of link RD are 1.5Mbps and 20ms. Source node S transfers an infinite large file to destination node D via node R. The size of transmitted packets is 1000 bytes. Clearly without a congestion control mechanism, source node S would transmit data as fast as its outgoing link allows. This would lead to node R being flooded by incoming packets. The TCP running at the node S is responsible for regulating the outgoing traffic so that the bottleneck node R is not overwhelmed while the bandwidth of the link RD is fully utilised.

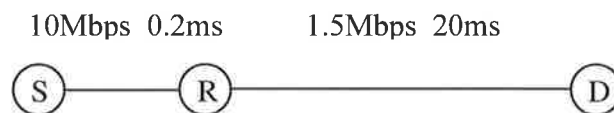


Fig. 3.7 A simple TCP connection

In the following simulations, source node S uses TCP Reno and ST-PD TCP respectively to transfer the file for the first 30 seconds. Comparison results of the performance of the two TCP implementations are presented. Since the behaviour of TCP Reno is affected by the bottleneck node's buffer dimension, simulations were carried out under the conditions that node R had either sufficient or insufficient buffer space.

The objectives of the simulations described in this sub-section are to investigate:

- 1) how accurate ST-PD TCP is able to estimate the available bandwidth, and thereby set its congestion window accordingly;
- 2) how effective ST-PD TCP is able to keep the bottleneck node's queue length around the designed level;
- 3) the impact of the bottleneck node's buffer dimension on TCP performance;
- 4) the throughput achieved by the two TCP implementations.

Recall that TCP Reno constantly overflows the bottleneck node's buffer. After experiencing a buffer overflow, TCP Reno cuts its congestion window by half when packet losses are detected by receiving a number of duplicate acknowledgments. If there are enough packets buffered at the bottleneck node to avoid the starvation of the bottleneck link after the congestion window reduction, then the buffer dimension of the bottleneck node can be regarded as sufficient. Assume the bandwidth- $RTT_{min}^{\dagger}$ product of a TCP connection can hold N packets, and its bottleneck node can buffer a further B packets. The maximal congestion window size that the TCP Reno source can reach without causing a buffer overflow is $N + B$. After the congestion window size exceeds $N + B$, packet losses occur due to buffer overflows, and the TCP Reno source reduces its congestion window size by half. To avoid starving the bottleneck link after the congestion window is cut, the halved congestion window size must be no less than the bandwidth- RTT_{min} product of the TCP connection, ie. it must satisfy the inequality $(N + B)/2 \geq N$. As a rule of thumb, the bottleneck node's buffer space is regarded as sufficient if it is no less than the connection's bandwidth- RTT_{min} product. Otherwise it is regarded as insufficient.

Since the bandwidth- RTT_{min} product of the TCP connection shown in Figure 3.7 can hold approximately 9 packets of the size of 1000 bytes, the buffer dimension of node R is set to be able to hold 14 and 4 packets in section 3.5.1.1 and section 3.5.1.2 respectively.

[†] RTT_{min} is the connection's minimal round-trip time. It is the sum of time of transferring a full-sized packet from a source node to a destination node without experiencing queuing delay and transferring its acknowledgment from the destination node to the source node without experiencing queuing delay.

3.5.1.1 The Behaviour of TCP under Sufficient Buffer Conditions

Figure 3.8(a) and (b) show the congestion window[†] traces of TCP Reno and ST-PD TCP respectively.

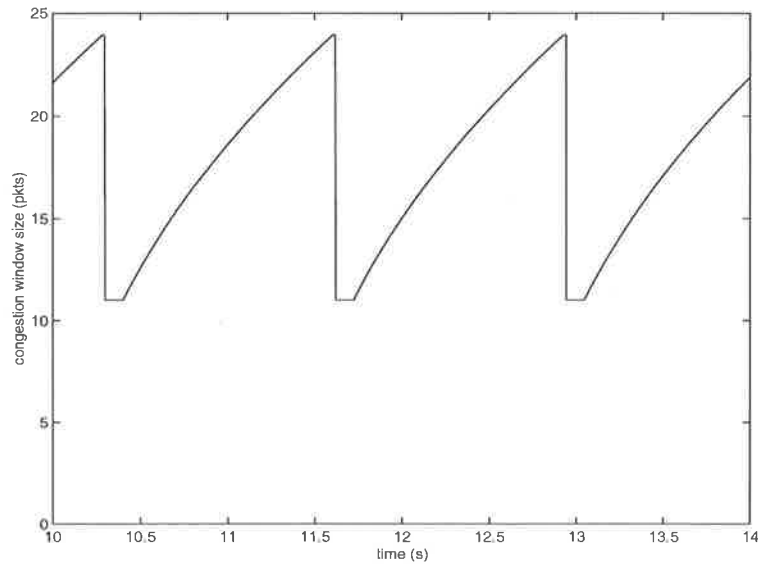


Fig. 3.8(a) Congestion window trace of TCP Reno under a sufficient buffer condition

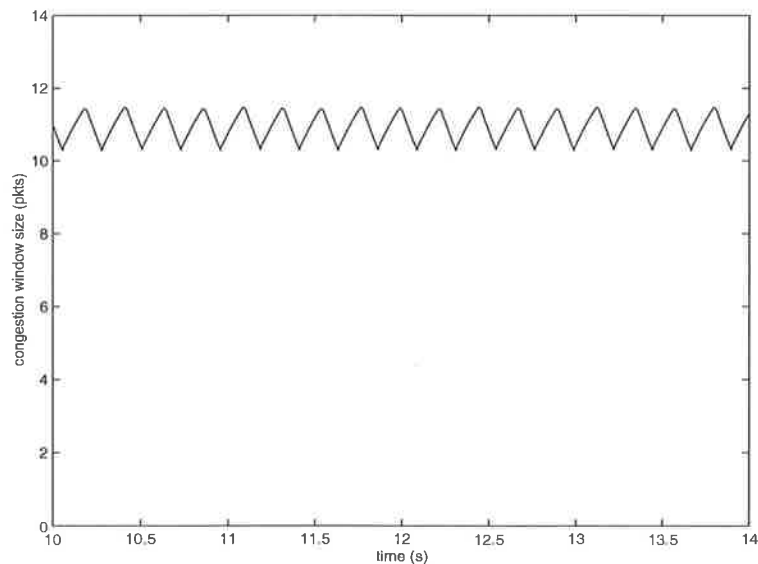


Fig. 3.8(b) Congestion window trace of ST-PD TCP under a sufficient buffer condition

[†] In ns-2, the congestion window variable (cwnd_) is declared as double. It is converted to an integer only when it is used to decide whether a TCP source is allowed to send out more packets.

Since the bandwidth-RTT_{min} product of the connection can hold about 9 packets and node R can buffer 14 packets, the maximum congestion window that can be set without causing buffer overflows is 23. From Figure 3.8(a), it can be seen that TCP Reno keeps increasing its congestion window size until the congestion window size exceeds 23, after which the buffer at the node R overflows. As a result several incoming packets are dropped by the congested node. When the packet losses are detected through receiving a number of duplicate acknowledgments, TCP Reno cuts its congestion window by half, and starts to increase its congestion window again. The periodical pattern of buffer overflows under the congestion control of TCP Reno is clearly shown in Figure 3.8(a).

One of the design objectives of ST-PD TCP is to keep the bottleneck node's buffer occupancy around one. The bandwidth-RTT_{min} product of the connection can hold approximately 9 packets, plus one in the buffer of node R, therefore the congestion window size set by ST-PD TCP should hover between the range of 10-12 packets. This feature is confirmed in Figure 3.8(b).

Figure 3.9(a) and (b) show the buffer occupancy traces of node R when the source node S uses TCP Reno and ST-PD TCP respectively. Under the congestion control of TCP Reno, the queue length of node R oscillates from 2 packets to the maximum buffer space it can use, as shown in Figure 3.9(a); whereas ST-PD TCP effectively keeps node R's buffer occupancy around one packet level.

Note that the burrs of the traces appearing in Figure 3.9(a) (b) and Figure 3.12(a) (b) in section 3.5.1.2 are due to the fact that the router R is not a cut-through router, ie. a packet must be received completely before it can be forwarded to an outgoing link. The sample interval in the Figure 3.9(a) (b) and Figure 3.12(a) (b) is 0.002 second.

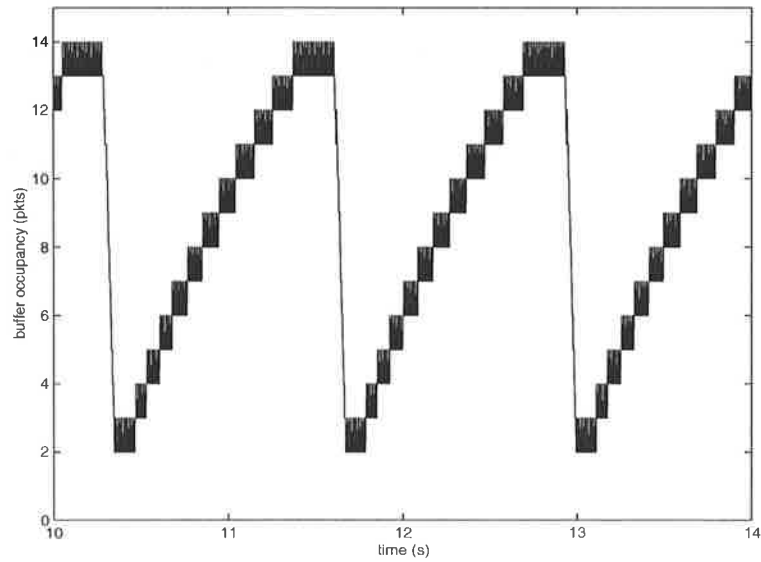


Fig. 3.9(a) Buffer occupancy of node R under the congestion control of TCP Reno under a sufficient buffer condition

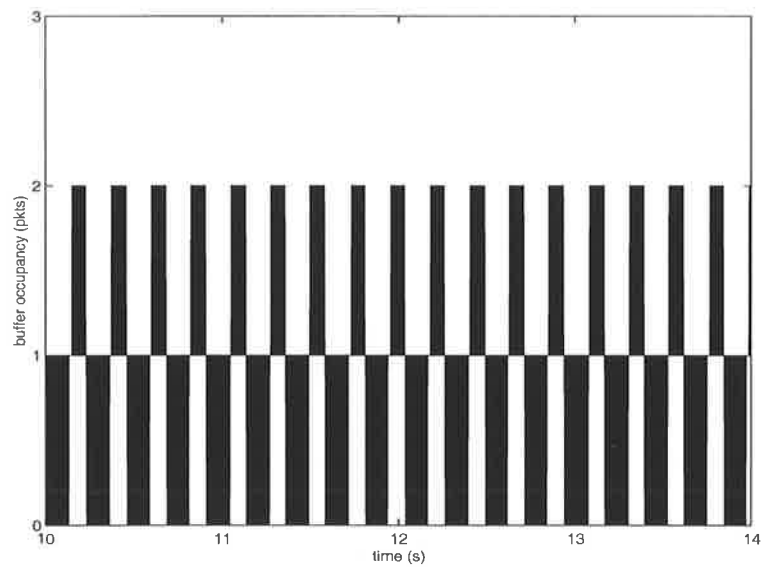


Fig. 3.9(b) Buffer occupancy of node R under the congestion control of ST-PD TCP under a sufficient buffer condition

The result of throughput comparison of TCP Reno and ST-PD TCP is shown in Figure 3.10. Throughput is measured as the number of bits transmitted over a given time. It is used to indicate how efficient the bottleneck link of a TCP connection is utilised. Since the bottleneck link speed of the TCP connection is 1.5Mbps, the maximum throughput of the

TCP connection can reach is $1.5\text{Mbps}^\ddagger$ if the bottleneck link can be prevented from starving. From Figure 3.9(a) and (b), it can be seen that there are always packets buffered in the bottleneck node R, which means the bottleneck link is kept busy, therefore both Reno and ST-PD TCP connections can reach the maximum throughput on a long run as indicated in Figure 3.10.

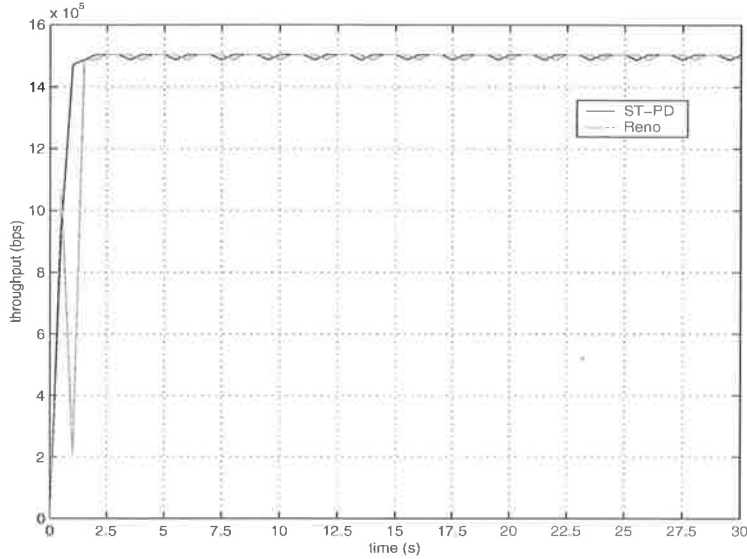


Fig. 3.10 Throughput of TCP Reno and ST-PD TCP under a sufficient buffer condition

The sample interval in the Figure 3.10 and Figure 3.14 in section 3.5.1.2 is 0.5 second.

3.5.1.2 The Behaviour of TCP under Insufficient Buffer Conditions

Figure 3.11(a) and (b) show the congestion window traces of TCP Reno and ST-PD TCP respectively. As stated before, the bandwidth- $\text{RTT}_{\min}$ product of the connection can hold about 9 packets, and node R can buffer 4 packets. Therefore after the congestion window size exceeds 13, packet losses occur due to buffer overflows, and TCP Reno is forced to cut its congestion window by half. However under the insufficient buffer condition, the halved congestion window of TCP Reno is not large enough to fill the bandwidth- $\text{RTT}_{\min}$ product of

[‡] Payload and protocol overhead are taken into account when calculating throughput.

the connection. Thereby the bottleneck link stays idle for some time immediately after these congestion window reductions.

Figure 3.11(b) shows that the buffer dimension of node R has no effect on the congestion window of ST-PD TCP.

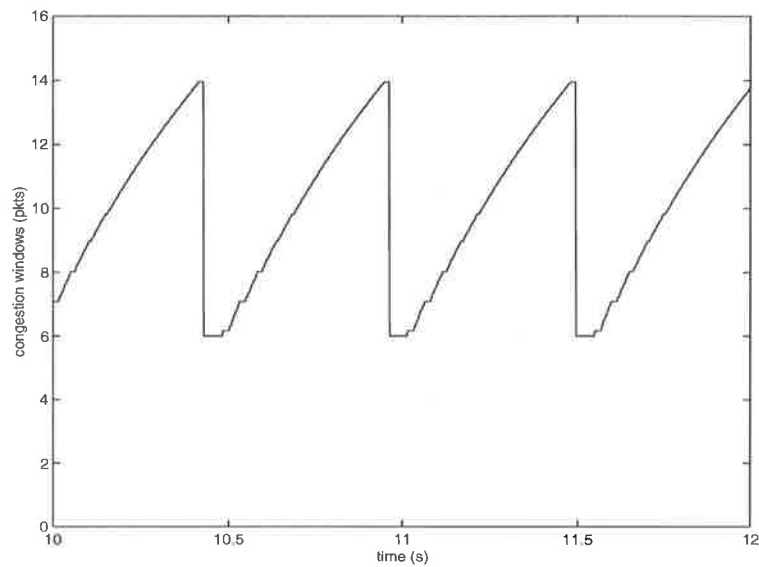


Fig. 3.11(a) Congestion window trace of TCP Reno under an insufficient buffer condition

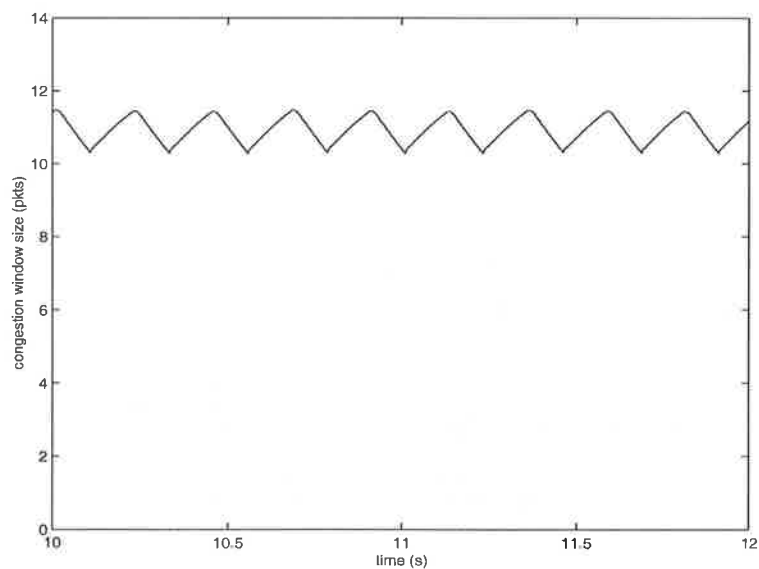


Fig. 3.11(b) Congestion window trace of ST-PD TCP under an insufficient buffer condition

Figure 3.12(a) shows the buffer occupancy dynamics of node R under the congestion control of TCP Reno. Unlike the case where node R has sufficient buffer space as shown in Figure 3.9(a), cutting the congestion window by half in an insufficient buffer condition leads to the buffer of node R being in an empty state for some time. During that period, the bandwidth of the bottleneck link is simply wasted.

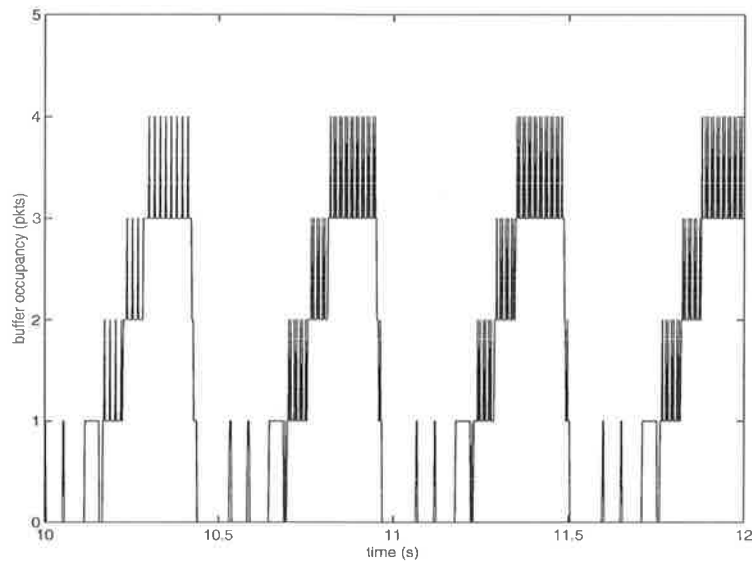


Fig. 3.12(a) Buffer occupancy of node R under the congestion control of TCP Reno under an insufficient buffer condition

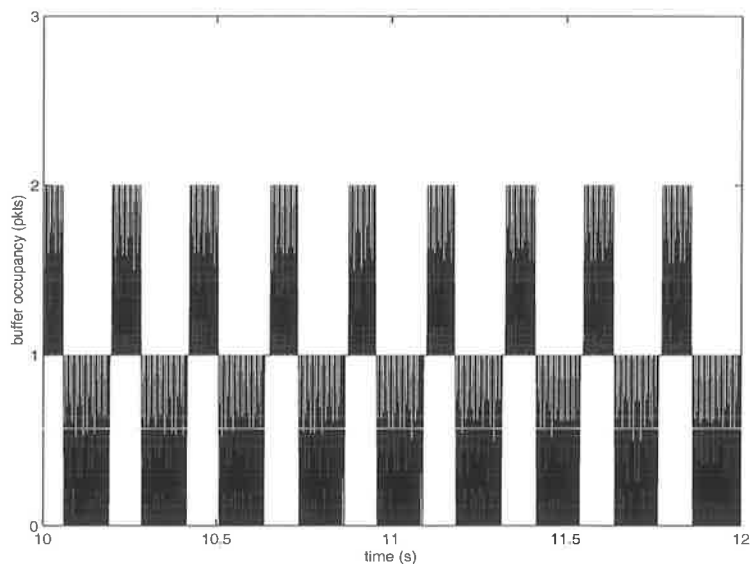


Fig. 3.12(b) Buffer occupancy of node R under the congestion control of ST-PD TCP under an insufficient buffer condition

Figure 3.12(b) confirms that the bottleneck node's buffer occupancy under the congestion control of ST-PD TCP is not affected by the buffer dimension.

The impact of insufficient buffer space on TCP Reno's throughput is shown in Figure 3.13. The throughput of TCP Reno drops to below 1.4Mbps due to the starvation of the bottleneck link immediately after congestion window reductions; while comparing Figure 3.13 to Figure 3.10, there is little difference of the throughput of ST-PD TCP under sufficient and insufficient buffer conditions.

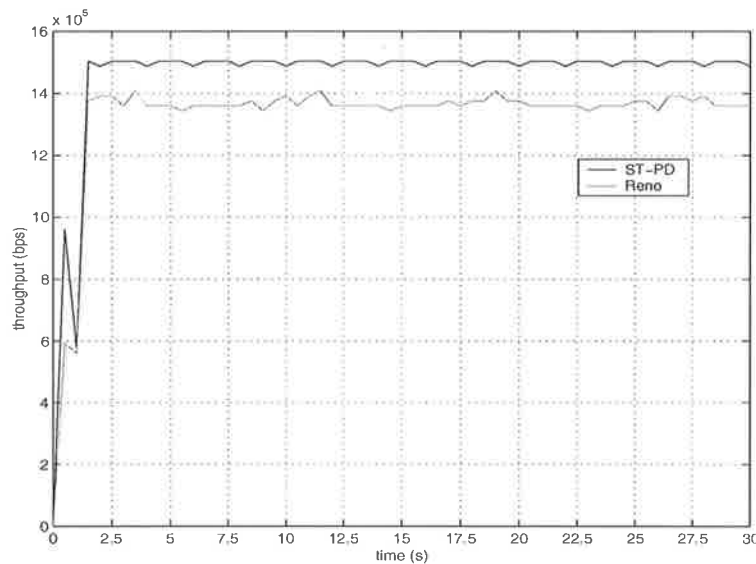


Fig. 3.13 Throughput of TCP Reno and ST-PD TCP under an insufficient buffer condition

3.5.2 Multiple TCP Connections

This subsection contains two simulation scenarios. In the first simulation scenario, a single bottleneck link is shared by three TCP connections, which start and end transmission at different times without resource reservation. In the second simulation scenario, a TCP connection traverses a number of links. Due to cross traffic variations, the bottleneck link of the TCP connection moves from one to another during the TCP connection's life time.

The objectives of simulations described in this subsection are to study:

- 1) how promptly ST-PD TCP reacts to variations of the available bandwidth,

- 2) how effectively ST-PD TCP is able to keep the bottleneck node's queue length around the designed level in the case that multiple TCP connections join in or leave the bottleneck link without being subjected to connection admission control, or in the case that the bottleneck node's position has moved from one node to another during a TCP connection's life time,
- 3) how fairly the bottleneck link's bandwidth is allocated among competing connections under the congestion control of ST-PD TCP.

3.5.2.1 A Fixed Bottleneck Link Case

The simulation scenario is set as follows: The network configuration is shown in Figure 3.14, in which link capacities and propagation delays are given.

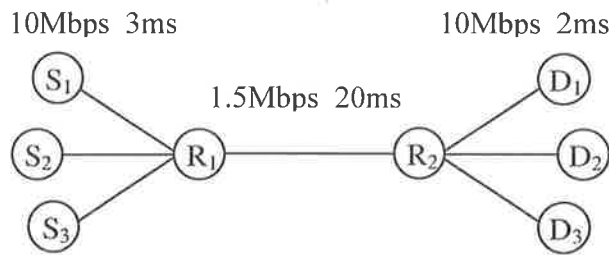


Fig. 3.14 Network configuration of a fixed bottleneck link case

Router R_1 has a finite buffer that can hold 20 packets of the size of 1000 bytes. Connection S_1D_1 starts at time 0 and ends at the 36th second. Connection S_2D_2 starts at the 12th second and ends at the 48th second. Connection S_3D_3 starts at the 24th second and ends at the 60th second. During the simulations, the buffer occupancy at router R_1 and the throughput of the three TCP connections is recorded. The performance of ST-PD TCP is compared with the performance of TCP Reno and TCP Vegas.

Figure 3.15(a) and (b) show the buffer occupancy of router R_1 and the throughput of the three connections respectively when TCP Reno is used. It can be seen that TCP Reno constantly

overflows the bottleneck buffer in order to probe available bandwidth. The more the number of connections shared the buffer, the more frequently the buffer overflows occur.

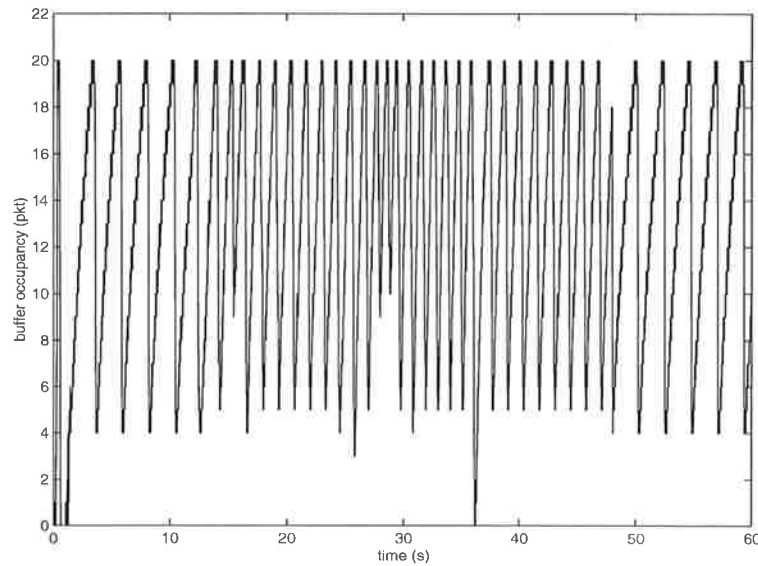


Fig. 3.15(a) Queuing behaviour of TCP Reno

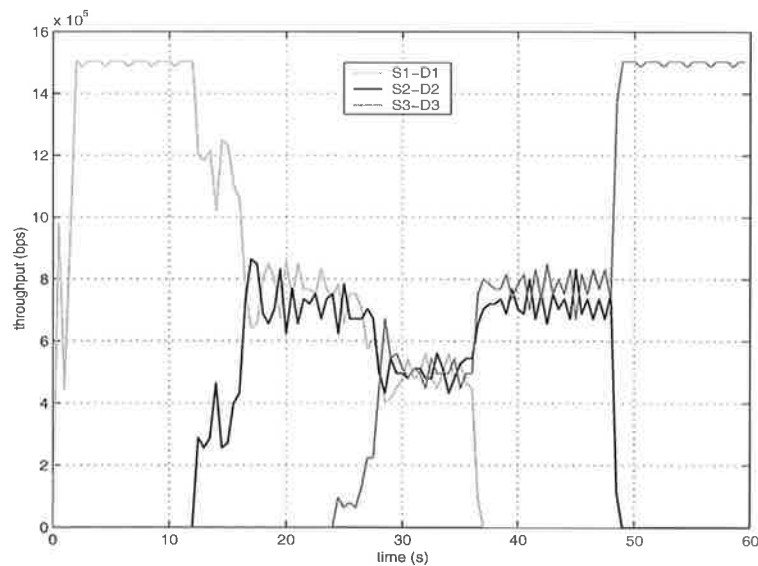


Fig. 3.15(b) Throughput of TCP Reno

Figure 3.16(a) and (b) show the buffer occupancy of router R_1 and the throughput of the three connections respectively when TCP Vegas is used. In Figure 3.16(a), the buffer occupancy depends on the number of connections sharing the buffer. One of the Vegas design objectives is to keep buffer occupancy between 1 to 3 packets [18]. It is clearly indicated in Figure

3.16(a) that the buffer occupancy is kept between 1 to 3 packets when there is only one connection occupying the buffer. However, the buffer occupancy grows rapidly when more connections join in. Like TCP Reno, the achievable throughput of TCP Vegas depends on the buffer size of the bottleneck node. If the buffer size is not allocated large enough, the performance of both TCP Reno and Vegas deteriorates.

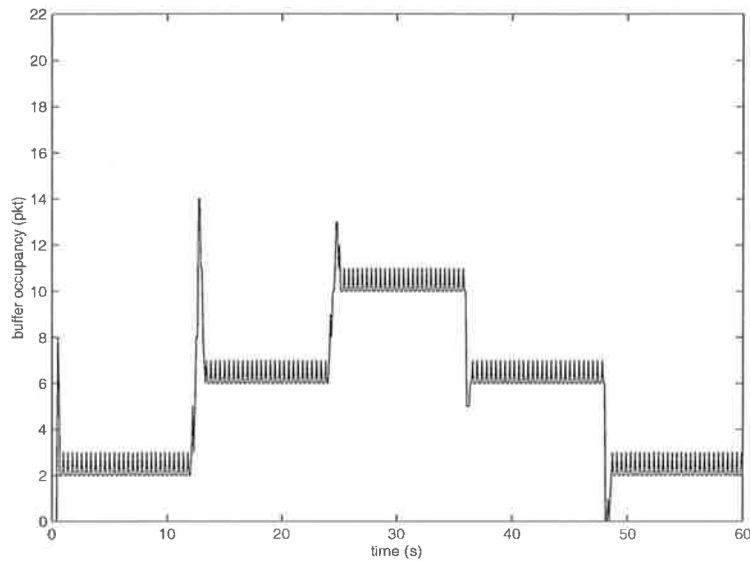


Fig. 3.16(a) Queuing behaviour of TCP Vegas

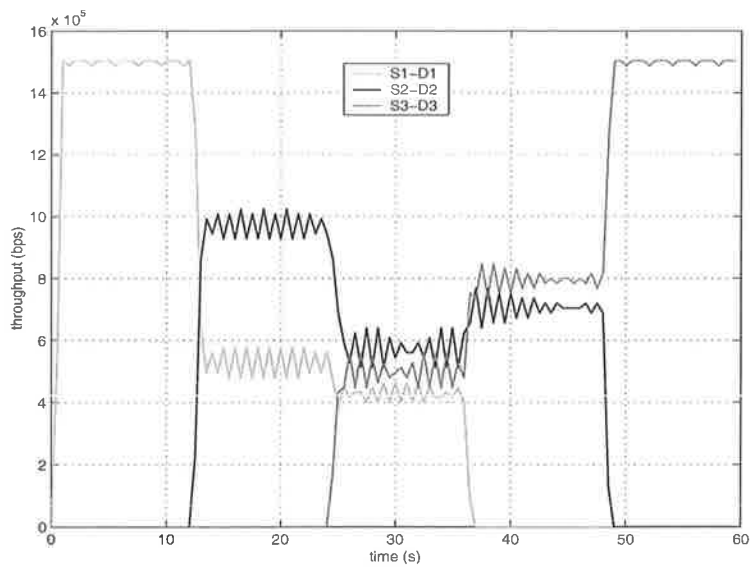


Fig. 3.16(b) Throughput of TCP Vegas

Figure 3.17(a) and (b) shows the buffer occupancy of router R_1 and the throughput of the three connections respectively when ST-PD TCP is used. It can be seen that ST-PD TCP keeps the buffer occupancy around one packet level. There are two overshoots at the 12th second (starting time of S_2D_2) and 24th second (starting time of S_3D_3). However these overshoots are quickly subdued. One of strengths of ST-PD TCP is that its performance is not sensitive to buffer sizes.

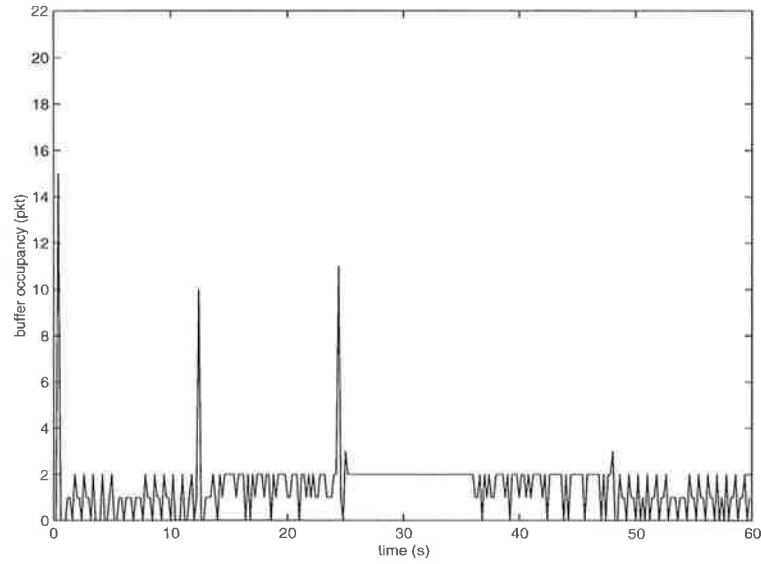


Fig. 3.17(a) Queuing behaviour of ST-PD TCP

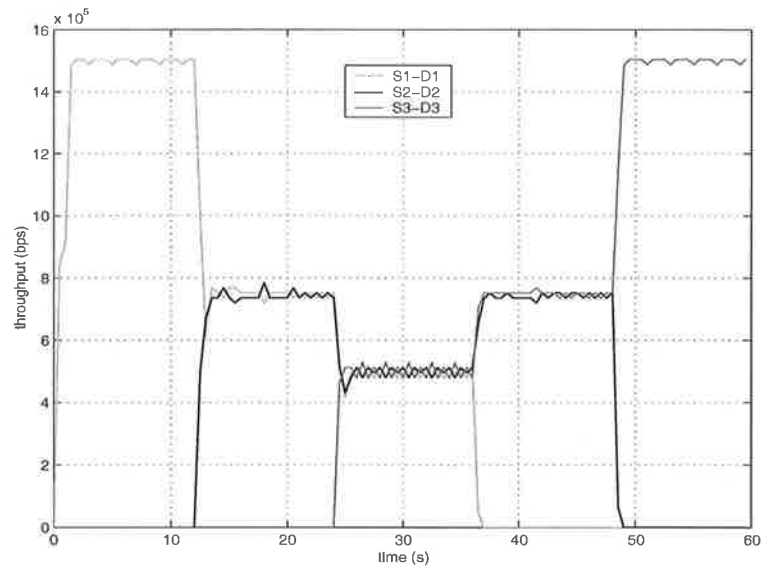


Fig. 3.17(b) Throughput of ST-PD TCP

Among Figure 3.15(b), Figure 3.16(b) and Figure 3.17(b), ST-PD TCP is the only one that makes each connection reach its fair share of bandwidth swiftly.

3.5.2.2 A Moving Bottleneck Link Case

The simulation scenario is set as follows: The network configuration is shown in Figure 3.18, in which link capacities and propagation delays are given. Node R_1 and R_2 each can buffer 20 packets of the size of 1000 bytes.

Connection S_1D_1 starts at time 0 and ends at the 60th second. It shares the link between R_1 and R_2 with connection S_2D_2 , which starts at the 15th second and ends at the 30th second. Connection S_1D_1 also shares the link between R_2 and R_3 with connection S_3D_3 , which starts at the 45th second and ends at the 60th second.

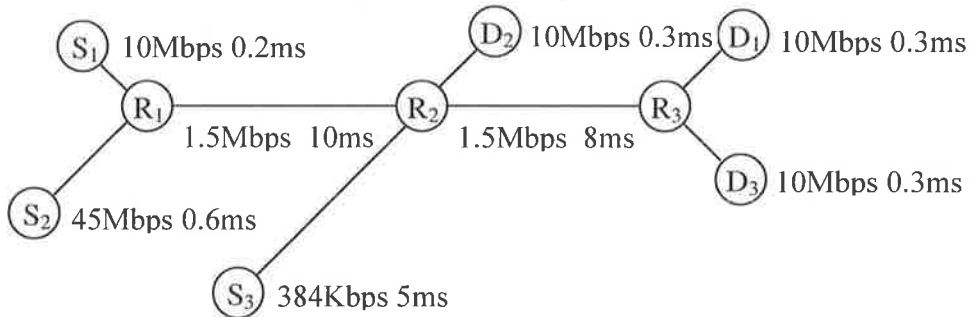


Fig. 3.18 Network configuration of a moving bottleneck link case

Figure 3.19(a) shows the buffer occupancy dynamics of node R_1 and R_2 when ST-PD TCP is used. For connection S_1D_1 , which lasts from time 0 to the 60th second, its bottleneck node moves from R_1 during the period of 0-45th second to R_2 during the period of 45-60th second. Figure 3.19(a) clearly demonstrates that ST-PD TCP is able to effectively keep the bottleneck node's queue length around the designed level, regardless of the bottleneck node's position.

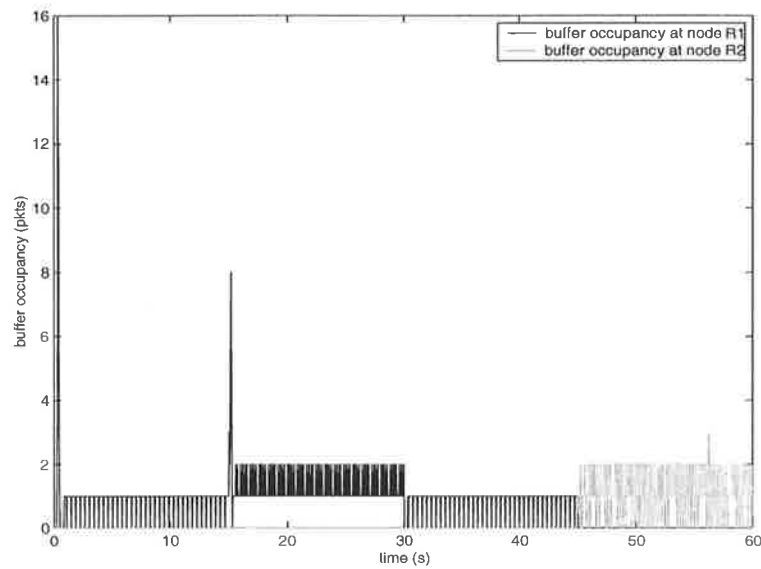


Fig. 3.19(a) Buffer occupancy at R_1 and R_2 under the congestion control of ST-PD TCP

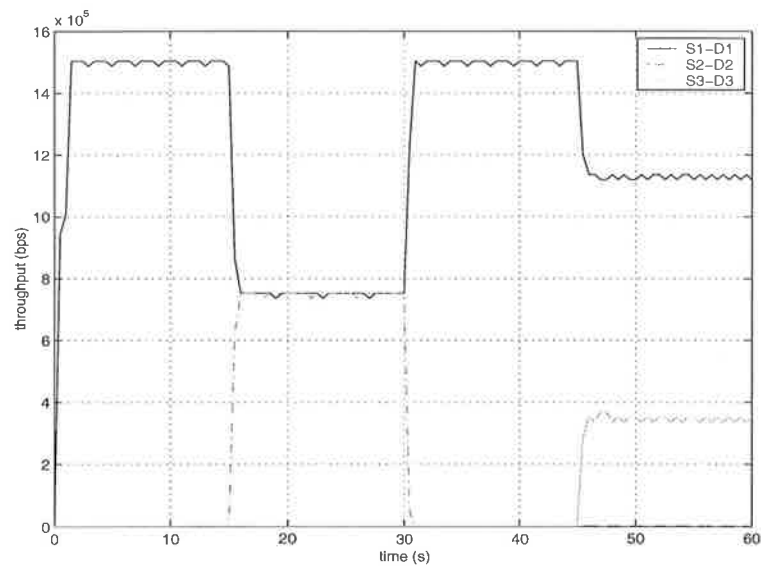


Fig. 3.19(b) Throughput of the three ST-PD TCP connections

Figure 3.19(b) shows the throughput of the three competing ST-PD TCP connections. When S_1D_1 and S_2D_2 sharing the bottleneck link R_1R_2 , the link capacity of 1.5Mbps of R_1R_2 is equally divided between the two competing connections, even though the incoming link to node R_1 from connection S_2D_2 has a much higher capacity than that of connection S_1D_1 . For connection S_3D_3 , which connects to node R_2 through a quarter of a T1 circuit, it can transmit

data at its full speed across the link between R_2 and R_3 , and the rest of the link capacity of R_2R_3 is fully utilised by connection S_1D_1 .

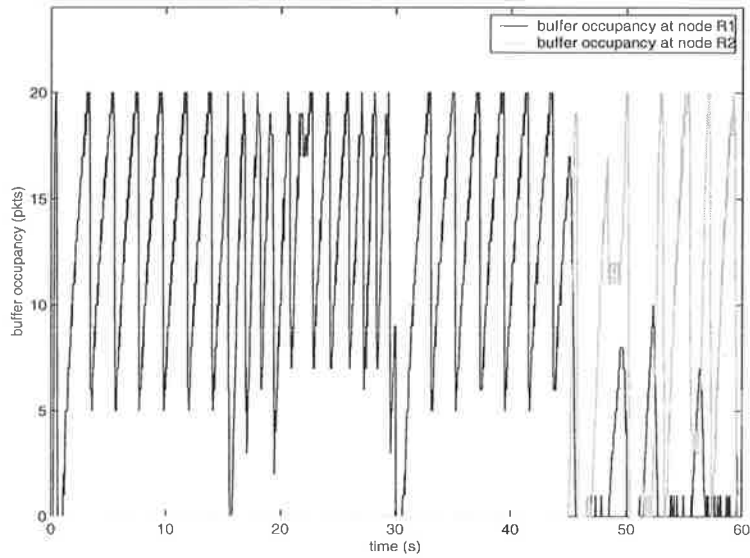


Fig. 3.20(a) Buffer occupancy at R_1 and R_2 under the congestion control of TCP Reno

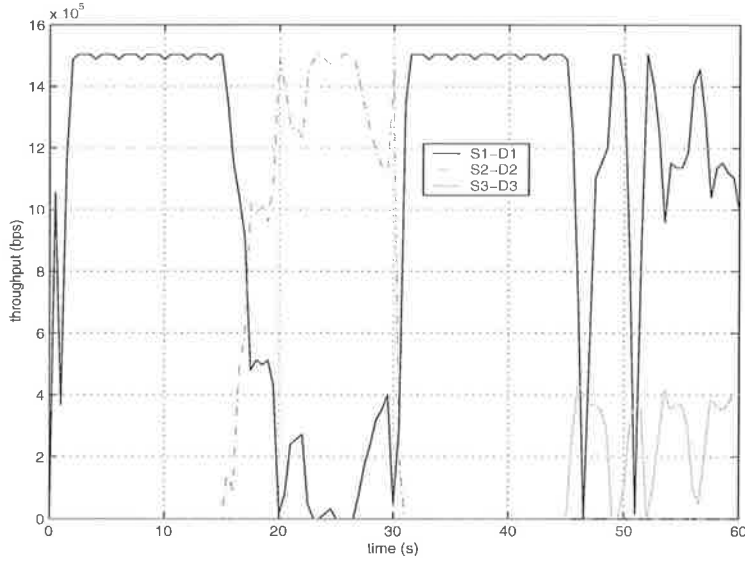


Fig. 3.20(b) Throughput of the three TCP Reno connections

Figure 3.20(a) shows the buffer occupancy dynamics of node R_1 and R_2 when TCP Reno is used. Since TCP Reno cannot restrain the queue length growth at the congested node, the connection whose packets occupy more buffer space of the congested node gets more than its fair share of the bottleneck link. Figure 3.20(b) shows the throughput of the three competing

TCP Reno connections. When connection S_1D_1 is competing with connection S_2D_2 for the link capacity of R_1R_2 , it only gets a small fraction of its fair share of link R_1R_2 's bandwidth. This is because the incoming link to node R_1 from connection S_2D_2 has a much higher capacity than that of connection S_1D_1 , therefore packets from connection S_2D_2 fill the buffer at the node R_1 more quickly than the packets from connection S_1D_1 . When the packets belonging to connection S_1D_1 arrives at node R_1 , they are more likely to face a fully occupied buffer, thereby have a higher chance of being dropped, which hampers the opening of the congestion window at the source node S_1 .

3.5.3 The Impact on Round-Trip Delay of TCP Connections

Although TCP is employed as the transport layer protocol primarily by applications that require the integrity of data transfer but can tolerate long delay of the data transfer, such as FTP and SMTP, there are a number of TCP-based applications that are sensitive to the latency of TCP connections. For example, interactive-oriented applications, such as Telnet, and rlogin (remote login), allow a user to login into a remote host transparently as if the user were working in front of a local host. These applications rely on TCP to transport their data across heterogeneous networks. Due to the interactive nature of these applications, the round-trip delay of the underlying TCP connection becomes an important performance parameter.

The objective of the simulations described in this subsection is to investigate the impact of ST-PD TCP congestion control scheme on TCP connections' round-trip delay.

The simulation scenario is set as follows: Two geographically spanned LANs are interconnected via a leased $\frac{1}{4}$ T1 circuit. The link capacities and propagation delays are shown in Figure 3.21. The node R_1 has a buffer space that can accommodate 40 packets of the size of 1000 bytes. A user at the node D_1 uses FTP to download a large file located in the node S_1 . In the mean time another user at the node S_2 uses Telnet to access the remote host D_2 . The interarrival time of Telnet traffic is drawn from the Telnet traffic library released with *ns-2*. During the simulations, the node R 's buffer occupancy trace, the round-trip delays of Telnet packets and the sequence number of acknowledged FTP packets are recorded.

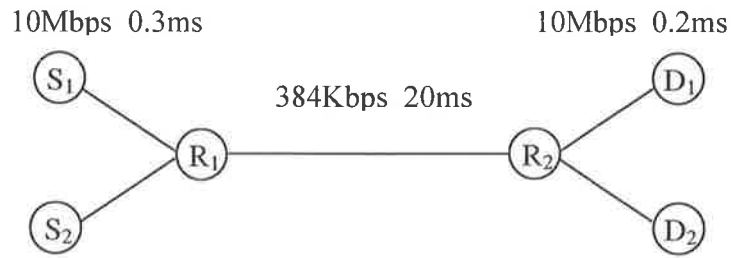


Fig. 3.21 Network configuration of FTP and Telnet connections

Figure 3.22(a) and (b) show the buffer occupancy trace of node R_1 and the round-trip delay of the Telnet packets respectively when ST-PD TCP is used.

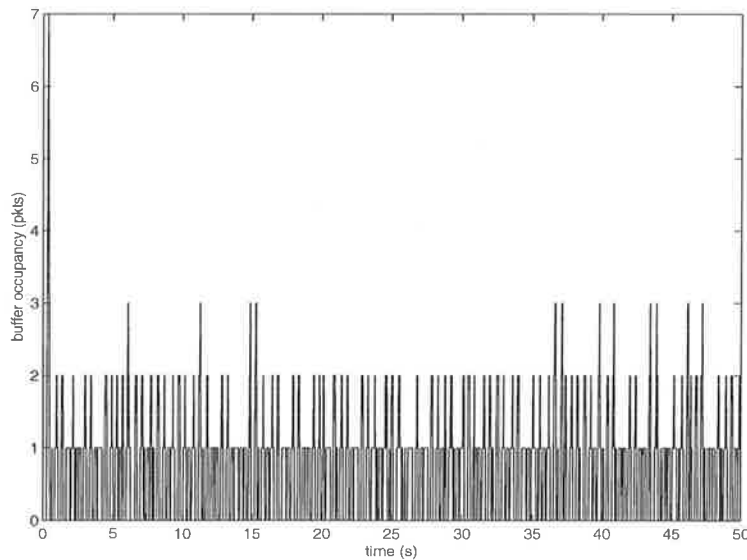


Fig. 3.22(a) Buffer occupancy of node R_1 under the congestion control of ST-PD TCP

The round-trip delay of a Telnet packet consists of propagation time, transmission time and queuing time in both forward and reverse directions. In these simulations, the packet size in the forward direction is 1000 bytes and the packet size in the reverse direction (ACK) is 40 bytes, therefore there is no queue built up in the reverse direction. Since ST-PD TCP effectively keeps the buffer occupancy in the range of 0 to 3 packets, as shown in Figure 3.22(a), the lower bound of the round-trip delay of the Telnet conversation is the round-trip time of a packet that does not experience queuing delay (ie. when the packet arrives at the

node R_1 , the buffer is empty), and the upper bound of the round-trip delay is the round-trip time of a packet that experiences 3-packet queuing delay (ie. when the packet arrives at node R_1 , two packets have already queued in the buffer). The round-trip time of the connection S_2D_2 without queuing delay is 63ms, and the round-trip time of the connection S_2D_2 with 3-packet worth of queuing delay is 127ms. Therefore the round-trip delay of the Telnet conversation should be bounded between 63ms to 127ms, which is confirmed in Figure 3.22(b).

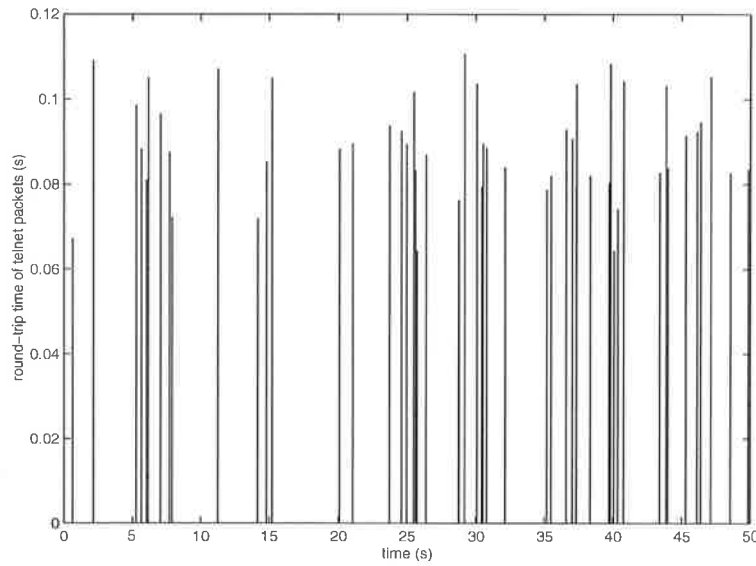


Fig. 3.22(b) RTT trace of Telnet packets under the congestion control of ST-PD TCP

Figure 3.23(a) and (b) show the buffer occupancy trace of node R_1 and the round-trip delay of the Telnet packets respectively when TCP Reno is used. Comparing Figure 3.23(a) and (b), it can be seen that the round-trip delay of the Telnet conversation is proportional to the buffer occupancy at the node R_1 . Under the congestion control of TCP Reno, the bottleneck node's queue growth is only bounded by the physical buffer size. In the simulations, the buffer at the node R_1 can accommodate 40 packets, therefore the upper bound of the round-trip delay of the connection S_2D_2 is the round-trip time of a packet that experiences 40-packet queuing delay, which is about 894ms. Figure 3.23(b) confirms that the maximum round-trip delay of the Telnet connection is just under 0.9 second.

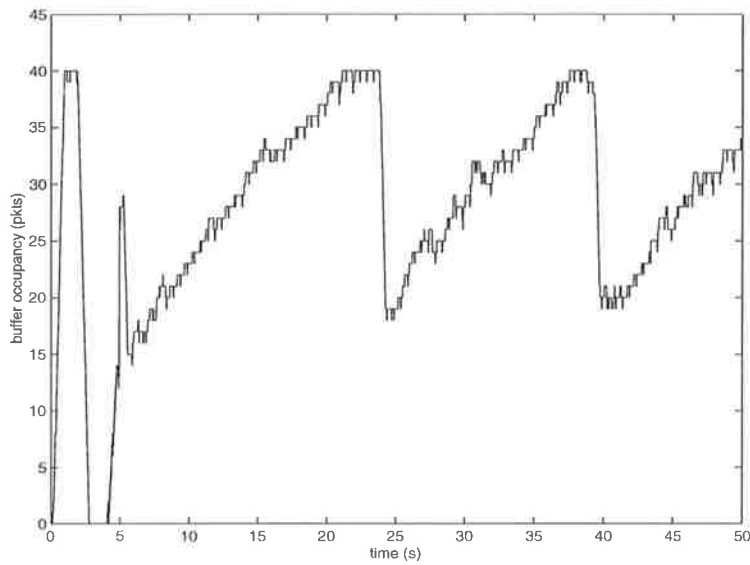


Fig. 3.23(a) Buffer occupancy of node R_1 under the congestion control of TCP Reno

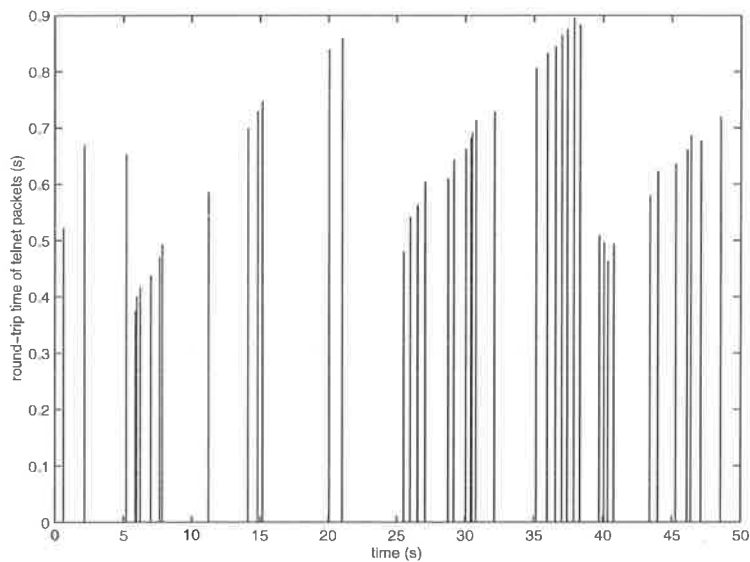


Fig. 3.23(b) RTT trace of Telnet packets under the congestion control of TCP Reno

Generally users can notice the latency in an interactive process if the round-trip delay exceeds 400ms. The typical example is a TV interview carried over Geosynchronous Earth Orbit (GEO) satellite links. The round-trip propagation time of a GEO satellite hop alone is at least 500ms. TV viewers can notice that there is a significant gap between the time that a question is asked by an interviewer and the time that an interviewee responds, which, to some extent, hampers the fluency of the interactive process.

The excessive round-trip delay has the same effect on networking users who use Telnet and rlogin or other similar interactive services. For example, when a user types a command to list a directory in a remote host, he expects the result should be displayed immediately on his screen as if he were listing a directory in a local host. When the round-trip delay exceeds 400ms as is the case shown in Figure 3.23(b), the user can feel that the remote access service is not so transparent.

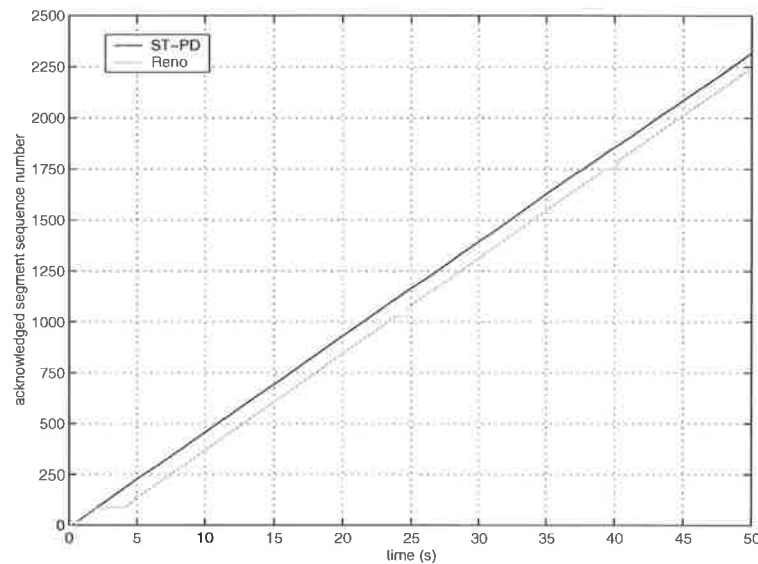


Fig. 3.24 Acknowledged sequence number of FTP packets

While keeping the round-trip delay of the Telnet connection S_2D_2 low, the goodput of the FTP connection S_1D_1 under the congestion control of ST-PD TCP is not compromised. A comparison result of the acknowledged sequence traces of FTP packets under the congestion control of Reno and ST-PD TCP is presented in Figure 3.24, which shows that the growth of the number of in-order delivered FTP packets is steady and consistent under the congestion control of ST-PD TCP, whereas buffer overflows under the congestion control of TCP Reno are clearly demonstrated on the acknowledged sequence trace of TCP Reno. There are a number of places where the acknowledged sequence number of TCP Reno does not change with time, which indicate duplicate acknowledgments have been issued due to receiving out-of-order packets.

3.6 Summary

A self-tuning PD based TCP congestion control scheme has been introduced in this chapter. In this scheme, TCP congestion window is adjusted by a classic PD controller aiming to keep bottleneck node's buffer occupancy around an ideal operating point; the control gains of the PD controller are tuned on-line by a fuzzy logic controller based on perceived bandwidth-delay product of the TCP connection.

Due to the proportional part in the PD controller, the increase or decrease of a congestion window size is proportional to the control error. When the buffer occupancy deviates far from the ideal operating point, the control error becomes large and the congestion window size will be aggressively adjusted accordingly so that the buffer occupancy can be quickly drawn back to the designed operating level. When the buffer occupancy approaches the ideal operating point, the congestion window will be finely adjusted so that amplitudes of buffer occupancy oscillation can be restrained. Furthermore, the derivative part of a PD controller, which is sensitive to error change rate, helps to suppress control loop latency induced overshoots. Compared with heuristic based congestion control schemes in current TCP implementations, ST-PD TCP can adapt to available bandwidth variations more swiftly and accurately.

In addition to this, a number of benefits result directly from ST-PD TCP's ability to confine bottleneck node's buffer occupancy into narrow ranges. The most obvious one is that the queuing delays incurred on packets transmitted by ST-PD TCP are more consistent and less excessive compared with the ones experienced by current TCP transmitted packets. This feature can substantially lift the performance of a class of interactive-oriented applications, such as Telnet and rlogin, which rely on TCP to transport their data across networks.

The less obvious but more important implication of ST-PD TCP's ability to restrain the excessive packet queue growth in intermediate nodes is that the functions of congestion control and error control can be decoupled in ST-PD TCP. In TCP Tahoe, TCP Reno and TCP Vegas, congestion control and error control are closely coupled. When a packet loss is detected, the congestion window is cut down (congestion control) and the lost packet is retransmitted (error control). The coupling of congestion control and error control has a

historical reason. When TCP was designed, computer networks were wired networks and congestion was the primary cause of packet losses. However, as wireless links become a fast growing part in the global information infrastructure, the assumption that packet losses are caused by buffer overflows does not hold any more. As a consequence, the performance of existing TCP implementations deteriorates significantly over unreliable transmission media. Unlike current TCP implementations, ST-PD TCP conducts congestion control and error control independently. When a packet loss is detected through receiving a number of duplicated acknowledgments, the lost packet is retransmitted (error control), but the congestion window size is still regulated (congestion control) according to the ST-PD congestion control scheme described in Section 3.4. Therefore, packet losses over wireless links will have less detrimental impact on the throughput of ST-PD TCP. The performance of ST-PD TCP over wireless links is presented in Chapter 4.

Chapter 4

Performance Analysis of ST-PD TCP over Wireless links

The rapid progress in wireless networking technology offers many possibilities in bringing computing and communications services to nomadic users. As Internet has become the dominant global information infrastructure, the imminent goal of wireless networking technology is to provide Internet access to mobile users with the quality and availability comparable to its landline counterpart.

TCP, one of the key protocols in Internet, was originally designed and well tuned for wired networking environments. However, its performance deteriorates significantly over wireless links, where packet losses caused by channel fading are unavoidable. This chapter provides performance analysis of ST-PD TCP in wireless networking environments using both analytical and simulation methods.

In Section 4.1, a broad range of wireless data networks are overviewed first, followed by a brief discussion of wireless channel characteristics. TCP performance deterioration over lossy wireless links and the techniques proposed to alleviate the performance deterioration are reviewed. After outlining the limitations of existing techniques, ST-PD TCP's solution to this problem is described in Section 4.2. A performance analytical model is developed in Section 4.3 to quantify the impact of wireless losses on the congestion window of ST-PD TCP, from which the achievable throughput of ST-PD TCP is derived. To confirm the numeric analysis, simulation results are presented in Section 4.4. The features of ST-PD TCP in wireless networking environments are summarised in Section 4.5.

4.1 Background

4.1.1 Overview of Wireless Data Networks

Wireless communication systems can be classified into two categories: terrestrial based and satellite based. Currently, there are two types of terrestrial wireless data networks: cellular packet data networks and wireless Local Area Networks (LANs).

Existing cellular packet data networks offer moderate data rate services with wide area mobility support. The following cellular packet data networks are commercially available: Advanced Radio Data Information Service (ARDIS) [29], Ram Mobile Data (RMD) [30], Cellular Digital Packet Data (CDPD) [31], and General Packet Radio Service (GPRS) [32]. The data rates in these systems range from 4.8kbps in ARDIS to 170kbps in GPRS. Mobility support in these systems usually covers metropolitan areas. The future third generation mobile communication systems known as Universal Mobile Telecommunications System (UMTS)/International Mobile Telecommunications 2000 (IMT2000) will support a data rate up to 384kbps with wide area coverage and a data rate up to 2Mbps in indoor or microcell environments [33].

Wireless LANs, on the other hand, offer high data rate services, but cover very limited areas. The data rate of wireless LANs normally exceeds 1Mbps and their coverage is often confined to a few buildings or even within a building [34]. Unlike cellular packet data networks, wireless LANs operate over unlicensed Industrial Scientific and Medical (ISM) radio spectrum or infrared band. Wireless LANs are preferable to wired LANs in situations where laying cables is either not feasible, uneconomical, or some local mobility is required such as in warehouse inventory systems. Since IEEE 802.11 became the industrial standard for wireless LANs, large scale production has made IEEE802.11-compatible wireless LAN products more affordable.

With the enormous popularity of Internet, there is a growing demand for Internet access anywhere and anytime. As a result, cellular packet networks and wireless LANs are becoming a part of access networks in the Global Information Infrastructure (GII). In terms

of network access technologies, Wireless Local Loop (WLL), which itself is not a network, is also worth a mention. Currently two types of WLL, namely Local Multipoint Distribution System (LMDS) and Multipoint Multichannel Distribution Service (MMDS) are also used to carry data traffic. The MMDS (also called wireless cable), which was originally designed to provide alternative cable TV services in rural and regional areas, operates over the spectrum between 2.150Ghz and 2.682Ghz with a typical cell radius of 25-35 miles. When used to provide two-way data services, MMDS requires additional reverse links [35]. LMDS provides full two-way broadband wireless access. It operates at 27.5-28.35 Ghz and 29.1-29.25Ghz bands with a cell radius of less than 5 miles. LMDS was introduced to deliver two-way high-speed data, video and telephony services to residential areas [36].

Despite being designed initially for relaying bulk telephone and TV traffic, telecommunications satellites have been used for data communications for some time. With the maturing of the Very Small Aperture Terminal (VSAT) technology, satellite data networks have gained a substantial market share in interconnecting corporate LANs which may span across a large geographic area. In North America, VSAT networks have been widely used to provide Point of Sale (POS) and banking services in regional areas.

To target the ever growing market of broadband Internet access, several companies started to provide Internet access services directly to end users via satellites. One example of such services is DirecPCTM run by Hughes Communication Inc., which can provide up to 400Kbps down-links to its customers anywhere in the United States with dial-up circuits as its up-links [37]. In the mean time a number of systems, which aim to deliver two-way broadband Internet access services directly to end users via satellites, are under construction. Examples of such systems are:

- SPACEWAY: SPACEWAY is a GEO satellite based broadband data network formulated by Hughes Communication Inc.. It will initially deploy two HS 702 GEO satellites, plus one in-orbit spare, to cover North America through 48 beams. Using a fixed 66cm Ultra Small Aperture Terminal (USAT), end users can acquire a 16kbps up-link and a 6Mbps down-link, and faster up-links will be available with larger antennae. SPACEWAY will adopt advanced on-board

packet-switching, multibeam antenna technologies and high-power transponders to achieve the goal of two-way broadband Internet access via satellites [38]. Many similar systems using GEO satellites have been proposed, including Astrolink (Lockheed Martin), Cyberstar (Loral), Millennium (TRW).

- Skybridge: Skybridge is a Low Earth Orbit (LEO) satellite based broadband Internet access system proposed by Alcatel. It will rely on a constellation of 80 LEO satellites to cover virtually every corner of the world. It will offer 2Mbps up-links and 20Mbps down-links to individuals through residential terminals, 10Mbps up-links and 100Mbps down-links to corporate users through larger terminals [39]. Similar systems include TELEDESIC, which will provide 2Mbps up-links and 64Mbps down-links to its customers through a constellation of 288 LEO satellites [40].

The common feature of all satellite based systems is the huge footprint of telecommunications satellites, which gives satellite based telecommunications systems several advantages. For example, unlike building terrestrial networks, the cost of rolling out a satellite-based network is not sensitive to the population density and geographic locations. Another important feature of satellite based systems is the inherent multicast capability of telecommunications satellites, which gives satellite data networks the edge in providing multimedia services.

Challenges associated with satellite networking technologies also exist. For LEO satellite networks, the system complexity is very high due to the need to handle dynamic routing and handover procedures. Another drawback of the non-GEO satellite based systems is that incremental system deployment is not feasible. Generally, LEO satellite systems require the whole satellite constellation be completed before they are operational, which means a substantial amount of capital must be invested before any revenue can be generated. There has not been a commercially successful case of running such networks. The debacle of the world's first LEO satellite mobile phone network Iridium demonstrates the commercial vulnerability of non-GEO satellite based systems. The huge up-front investment of Iridium system made it difficult to cut its service prices, further alternative services offered by new

technologies gradually occupy its niche market. After the fiasco of Iridium system, it is uncertain whether other similar systems under planning will be rolled out on schedule.

On the other hand, GEO satellite based systems have achieved considerable commercial success for some time. The technology behind GEO satellite systems is relatively simple, thereby the cost of services offered by such systems is competitive. For example, the Internet access services provided by DirecPCTM can compete with its terrestrial counterpart in prices. However, due to the high altitude of geosynchronous earth orbit, signals passed through GEO satellite links incur long propagation delays. The typical one-hop delay of GEO satellite networks is about 250ms, and a round-trip delay is about half a second. In addition to satellite channel fading, the long propagation delays further deteriorate TCP performance in GEO satellite networks.

Since currently there are no commercial LEO satellite data networks operational and the future of several planned LEO projects is in doubt, the chapter will only discuss the issues related to TCP performance in GEO satellite networking environments.

4.1.2 Characteristics of Wireless Channels

Thermal noise exists in any types of communication channels, regardless of whether they be landline or wireless. Thermal noise is modelled as Additive White Gaussian Noise (AWGN), which has a flat power spectral density and a zero-mean Gaussian voltage probability density function. Thermal noise can cause bit error in two ways: 1) Thermal noise corrupts data samples directly, ie., Signal Noise Ratio (SNR) at the receiver side falls below a threshold and data samples cannot be detected. 2) The phase distortion effect introduced by bandlimiting filter at receiver side causes Inter-Symbol Interference (ISI) when thermal noise passes through such a filter.

When radio signal propagates in ideal free space, the received signal power is attenuated by a factor of $\left(\frac{4\pi d}{\lambda}\right)^2$, where d is the distance between the transmitter and the receiver, and λ is the wavelength of the signal [41]. Clearly, in ideal free space, the longer the radio signal

travels or the higher the radio signal frequency is, the more attenuation the radio signal suffers.

In a real world, the ideal free space propagation channel seldom exists, and radio signal propagation is usually affected by reflection, diffraction and scattering [42]. As a result, a radio signal travels to a receiver over multiple paths, and multipath propagation causes fluctuation in the received signal's amplitude and phase. This phenomenon is known as multipath fading. If there is a dominant component, such as a line-of-sight ray, among the received signal, the channel is termed Rician, otherwise, the channel is called a Rayleigh channel [43].

Multipath fading can be categorised as frequency-selective fading or flat fading. These cause bit errors in different ways [43].

- Frequency-selective fading: the delay spread caused by multipath propagation exceeds the symbol duration, and this causes ISI.
- Flat fading: the delay spread caused by multipath propagation is within the symbol duration. However, components from different paths add up destructively at the receiver side, and this leads to a substantial reduction in SNR.

Despite the fact that various techniques, such as channel equalisation, are used to combat the wireless channel fading effects, transmission errors over wireless links cannot be eliminated.

4.1.3 TCP Performance Degradations over Wireless Links

As mentioned earlier in Chapter 3, higher priority on robustness has led ARPANET to adopt a stateless and thin IP layer. The IP layer function is very simple: packet forwarding. When arriving packets fail to pass error checks at intermediate nodes or intermediate nodes' buffer is full, packets are simply discarded. No actions are taken to inform the corresponding senders about the packet drops. It is the higher layer protocols' responsibility to recover packet losses.

Without any assistance from the IP layer, TCP senders can only rely on acknowledgments sent back from TCP receivers to infer packet losses and retransmit lost packets. When TCP was designed, packet losses were primarily caused by buffer overflows in intermediate nodes. Since it is also TCP's responsibility to provide congestion control, besides retransmitting lost packets, TCP cuts down its congestion window size to reduce the network load at the same time. Clearly, the functions of TCP congestion control and TCP error control are closely coupled, and TCP congestion control function interprets packet losses as signals of network congestion. This mechanism works well as long as congestion is the primary cause of packet losses.

However, as the usage of both terrestrial wireless and satellite links becomes more common, the assumption that congestion is the major cause of packet losses no longer holds. When packets are corrupted during transmission over wireless links and are consequently discarded due to failed error checks, TCP still interprets this kind of packet loss as an indication of network congestion, and reduces its congestion window size to slow down its transmission unnecessarily. Therefore transmission errors over wireless channels degrade TCP performance in two ways:

- 1) Corrupted packets need to be retransmitted.
- 2) TCP is forced to shrink its congestion window and slow down its transmission to alleviate the "congestion" that does not exist.

It is the second factor that causes significant TCP performance deterioration. The need to share scarce radio spectrum already restricts wireless link speed to lag far behind its landline counterpart. The inability of TCP to differentiate congestion related packet losses and transmission error induced packet losses further exaggerates the poor performance of data transfer over wireless links.

4.1.4 Existing Performance Enhancing Techniques and Limitations

Since the emergence of wireless data networking technologies, TCP performance degradation over lossy links has been widely reported, and a substantial number of

techniques have been developed aiming to alleviate the performance deteriorations. An extensive survey plus comparison studies of proposed techniques in this area have been conducted in [44]. Essentially, these techniques follow two distinct directions. One solution is to shield TCP from packet losses incurred over error-prone links. The other approach is to make TCP to be able to differentiate link error induced packet losses and congestion caused packet losses, and to treat them differently.

Enhancing TCP performance through shielding TCP from packet transmission errors does not require modifications of TCP implementations. Packet transmission errors are expected to be solved locally on a link-by-link basis before they reach the end-to-end oriented transport layer. So far, techniques based on physical layer, data link layer and network layer have been proposed to shield TCP from transmission errors.

Physical layer schemes: robust modulation and Forward Error Correction (FEC) coding techniques have been developed to combat wireless channel fading [45].

Although channel coding techniques can reduce bit error rate, they cannot eliminate transmission errors totally. When surrounding objects or communication terminals change their positions with time, wireless channel fading behaviour is rather unpredictable. Therefore, physical layer enhancement schemes cannot solely shield TCP from wireless losses.

Data link layer schemes: interleaving is used to spread clustered bit errors and Automatic Repeat reQuest (ARQ) is adopted to retransmit corrupted frames locally [46][47].

Interleaving techniques can scramble clustered bit errors, thus improve the performance of FEC. However, the side effect is interleaving delay. The larger the interleaving depth, the longer the interleaving delay is incurred. The data link layer provides services to a wide range of applications, some applications have strict requirements on delays but can tolerate some degree of error while other applications require data integrity but are insensitive to delays. Therefore interleaving depth should be moderate to satisfy the distinct requirements of different applications.

ARQ schemes provide link layer error recovery by retransmitting corrupted frames on a link-by-link basis. However, the number of retransmissions should be limited, and the residual errors should be left to higher layer protocols to recover. There are two reasons to limit the number of link layer retransmissions:

- 1) Some applications do not require data integrity but have a very tight budget on delays.
- 2) TCP has its own error recovery mechanism. If a packet is not acknowledged within a specific time, TCP will retransmit the packet regardless of the undergoing link layer retransmission efforts.

Clearly, data link layer schemes have a limited effect on shielding TCP from wireless losses.

Network layer schemes: in the snoop-TCP scheme [48], snoop agents are deployed at the gateway between wireless networks and wired networks. Snoop agents conduct local error recovery on network layer by caching unacknowledged packets, retransmitting lost packets and suppressing duplicate acknowledgments, thereby shield TCP from transmission errors. Snoop-TCP scheme is very similar to the Split-TCP [49] and Indirect-TCP [50] schemes in the sense that error recovery is conducted locally. The only difference is that Snoop-TCP maintains TCP end-to-end semantics, while Split-TCP and Indirect-TCP break a TCP connection into two end-to-end connections, one covering the wireless networking part and the other covering the wired networking part. In that sense Split-TCP and Indirect-TCP can be classified as transport layer schemes.

The main drawback of Split-TCP and Indirect-TCP is the violation of TCP end-to-end semantics. It is not always feasible to split a TCP connection into two parts. For example, if TCP segments are encrypted, intermediate nodes will not be able to retrieve the header information from TCP segments. In that case, it is impossible to split a TCP connection into two parts.

Snoop-TCP does not break TCP end-to-end connections, therefore its applicability is not affected by transport layer encryption. However, Snoop-TCP has its own limitations. First, the Snoop-TCP scheme requires forward and reverse paths of a TCP connection pass the same gateway where snoop agents are deployed. If TCP connections have asymmetrical

structures, the Snoop-TCP scheme fails. The other problems Snoop-TCP faces are deployability and scalability. Snoop agents must be implemented at the gateways between wired and wireless networking parts, and snoop agents must maintain every TCP connection's state, which is not a trivial task. Finally, Snoop-TCP has the same limitation that data link layer ARQ schemes have: retransmissions are effective only for a limited number of times. If a packet is not acknowledged before TCP's retransmission timer expires, TCP will retransmit the lost packet regardless of the retransmission efforts made by snoop agents.

Another way to alleviate TCP performance degradations is to make TCP aware of corruption induced packet losses and those caused by congestion. This can be done by either Explicit Loss Notification (ELN) or Explicit Congestion Notification (ECN) schemes [51]. With the assistance of ELN or ECN signalled by intermediate nodes, TCP will slow down transmission only when packet losses are caused by network congestion.

ECN and ELN schemes also face the problem of deployability. ECN and ELN should be signalled by intermediate nodes. Currently, commercial routers or layer-3 switches do not provide such functions.

4.2 Decoupling Error Control and Congestion Control in ST-PD TCP

As described in earlier in Section 4.1.3, the fundamental cause of TCP performance degradation over error-prone links is the coupling of error control and congestion control functions in TCP. ECN and ELN schemes try to decouple error control and congestion functions by explicitly notifying TCP senders whether congestion takes place or whether transmission errors occur. The problem associated with ECN and ELN schemes is that intermediate nodes must provide extra services of marking ECN or ELN in the IP header field when experiencing congestion or wireless losses. These requirements contradict the principle of IP-based network design: network layer functionality should be kept simple to ensure the robustness of network layer services.

ST-PD TCP also applies the principle of decoupling error control and congestion control to enhance its performance over error-prone links. However, unlike ECN and ELN schemes,

ST-PD TCP can conduct congestion control and error control independently without requiring any assistance from intermediate nodes.

As described in Section 3.4, a ST-PD TCP source node estimates the bottleneck node's buffer occupancy based on the variations of the connection's round-trip time. From the estimated buffer occupancy, the ST-PD TCP source node can infer the current network load condition, and the self-tuning PD based controller performs congestion control function all the time, aiming to keep the bottleneck node's buffer occupancy around the designed operating level. Hence, ST-PD TCP no longer relies on the detection of packet losses to trigger its congestion control function.

The flow-chart of ST-PD TCP error control and congestion control is shown in Figure 4.1. The left branch of the flow chart shows that, as long as an acknowledgment arrives, regardless of whether it is a duplicate acknowledgment, ST-PD TCP always calculates the current RTT. From this the PD controller can infer the current network load condition and adjust the congestion window size. Error control is independent of this process: if the number of duplicate acknowledgments exceeds a threshold, ST-PD TCP retransmits the lost packet.

In severe network congestion when no acknowledgment is received, ST-PD TCP retains the retransmission timer mechanism from current TCP implementations as a safety net against congestion collapse. If no acknowledgment is received before the retransmission timer expires, the ST-PD TCP sender shuts its congestion window size to one segment. This mechanism is indicated by the right branch of Figure 4.1.

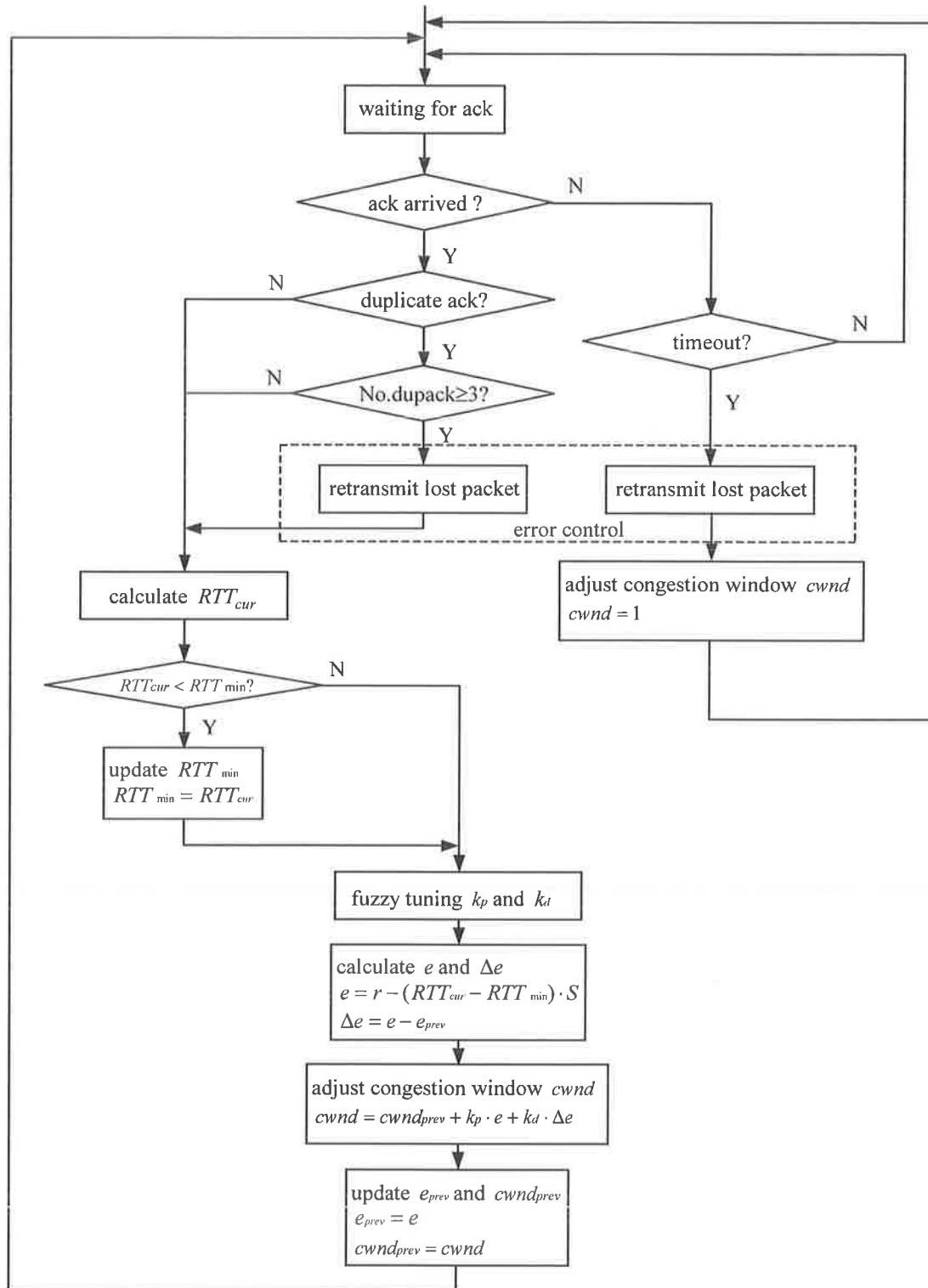


Fig. 4.1 Flow-chart of ST-PD TCP error control and congestion control

4.3 Performance Analysis

To study the performance of ST-PD TCP in wireless networking environments, an analytical model has been developed to quantify the effect of packet losses on the congestion window of ST-PD TCP. The numerical analysis of ST-PD TCP performance over lossy links is presented in this section.

4.3.1 The Impact of Wireless Losses on the Congestion Window of ST-PD TCP

Since the congestion window size of ST-PD TCP is regulated by the self-tuning PD controller according to estimated buffer occupancy at the bottleneck node, ST-PD TCP does not shrink its congestion window drastically on experiencing packet losses as conventional TCPs do. However, wireless losses cause disturbance on buffer occupancy at the bottleneck node, thereby disturbing the congestion window regulation process. It takes time for the self-tuning PD controller to settle down the disturbance.

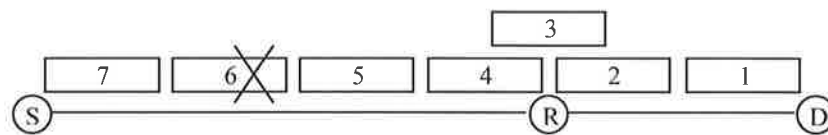


Fig. 4.2(a) Buffer occupancy before a wireless loss

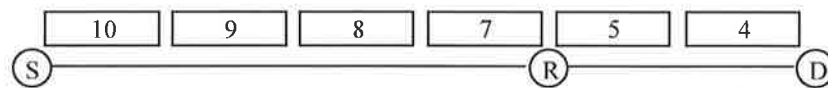


Fig. 4.2(b) Buffer occupancy after a wireless loss

Figure 4.2(a) and (b) show how a packet loss due to transmission errors affects the buffer occupancy at node R. In Figure 4.2(a), node R has just finished transmitting packet 2. Packet 3 is buffered at node R, and packets 4, 5, 6, 7 are in transit. When the self-tuning PD controller at node S reaches a steady state, the buffer occupancy at node R remains constant and is equal to the reference point, which is the ideal operating point set by the self-tuning PD controller. Therefore the control error is reduced to zero, and the output of the self-tuning

PD controller, which is the congestion window size, remains unchanged. The reference point $r(t)$, buffer occupancy $b(t)$, control error $e(t)$ and congestion window $w(t)$ are shown in Figure 4.3(a), (b), (c) and (d)[†] respectively, and the steady state remains until time t_0 .

Assuming packet 6 is corrupted and discarded during the transmission. When packet 7 arrives at node R, the buffer at node R is empty, as shown in Figure 4.2 (b). Therefore packet 7 is transmitted immediately without experiencing any queuing delay. After the acknowledgment of packet 7 reaches source node S at time t_0 , the PD controller detects the buffer occupancy variations. In this case, the estimated buffer occupancy $b(t_0)$ falls below the reference point. As a result, the PD controller starts to increase the congestion window size. Due to the round-trip delay, the PD controller notices the buffer occupancy starting to increase at time t_1 , where $t_1 = t_0 + RTT$. Figure 4.3(b), (c) and (d) show the impact of a packet loss on buffer occupancy, control error and congestion window size respectively after time t_0 .

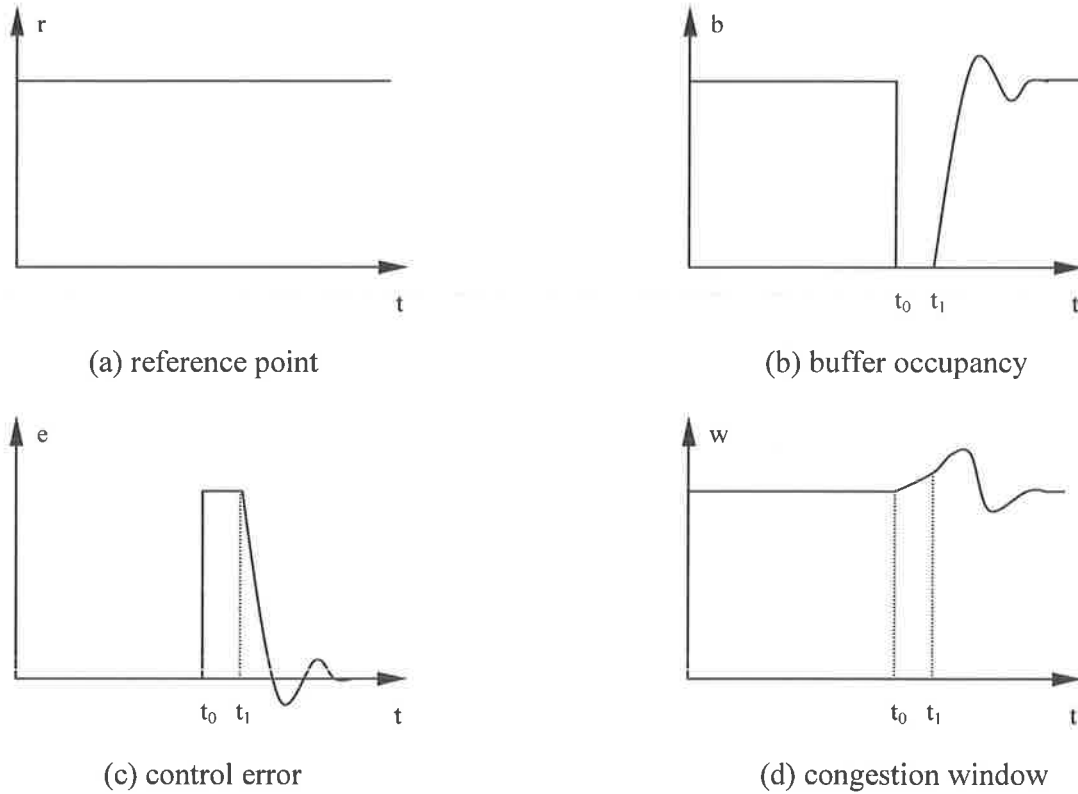


Fig. 4.3 The effect of a packet loss on ST-PD TCP

[†] These figures are for illustration only. There are not any scales on the axes.

Clearly, the impact of wireless losses on the congestion window of ST-PD TCP can be quantified through transient response analysis of the PD controller, from which the effect on ST-PD TCP's throughput can be derived.

4.3.2 Transient Response Analysis of ST-PD TCP under Disturbance

Consider the node R in Figure 4.2. Node R is the bottleneck node and the capacity of its outgoing link is denoted as departure rate $d(n)$ in this transient response analysis. The ST-PD TCP source node S regulates its congestion window size, which determines the packet arrival rate $a(n)$ at node R. If the packet arrival rate $a(n)$ is higher than the packet departure rate $d(n)$, a packet queue is built up at node R, and the queue length is denoted as $b(n)$. If the packet arrival rate $a(n)$ is lower than the packet departure rate $d(n)$, the existing packet queue length is reduced until the packet queue disappears.

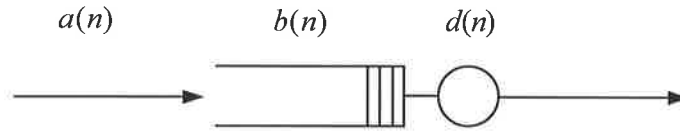


Fig. 4.4 Queuing dynamics at node R

The packet arrival rate $a(n)$ is determined by congestion window size, which in turn is regulated by a PD controller. Therefore $a(n)$ can be written as,

$$a(n) = a(n-1) + k_p \cdot e(n-m) + k_d \cdot (e(n-m) - e(n-m-1)) \quad (4.1)$$

Note that the control error that the PD controller calculates at time n has incurred a round-trip delay, which is denoted as m in equation (4.1).

The control error $e(n)$ is the difference between reference point $r(n)$ and buffer occupancy $b(n)$,

$$e(n) = r(n) - b(n) \quad (4.2)$$

Substituting $e(n)$, equation (4.1) can be written as,

$$\begin{aligned} a(n) = & a(n-1) + k_p \cdot (r(n-m) - b(n-m)) \\ & + k_d \cdot (r(n-m) - b(n-m) - (r(n-m-1) - b(n-m-1))) \end{aligned} \quad (4.3)$$

The z-transform of equation (4.3) can be written as,

$$\begin{aligned} A(z) = & z^{-1} \cdot A(z) + k_p \cdot z^{-m} \cdot (R(z) - B(z)) + k_d \cdot z^{-m} \cdot (1 - z^{-1}) \cdot (R(z) - B(z)) \\ = & z^{-1} \cdot A(z) + z^{-m} \cdot (k_p + k_d \cdot (1 - z^{-1})) \cdot (R(z) - B(z)) \end{aligned} \quad (4.4)$$

The buffer occupancy dynamics can be expressed as,

$$b(n) = \begin{cases} b(n-1) + a(n) - d(n) & \text{if } b(n-1) + a(n) > d(n) \\ 0 & \text{if } b(n-1) + a(n) \leq d(n) \end{cases} \quad (4.5)$$

where $b(n-1)$ is previous buffer occupancy, $a(n)$ is the number of newly arrived packets in an unit time and $d(n)$ is the number of packets departed in an unit time. To make the analysis tractable, the nonlinear equation of (4.5) is simplified as,

$$b(n) = b(n-1) + a(n) - d(n) \quad (4.6)$$

Since the buffer occupancy is above zero most time, the simplified equation (4.6) can still capture the buffer occupancy dynamics at a reasonable degree of accuracy.

The z-transform of equation (4.6) can be written as,

$$B(z) = z^{-1} \cdot B(z) + A(z) - D(z) \quad (4.7)$$

When the bottleneck node serves only one connection, the departure rate $d(n)$ is a constant. Normalise $d(n)$ to one, so $D(z)$ can be expressed as,

$$D(z) = (1 - z^{-1})^{-1} \quad (4.8)$$

Substituting $D(z)$, equation (4.7) can be written as,

$$B(z) = (1 - z^{-1})^{-1} \cdot A(z) - (1 - z^{-1})^{-2} \quad (4.9)$$

Substituting $B(z)$, equation (4.4) can be written as,

$$\begin{aligned} A(z) &= z^{-1} \cdot A(z) \\ &+ z^{-m} \cdot (k_p + k_d \cdot (1 - z^{-1})) \cdot \left(R(z) - \left((1 - z^{-1})^{-1} \cdot A(z) - (1 - z^{-1})^{-2} \right) \right) \end{aligned} \quad (4.10)$$

From equation 4.(10), the closed form of $A(z)$ can be expressed as,

$$A(z) = \frac{z^{-m} \cdot (k_p + k_d \cdot (1 - z^{-1})) \cdot (R(z) + (1 - z^{-1})^{-2})}{1 - z^{-1} + z^{-m} \cdot (1 - z^{-1})^{-1} \cdot (k_p + k_d \cdot (1 - z^{-1}))} \quad (4.11)$$

The inverse z-transform of equation (4.11) yields the closed form of $a(n)$, which is normalised packet arrival rate at the bottleneck node. The throughput ϕ of ST-PD TCP is the average of $a(n)$, that is,

$$\phi = \frac{\sum_{i=0}^N a(i)}{N} \quad (4.12)$$

where N is the time when the TCP connection finishes transmission.

Now we incorporate the factor of wireless transmission error induced packet losses into equation (4.11). As shown in Figure 4.3(b), a single packet loss causes the buffer occupancy to fall below the reference point for a certain period. The duration from the detection of the buffer occupancy falling to buffer occupancy starting to grow again lasts about a round-trip time. The disturbance of buffer occupancy $b(n)$ affects control error $e(n)$ and congestion window $w(n)$ as shown in Figure 4.3(c) and (d). Assuming the buffer occupancy falls below its ideal operating point at time p due to wireless losses, then the control error $e(n)$ can be written as,

$$\begin{aligned}
e(n) &= r(n) - \left(b(n) - \sum_{i=p}^{p+m} \delta(n-i) \right) \\
&= \left(r(n) + \sum_{i=p}^{p+m} \delta(n-i) \right) - b(n)
\end{aligned} \tag{4.13}$$

where $\delta(n)$ is the unit impulse function, and m is the round-trip time. As shown in equation (4.13), the disturbance on buffer occupancy is equivalent to the disturbance on reference point. In ST-PD TCP, the reference point is set to be one packet, that is

$$r(n) = u(n) \tag{4.14}$$

where $u(n)$ is the unit step function. Taking the disturbance into account, the reference point $r(n)$ can be written as,

$$r(n) = u(n) + \sum_{i=p}^{p+m} \delta(n-i) \tag{4.15}$$

The z-transform of equation (4.15) yields,

$$R(z) = (1 - z^{-1})^{-1} + \sum_{i=p}^{p+m} z^{-i} \tag{4.16}$$

Substituting $R(z)$ into equation (4.11) gives,

$$A(z) = \frac{z^{-m} \cdot (k_p + k_d \cdot (1 - z^{-1})) \cdot \left((1 - z^{-1})^{-1} + \sum_{i=p}^{p+m} z^{-i} + (1 - z^{-1})^{-2} \right)}{1 - z^{-1} + z^{-m} \cdot (1 - z^{-1})^{-1} \cdot (k_p + k_d \cdot (1 - z^{-1}))} \tag{4.17}$$

The inverse z-transform of equation (4.17) yields $a(n)$, and the throughput of ST-PD TCP can be calculated through equation (4.12).

4.3.3 Numerical Results

Consider a TCP connection consisted of a wireless link. The round-trip time of this TCP connection is 16ms. The wireless link has bandwidth of 1.5Mbps and it is the bottleneck link. Assuming the packet size is 1000 bytes, it takes the 1.5Mbps link 5.3ms to transmit such a packet. Therefore the round-trip time 16ms is equivalent to 3 packet times, ie. $m = 3$. Assume that a corrupted packet affects the buffer occupancy at time $p = 100$. In ST-PD TCP the gains k_p and k_d vary when the perceived bandwidth-delay product changes, as described in Session 3.4.3.2. In this analysis, the average values of k_p and k_d are used, where $k_p = 0.368$ and $k_d = 0.033$. Substituting m , p , k_p and k_d in equation 4.17 yields,

$$A(z) = \frac{z^{-3} \cdot (0.368 + 0.033 \cdot (1 - z^{-1})) \cdot \left((1 - z^{-1})^{-1} + \sum_{i=100}^{103} z^{-i} + (1 - z^{-1})^{-2} \right)}{1 - z^{-1} + z^{-3} \cdot (1 - z^{-1})^{-1} \cdot (0.368 + 0.033 \cdot (1 - z^{-1}))} \quad (4.18)$$

The inverse z-transform of equation (4.18) can be obtained with the aid of the symbolic toolbox in Matlab[®]. The four poles of equation (4.18) are:

$$\begin{aligned} & -0.5023 \\ & 0.8031 + 0.4155i \\ & 0.8031 - 0.4155i \\ & 0.8961 \end{aligned}$$

Clearly, these four poles are within the unit circle, therefore $a(n)$ is a bounded sequence. The sequence of $a(n)$ is plotted in Figure 4.5.

Figure 4.5 shows the impact of a packet loss on ST-PD TCP connection's packet arrival rate at the bottleneck node. Due to the feedback delay, it takes time for the PD controller to settle down the disturbance.

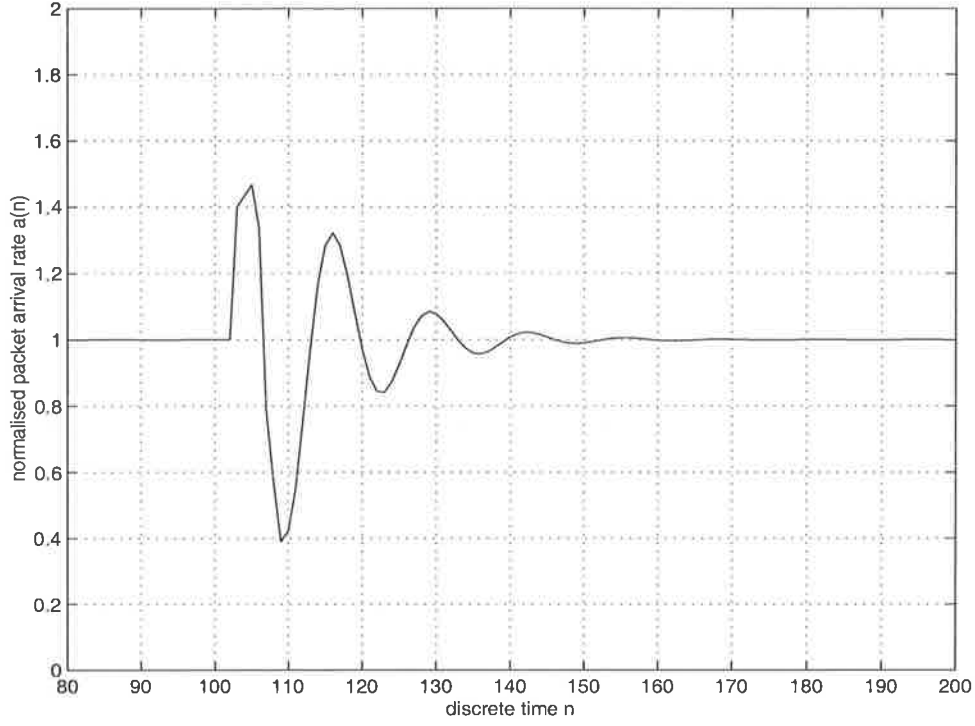


Fig. 4.5 Normalised packet arrival rate

Assume the packet transmission error probability is one in every K packets transmitted. The normalised throughput of ST-PD TCP under such a transmission error condition can be calculated as,

$$\phi = \frac{\sum_{i=0}^K \min\{a(i), 1\}}{K} \quad (4.19)$$

The term $\min\{a(i), 1\}$ in equation (4.19) puts an upper bound on the contribution of packet arrival rate to the normalised throughput. When packet arrival rate $a(i)$ increases to one, it means the arrival rate is equal to the departure rate at the bottleneck node. Increasing $a(i)$ further beyond one only contributes to larger queuing delay rather than higher throughput. According to equation (4.19), the throughput of ST-PD TCP under a range of packet transmission error probabilities is shown in Figure 4.6.

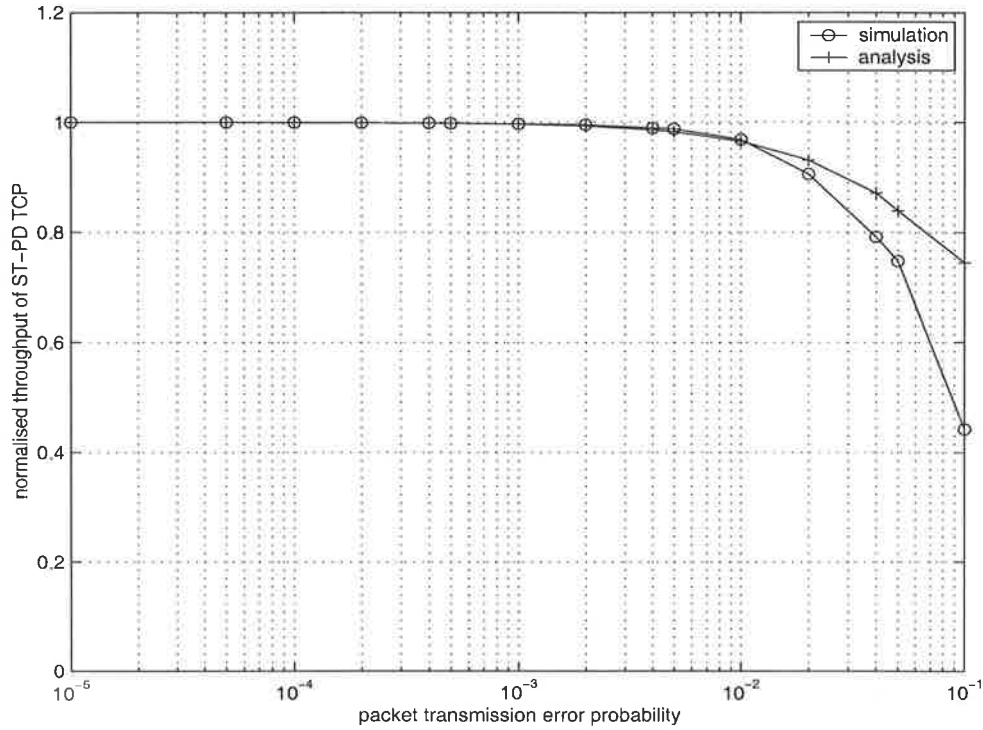


Fig. 4.6 Throughput of ST-PD TCP under various packet error probabilities

From Figure 4.6, it can be seen that the analysis results are consistent with the simulation results in regions where the packet transmission error probability is smaller than 10^{-2} . When the packet transmission error probability increases beyond 10^{-2} , the throughput obtained through analysis is higher than the throughput obtained through simulations, and the higher the packet transmission error probability, the larger the difference between analysis results and simulation results.

The factor that contributes to the inaccuracy of the analytical model under severe wireless loss conditions is as follows. ST-PD TCP congestion control scheme is not purely reliant on the self-tuning PD controller. As described in section 4.2, ST-PD TCP retains the retransmission timer mechanism in the original TCP implementations as a safety net against congestion collapse. In very poor wireless channel conditions, it is possible that a whole window of packets is corrupted over a lossy link, and that is very likely to happen when the congestion window is small and the packet transmission error probability is high. In that situation, ST-PD TCP source nodes remain idle until their retransmission timer expires. After the expiration of the timer, the ST-PD TCP source nodes set their congestion window sizes to

one as do conventional TCP source nodes. During this period, the throughput of the ST-PD TCP connections deteriorates significantly. Unfortunately, this feature cannot be incorporated into the analytical model described in Section 4.3.2, which is based on the transient response analysis of a PD controller.

4.4 Simulation Results

Since some mechanisms of ST-PD TCP such as the retransmission timer are omitted from the analytical model to make the performance analysis tractable, the numeric results derived from the analytical model only captures ST-PD TCP performance approximately. To reveal the behaviour of ST-PD TCP in wireless networking environments more thoroughly, simulation results are presented in this section.

Due to the unique propagation delay characteristics associated with GEO satellite links, TCP performance over terrestrial wireless links and GEO satellite links is covered in a separate subsection.

4.4.1 TCP Performance over Terrestrial Wireless Links

The simulation scenario is set up as follows: A host D downloads a very large file from a remote file server S through a wireless link and a wired link. The link capacities and propagation delays are shown in Figure 4.7. The packet size is 1000 bytes, and the TCP source always has packets to transmit. The node R has a finite buffer that can hold 30 packets.



Fig. 4.7 A TCP wireless connection

Transmission errors over wireless channels usually exhibit bursty characteristics, and interleaving techniques in channel coding are adopted to spread clustered errors. Depending

on the interleaving depth and wireless channel characteristics, the final packet transmission error pattern may or may not be bursty. To investigate the impact of different wireless loss distributions on TCP throughput, simulations were conducted under both uniformly distributed wireless loss and bursty wireless loss conditions.

4.4.1.1 TCP Performance under Uniformly Distributed Wireless Loss Conditions

Under uniformly distributed wireless loss conditions, every packet has the same loss probability over a wireless link. To simulate packet losses which have a uniform distribution, a random number is generated uniformly over the range $[0,1)$ every time a packet is transmitted over the wireless link, and the random number is compared with the packet transmission error probability of the wireless link. If the random number is larger than the packet transmission probability, the packet is passed through, otherwise the packet is dropped.

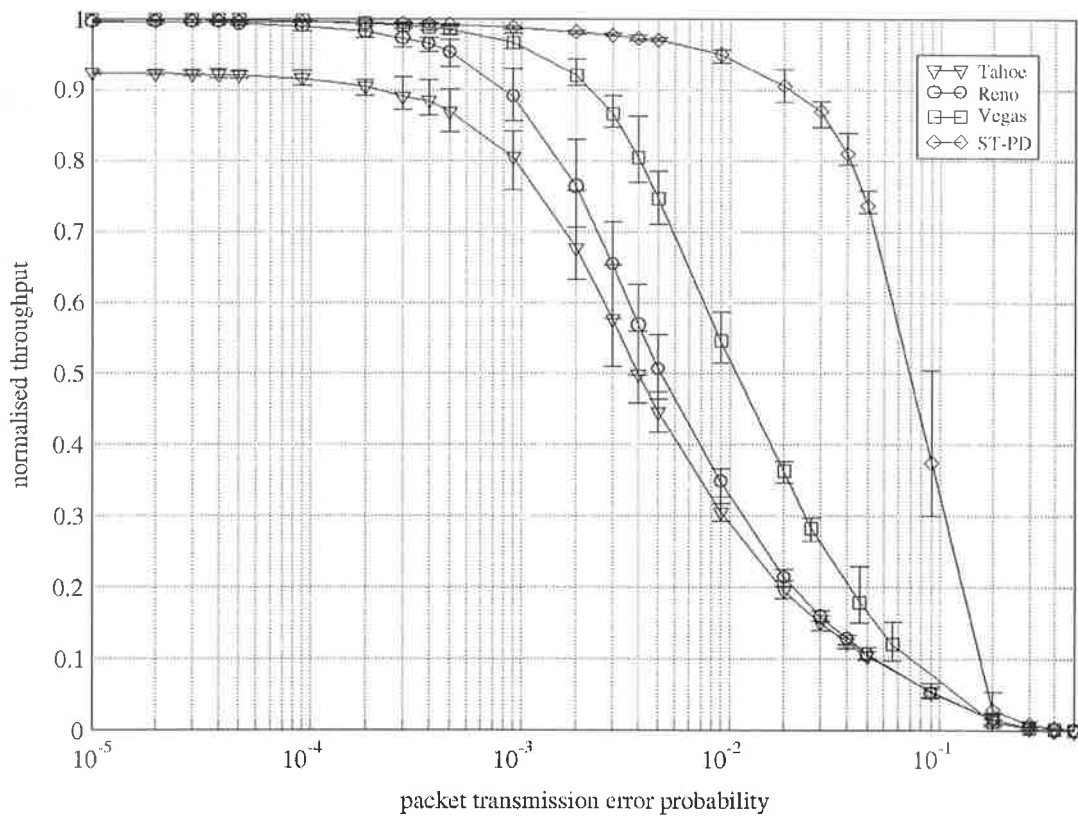


Fig. 4.8 Throughput of different versions of TCP

Figure 4.8 shows the normalised throughput of different versions of TCP under various packet transmission error probabilities through simulations. In these simulations, the packet transmission error pattern conforms to uniform distributions.

It can be seen from Figure 4.8 that the throughput of ST-PD TCP is much higher than the throughput of existing TCP implementations under the same wireless loss conditions. For example, at a packet loss probability of 10^{-2} , ST-PD TCP can achieve 94 percent of the maximum throughput, while the throughput of TCP Vegas has dropped to below 50 percent of the maximum throughput and the throughput of TCP Reno and TCP Tahoe has dropped to around 30 percent of the maximum throughput. At a packet loss probability of 0.1 (ie. one in ten packets is corrupted over wireless link), ST-PD TCP can still retain 40 percent of the maximum throughput, while the throughput of Vegas, Reno and Tahoe is well below 10 percent of the maximum throughput.

To illustrate how uniformly distributed packet losses over wireless links affect the throughput of various versions of TCP, the congestion window traces of Tahoe, Reno, Vegas and ST-PD TCP at the packet error probability of 10^{-2} are shown in Figure 4.9-4.12.

Recall that the bandwidth- RTT_{min} product of the TCP connection shown in Figure 4.7 can hold approximately 31 packets of the size of 1000 bytes. In order to fully utilise the bandwidth of the bottleneck link, the congestion window of TCP must stay above 31 packets. Furthermore a buffer that can hold 30 packets is allocated at the bottleneck node, therefore, the congestion windows of TCP Tahoe and TCP Reno can grow to 61 under an error-free transmission condition.

Figure 4.9 shows the congestion window trace of TCP Tahoe at a packet error probability of 10^{-2} . Clearly, the congestion window of TCP Tahoe is not able to grow to its full potential before a packet loss occurs over the wireless link. The congestion window stays well below 20 packets during most of the simulation time, and the maximum value reaches 37 packets.

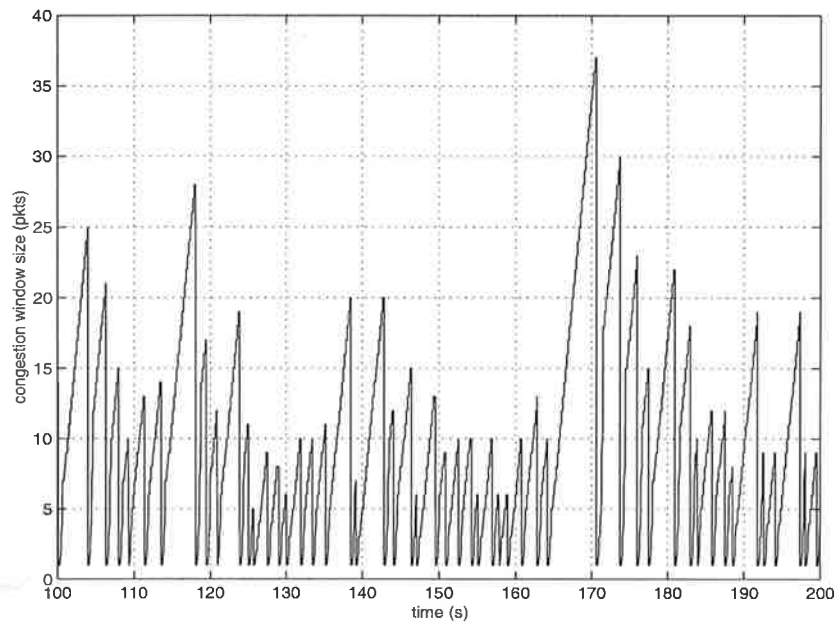


Fig. 4.9 Congestion window trace of TCP Tahoe

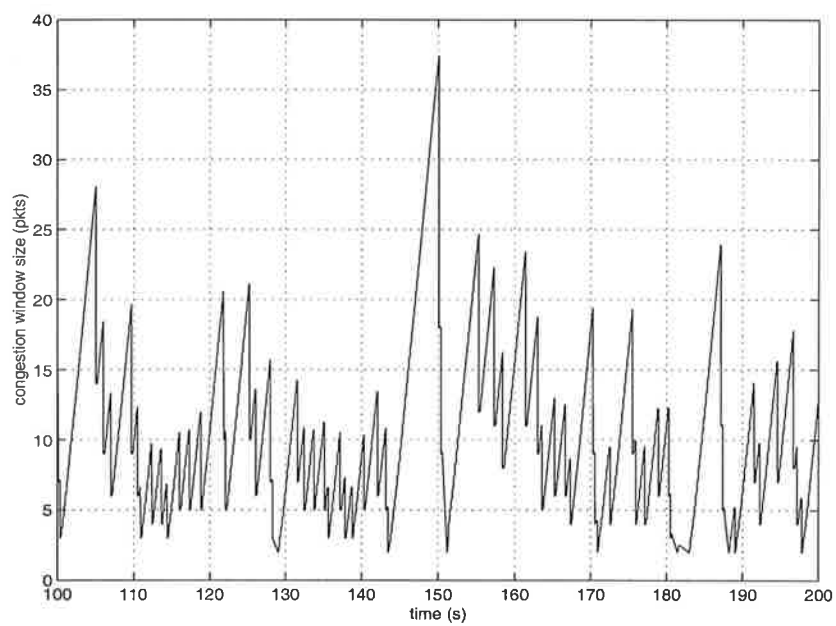


Fig. 4.10 Congestion window trace of TCP Reno

The congestion window for TCP Reno behaves similarly, as shown in Figure 4.10. Comparing Figure 4.10 to Figure 4.9, it can be seen that TCP Tahoe and TCP Reno react to a packet loss differently as expected. TCP Tahoe always reduces its congestion window to one

and switches to slow start phase on experiencing packet losses, while TCP Reno cuts its congestion window by half and stays in congestion avoidance phase if packet losses are detected by duplicate acknowledgments.

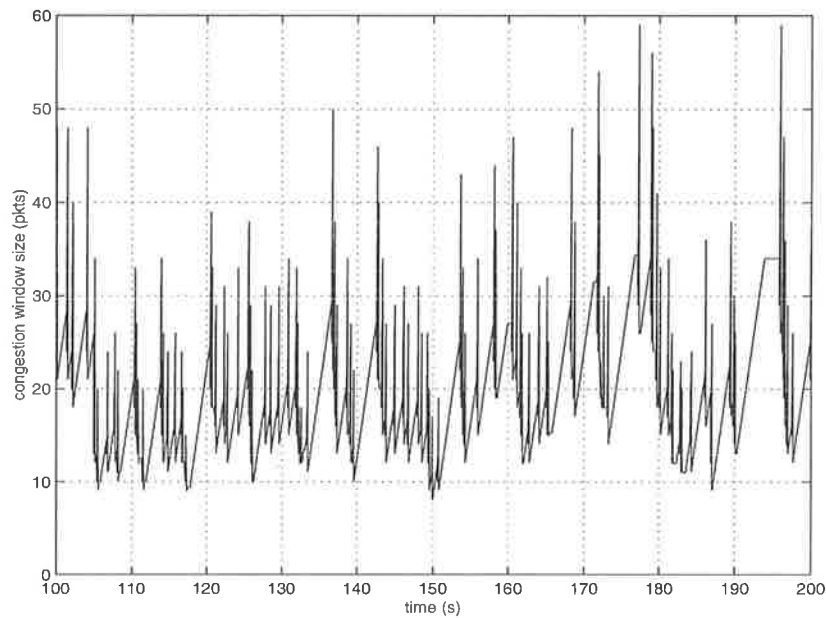


Fig. 4.11 Congestion window trace of TCP Vegas

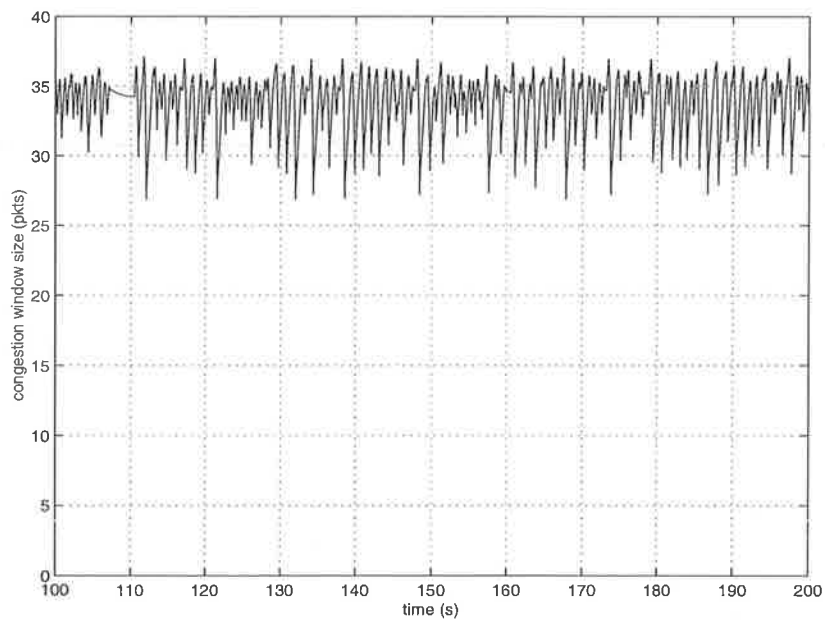


Fig. 4.12 Congestion window trace of ST-PD TCP

TCP Vegas reacts to packet losses in the same way as TCP Reno does, ie. cuts its congestion window by half on experiencing packet losses, as shown in Figure 4.11. However, due to the different congestion window updating algorithm adopted by TCP Vegas in the *congestion avoidance* phase, TCP Vegas can retain slightly larger congestion window, thereby slightly higher throughput than TCP Reno at the same packet transmission error probability.

Figure 4.12 shows the congestion window trace of ST-PD TCP at the packet error probability of 10^{-2} . Unlike Tahoe, Reno and Vegas TCP, which cut their congestion window either to one or by half on experiencing packet losses, the congestion window of ST-PD TCP is regulated by a self-tuning PD controller. Packet losses over a wireless link only cause small disturbances to the self-tuning PD controller and do not result in drastic reduction of the congestion window.

4.4.1.2 TCP Performance under Bursty Wireless Loss Conditions

The clustered packet losses over wireless links are modelled as a two-state process. One state is the error-free state S_0 and other is the error state S_1 . A wireless link transits from one state to the other alternatively. The length of an error burst, which is the number of consecutively corrupted packets, is determined by the duration of state S_1 , and the length of an error-free burst, which is the number of consecutive error-free packets passed through a wireless link, is determined by the duration of state S_0 . The duration of each state is exponentially distributed.

Figure 4.13 shows a clustered packet error distribution that has an average error burst of 8 packets and an overall packet error ratio of 10^{-2} . The width of the bars represents the error burst length. Noting that the packet error distribution is independent of time, packet index rather than time is used in the horizontal axis in Figure 4.13. Figure 4.14 and Figure 4.15 are the traces of congestion window size of Reno TCP and ST-PD TCP respectively under the clustered error distribution as shown in Figure 4.13. To demonstrate the impact of bursty wireless losses on congestion window sizes, the horizontal axial ranges of Figure 4.14 and Figure 4.15 are chosen to be the same as the horizontal axial range of Figure 4.13.

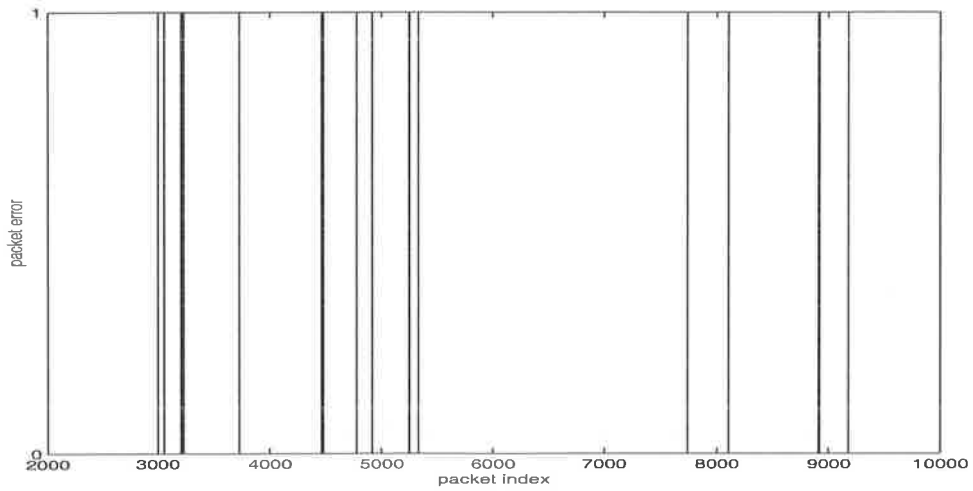


Fig. 4.13 Clustered error distribution in respect of packet index

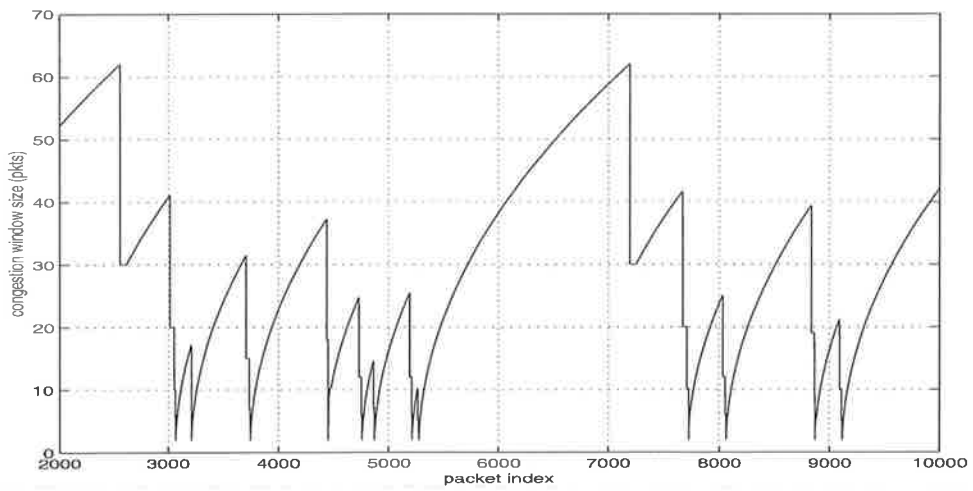


Fig. 4.14 Congestion window trace of TCP Reno in respect of packet index

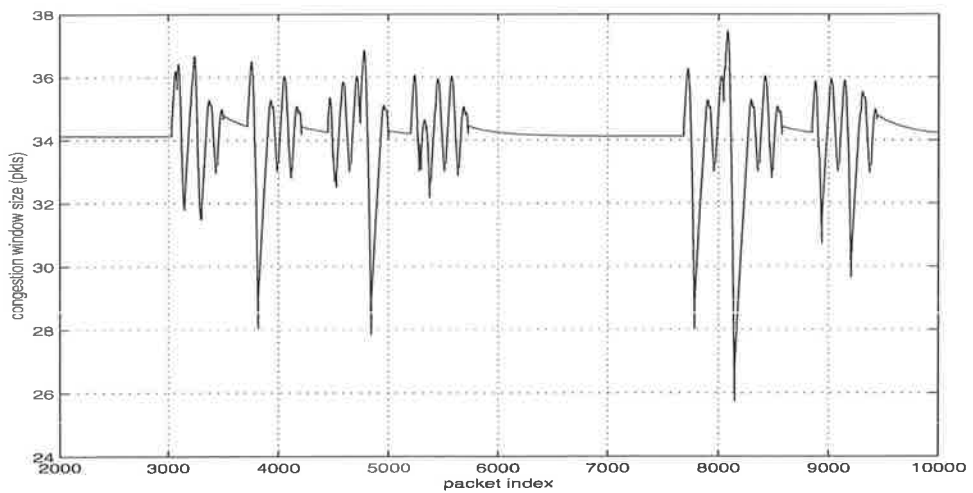


Fig. 4.15 Congestion window trace of ST-PD TCP in respect of packet index

In Figure 4.14 the congestion window grows to over 60 packets twice. One happens around the packet index of 2500 and the other happens around the packet index of 7200. Recall that in this simulation, the bandwidth-RTT_{min} product can contain about 31 packets and a buffer can hold other 30 packets. If the congestion window grows beyond 61 packets, buffer overflow will occur and the congestion window will be cut by half. Therefore in Figure 4.14, there are two congestion window cuts that are due to buffer overflows. The rest of congestion window cuts are due to packet losses over the wireless link.

The effect of bursty losses on the congestion window size of TCP Reno can be seen by comparing Figure 4.14 with Figure 4.10 in which packet errors are uniformly distributed[†]. Clustered errors cause the congestion window to be cut multiple times and eventually each error burst leads to the congestion window being cut to one as indicated in Figure 4.14. By comparison in Figure 4.10, each uniformly distributed packet loss only causes the congestion window to be cut by half.

Through comparing Figure 4.15 with Figure 4.12 in which the packet error pattern conforms to a uniform distribution, it can be seen that there is not much difference in the impact on ST-PD TCP's congestion window size whether a single packet loss or clustered packet losses occur. This is because clustered losses do not cause ST-PD TCP to cut its congestion window multiple times. As long as an acknowledgment is able to reach the ST-PD TCP source, ST-PD TCP can estimate the current network load condition and set its congestion window properly.

Since Figure 4.14 and Figure 4.15 use packet index as the horizontal axes, they cannot reveal how long it takes for Reno TCP and ST-PD TCP to transmit the packets, with starting index of 2001 and ending index of 10000. To demonstrate the congestion window evolution in respect of time, Figure 4.14 and Figure 4.15 are redrawn with time used as the horizontal axes, and shown in Figure 4.16 and Figure 4.17 respectively.

[†] The congestion window evolution patterns look different in Figure 4.10 and Figure 4.14, since time and packet index are used as the horizontal axes in Figure 4.10 and Figure 4.14 respectively.

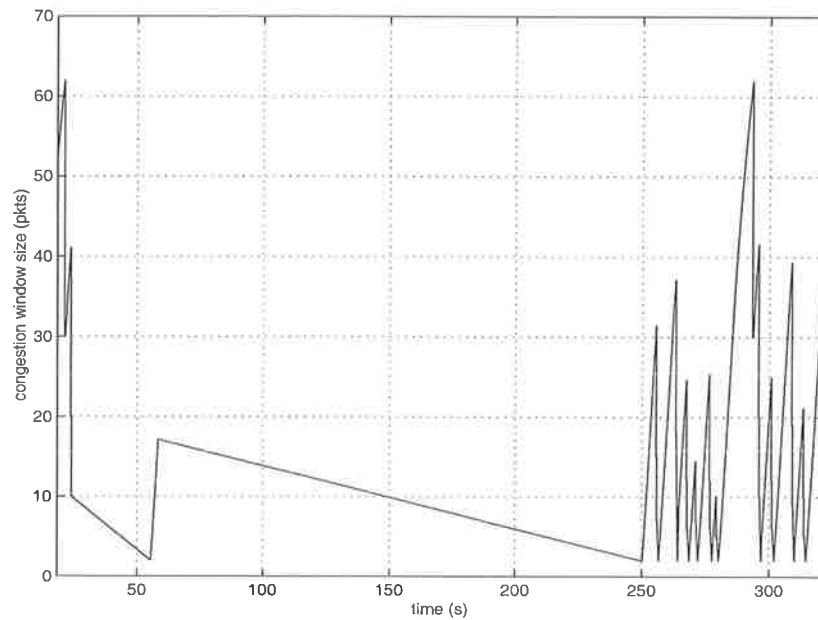


Fig. 4.16 Congestion window trace of TCP Reno in respect of time

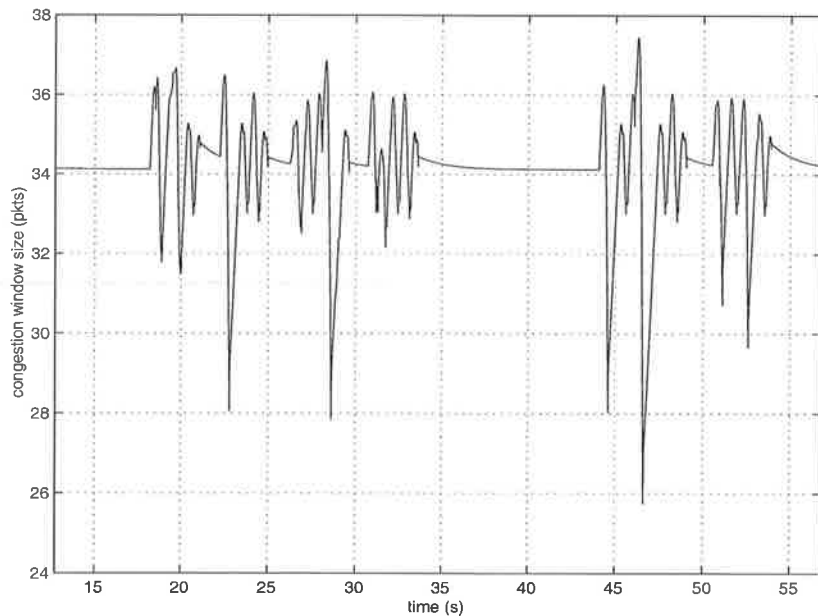


Fig. 4.17 Congestion window trace of ST-PD TCP in respect of time

In Figure 4.16, TCP Reno starts to transmit the 2001st packet at the 18.1097th second and finishes transmitting the 10000th packet at the 322.0880th second. During this period, the TCP Reno source stalls twice due to experiencing bursty losses. One happens around the period of 22-55th second and the other occurs around the period of 56-250th second. During these two

periods, the TCP Reno source is in an idle state waiting for retransmission timeout. The average throughput of TCP Reno during the period of 18.1097-322.0880th second is 0.21Mbps, and the normalised throughput is 0.14. It can be seen from Figure 4.8, that under uniform packet loss distribution the normalised throughput of TCP Reno is about 0.31 at the same overall packet loss ratio of 10^{-2} . Therefore, clustered packet losses can cause the performance of TCP Reno to deteriorate more significantly than uniformly distributed packet losses do.

In Figure 4.17, ST-PD TCP starts to transmit the 2001st packet at the 12.7079th second and finishes transmitting the 10000th packet at the 56.8467th second. The average throughput of ST-PD TCP during this period is 1.45Mbps, and the normalised throughput is 0.96. Referring to Figure 4.8, under uniform packet loss distribution the normalised throughput of ST-PD TCP is about 0.95 at the same overall packet loss ratio of 10^{-2} . Therefore ST-PD TCP performs slightly better under bursty packet loss conditions than under uniformly distributed packet loss conditions. This is because at the same overall packet loss ratio, the error-free period is relatively longer in the bursty loss case than in the uniformly distributed loss case, and clustered packet losses do not affect the congestion window of ST-PD TCP more than a single packet loss does.

4.4.2 TCP Performance over GEO Satellite Links

Beside wireless losses, the large propagation delays incurred over GEO satellite links further deteriorate the achievable throughput of TCP, which was originally designed for terrestrial wired networks. The inefficiencies of TCP operating in GEO satellite networks have long been identified, and standard bodies are now considering several options on the protocol side to alleviate the performance degradation. The following is a list of factors that adversely affect TCP performance over GEO satellite links.

The 64KB limit on TCP congestion window size: The congestion window size of a TCP sender cannot exceed the window size advertised by its receiver. In TCP specifications, the field in TCP headers used to convey the receiver advertised window size is only 16 bits wide, which limits the maximum congestion window size of TCP to 64KB. In terrestrial networks,

the 64KB congestion window size is large enough compared with the relative small bandwidth-delay product of a TCP connection. However the 64KB maximum congestion window size limits the achievable throughput of TCP over one GEO satellite hop to just over 1Mbps.

This problem has been solved by adopting a window scaling option [52]. Using the window scaling option, the receiver advertised window is set to the value in the window field of a TCP header multiplied by a scale factor.

The three-way handshake mechanism in establishing a TCP connection: TCP uses a three-way handshake mechanism to establish connections over connectionless IP networks. Hence, before any user data can be transferred, one or one and a half RTTs must be spent on the connection setup, depending upon whether the TCP sender starts the connection actively or passively. In terrestrial networks, the connection setup delay is negligible. However, in GEO satellite data networks, the hand-shaking latency is disproportionably high for short-lived TCP connections, such as those generated by request/response-oriented transaction applications.

The standard body IETF is considering the proposal — TCP for Transactions (T/TCP) [53], which bypasses the three-way handshake by allowing TCP senders to transmit data in the first segment sent along with the SYN flag. T/TCP can reduce up to 1.5 round-trip delays.

The linear growth of congestion window size during congestion avoidance phase: A TCP congestion window growing process switches from a *slow start* phase to a *congestion avoidance* phase when the slow start threshold (*ssthresh*) is reached or for TCP Reno, a number of duplicate acknowledgments are received. During a *congestion avoidance* phase, a TCP sender increases its congestion window size by one segment for the acknowledgments of the previous whole window of data. That is equivalent to increase one segment every round-trip time. This kind of linear congestion window growing process is quite sluggish in a GEO satellite network where the round-trip time of a TCP connection is at least half a second.

Modern satellite data links are more likely to operate at Ku or even Ka bands, therefore their channel conditions are more vulnerable to rain or fog induced attenuations than those operated at C or UHF bands. Although advanced techniques, such as robust channel coding and high-power transponders, can reduce the BER of satellite links to a certain extent, data corruptions over satellite links cannot be eliminated. Transmission error induced packet losses force conventional TCP connections to enter into a *congestion avoidance* phase, during which the available bandwidth of a satellite link is probed at a very slow pace.

The following simulations were conducted to study the impact of both long propagation delays and wireless losses on TCP throughput over GEO satellite links.

Two LANs are interconnected via a GEO satellite T1 circuit, as shown in Figure 4.18. The source node S transfers an infinite file to the destination node D by using TCP Reno, TCP Vegas and ST-PD TCP respectively. The packet size is 1000 bytes. Each of the file transfers lasts 60 seconds. During the 60 second period, there is only one error burst over the GEO satellite T1 circuit, which corrupts two consecutive packets, namely the 2000th and the 2001st packet. During these simulations, the congestion window size, throughput and the in-order delivered segments sequence number of the three TCP implementations are recorded.

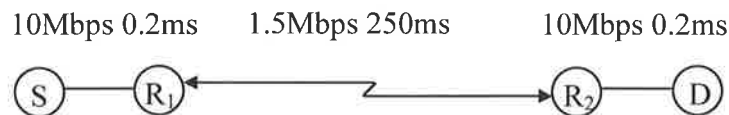


Fig. 4.18 A TCP connection over GEO satellite links

Figure 4.19(a) and (b) show the congestion window trace and throughput of TCP Reno respectively. Since the bandwidth-RTT_{min} product of the TCP connection shown in Figure 4.18 can contain about 94 packets of the size of 1000 bytes, the bandwidth of the T1 circuit cannot be fully utilised if the congestion window size of the TCP source drops to below 94. In Figure 4.19(a), the two consecutive corrupted packets cause the TCP Reno source to cut its congestion window size by half twice to around 33, and force the congestion window growing process to enter into a *congestion avoidance* phase, during which the congestion

window size increases one segment every half a second. It takes more than 30 seconds for TCP Reno to increase its congestion window size to above 94 again after the two consecutive congestion window cuts. Figure 4.19(b) confirms that after experiencing the packets losses, the starving period of the T1 circuit lasts more than 30 seconds.

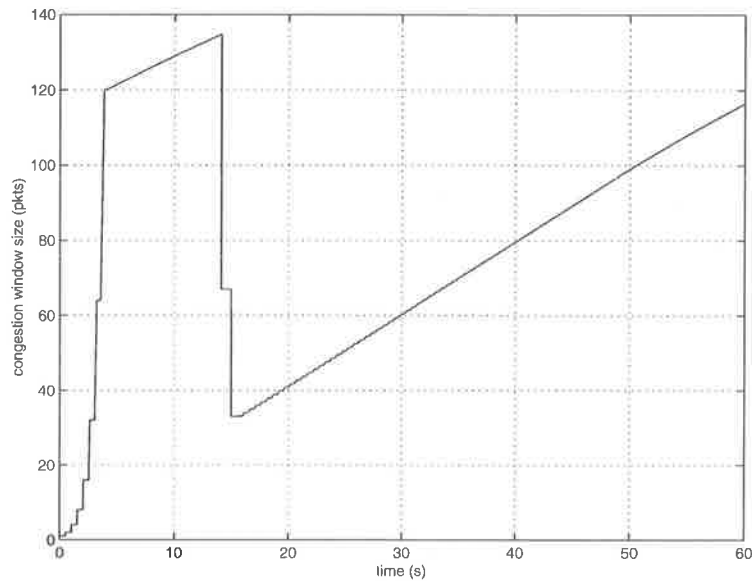


Fig. 4.19(a) Congestion window trace of TCP Reno over GEO satellite links

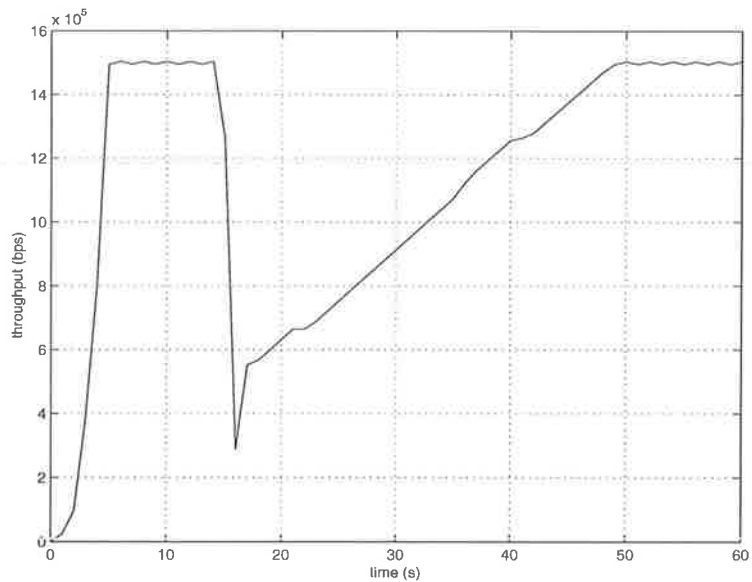


Fig. 4.19(b) Throughput of TCP Reno over GEO satellite links

Figure 4.20(a) and (b) show the congestion window trace and throughput of TCP Vegas respectively. In order to prevent buffer overflows at the beginning of the data transfer, TCP

Vegas doubles its congestion window size every two RTTs during the *slow start* phase, and the *slow start* threshold set by TCP Vegas is very conservative. As a result, TCP Vegas ends the *slow start* phase too early, which is clearly indicated in Figure 4.20(a). When a packet is discarded, TCP Vegas increases its congestion window size, since it experiences a smaller round-trip delay. After receiving three duplicate acknowledgments because of the packet loss, TCP Vegas reduces its congestion window by half immediately just as TCP Reno does.

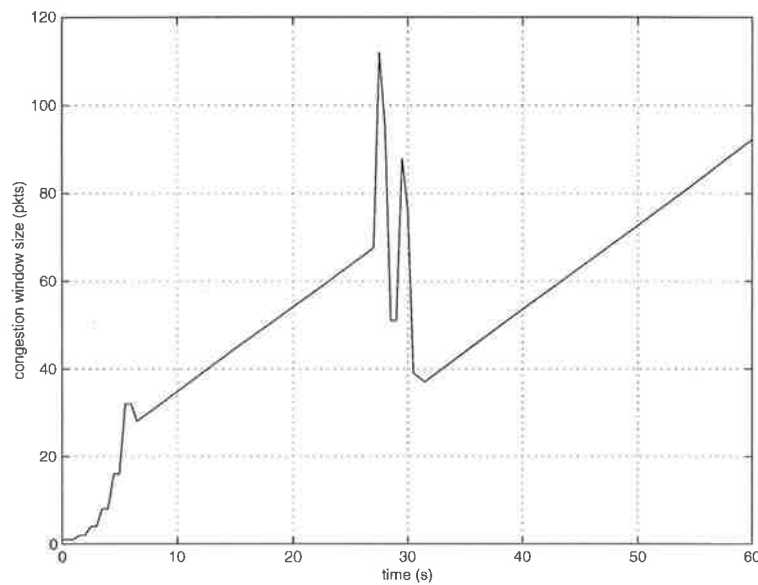


Fig. 4.20(a) Congestion window trace of TCP Vegas over GEO satellite links

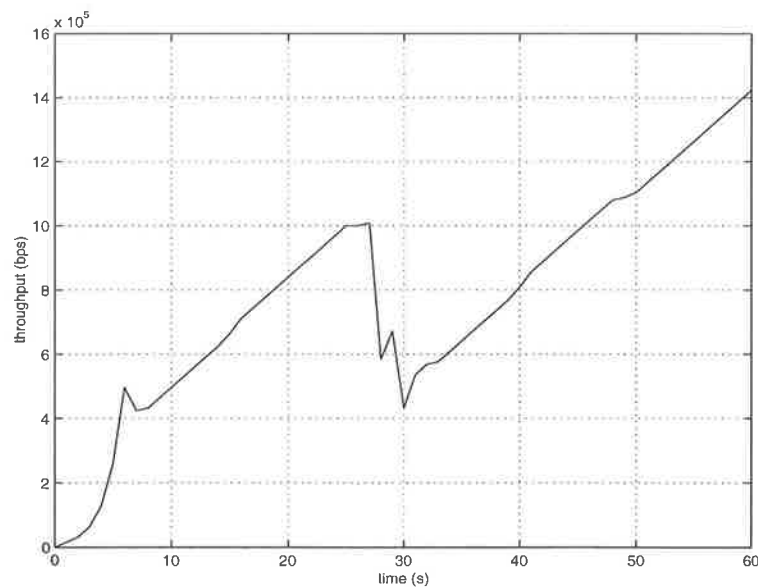


Fig. 4.20(b) Throughput of TCP Vegas over GEO satellite links

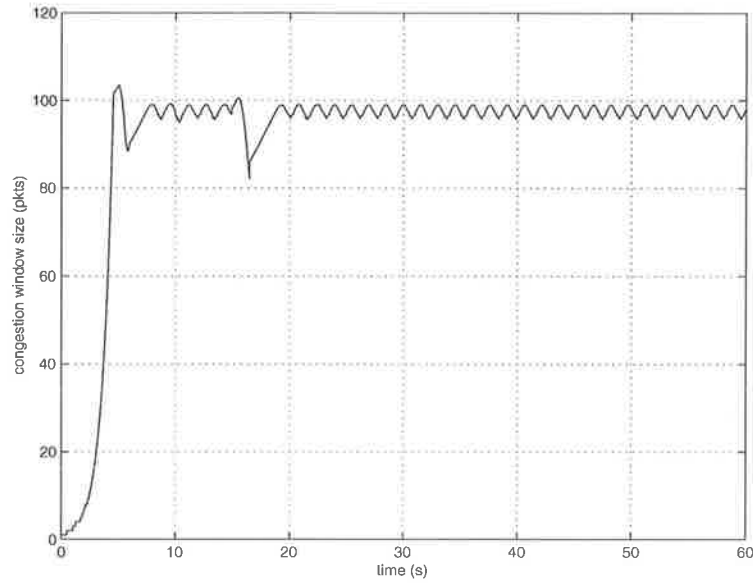


Fig. 4.21(a) Congestion window trace of ST-PD TCP over GEO satellite links

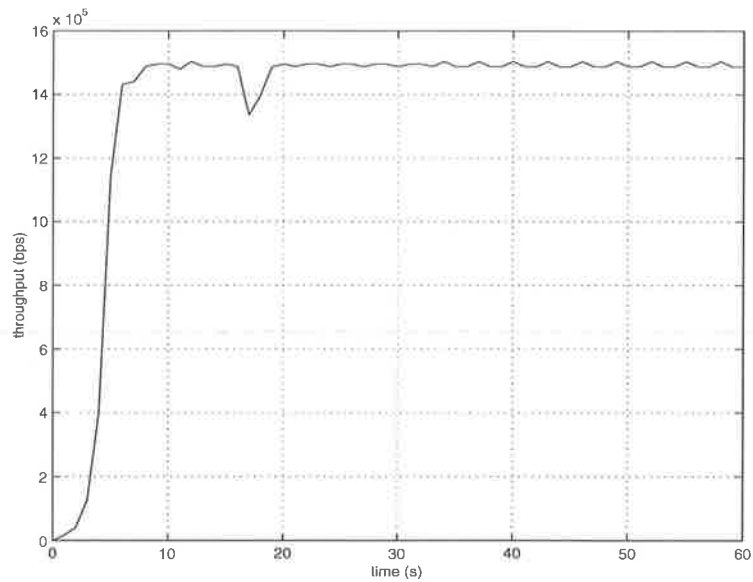


Fig. 4.21(b) Throughput of ST-PD TCP over GEO satellite links

Figure 4.21(a) and (b) show the congestion window trace and throughput of ST-PD TCP respectively. The congestion window size of ST-PD TCP is kept around the bandwidth- $RTT_{\min}$ product plus one level. The impact of the two consecutive packet losses on the congestion window and the throughput of ST-PD TCP is minimal.

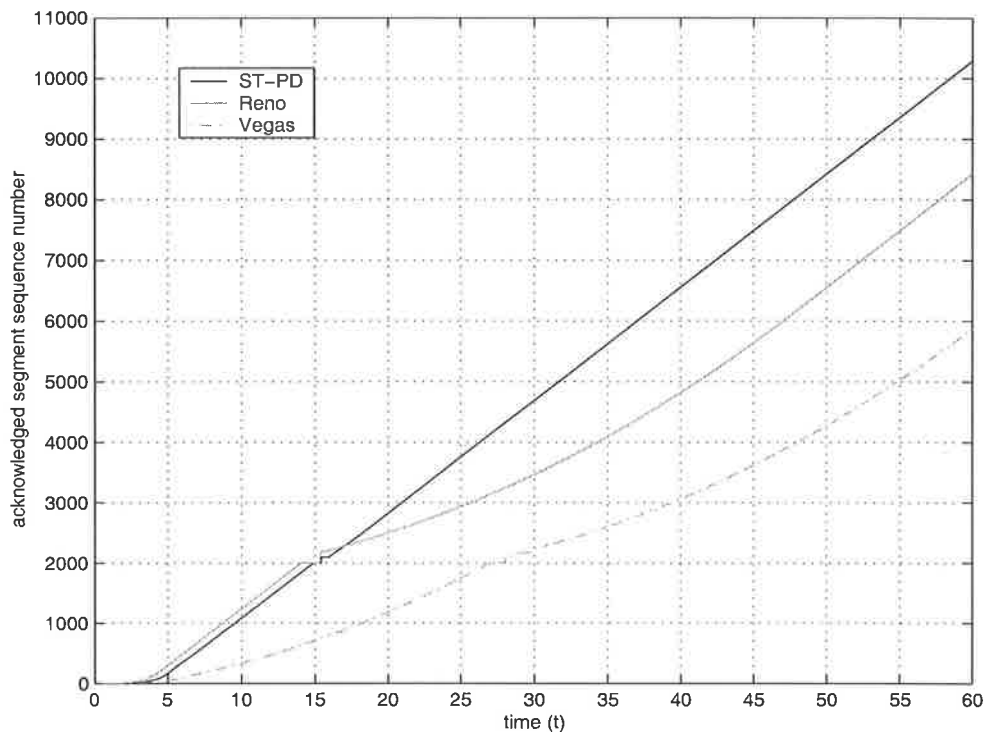


Fig. 4.22 A comparison of acknowledged segment sequence traces

Figure 4.22 compares the traces of the acknowledged sequence number of the three TCP implementations. It can be seen that the acknowledged segment sequence number growth rate of TCP Reno reduces significantly after experiencing packet losses. As a result, the number of in-order segments delivered by TCP Reno during the 60 seconds is 1861 segments less than those delivered by ST-PD TCP. Since TCP Vegas increases its congestion window size only half as much as TCP Reno does during the *slow start* phase, the number of segments transmitted by TCP Vegas is substantially reduced from the beginning. Wireless losses further deteriorate the throughput of TCP Vegas.

4.5 Summary

Massive market demands and rapid technology progress have made wireless telecommunications systems a fast growing part of the global information infrastructure. Terrestrial wireless networks have gained a substantial share in the telephony market in less than two decades, and will continue to rival its landline counterpart in providing data and multimedia services with the introduction of the 3G mobile systems.

The roles of telecommunications satellites have evolved from passively relaying bulk telephony traffic and TV signals across large geographically spanned areas to delivering two-way broadband data services to residential users through advanced on-board packet/cell switching and spot-beam technologies in less than four decades. The inherent features of wireless networking technologies, such as large coverage, multicast capability and rapid infrastructure deployment, will make wireless telecommunications systems a promising solution in providing a wide range of services on demand, anytime, anywhere with relatively low setup cost.

However, before the full potential of wireless data networking systems can be realised, the inefficiency of current TCP implementations over unreliable transmission media must be overcome first. ST-PD TCP addresses this problem by decoupling its congestion control and error control operations. Unlike existing TCP implementations, ST-PD TCP does not rely on detecting packet losses to trigger its congestion control function, therefore it can conduct congestion control and error control separately. This is a beneficial feature for ST-PD TCP to operate more efficiently over lossy wireless links, as wireless losses always cause existing TCP implementations reducing their congestion window sizes unnecessarily.

In this chapter, both analytical and simulation methods have been used to study the performance of ST-PD TCP over a range of lossy wireless links.

The impact of wireless losses on congestion window of ST-PD TCP is modelled as disturbance on the control errors estimated by a PD controller. Through transient response

analysis of the PD controller under disturbance, the throughput of ST-PD TCP over lossy links can be derived.

To make the numeric analysis tractable, the analytical model does not incorporate all detailed aspects of ST-PD TCP. Furthermore, the wireless error pattern is restricted to a uniform distribution in the analytical studies. As a result, the analytical method only captures TCP performance approximately and partially. To fully understand the behaviour of ST-PD TCP in wireless networking environments, simulations were conducted under both uniformly distributed and bursty wireless loss conditions.

Both analytical and simulation results confirm that wireless losses have no significant impact on the congestion window size of ST-PD TCP. In simulations conducted in this study, ST-PD TCP can achieve 95 percent of the maximum throughput under a packet loss ratio up to 10^{-2} . Even when the packet loss ratio deteriorates to 10^{-1} , and such channel conditions are normally regarded as unsuitable for data communications, ST-PD TCP can still retain 40 percent of the maximum throughput.

Simulations also show that compared with uniformly distributed packet errors, clustered packet errors cause more performance deterioration on TCP Reno. This is because an error burst causes TCP Reno to cut its congestion window multiple times and forces TCP Reno to enter into a *congestion avoidance* phase with a very small congestion window. By comparison, the congestion window of ST-PD TCP is not sensitive to single packet loss or clustered packet losses, as long as not all packets in a congestion window are discarded.

Beside wireless losses, the long propagation delays incurred over GEO satellite links further exaggerate the poor performance of current TCP implementations in GEO satellite networks. Once the congestion window is halved after experiencing packet losses, it will take a considerable time for current TCP implementations to ramp up their congestion window size again. On the other hand ST-PD TCP retransmits lost packets without reducing its congestion window drastically after experiencing packet losses. Therefore the utilisation of GEO satellite links under the congestion control of ST-PD TCP is less susceptible to transmission errors.

Compared with current TCP versions, the features mentioned above make ST-PD TCP potentially far more suitable for use over a variety of both reliable and unreliable transmission media.

Chapter 5

The Implementation of ST-PD TCP and Experiment Results

One of the strengths of Unix is its powerful and comprehensive networking capability. Unix is the first operating system that has TCP/IP protocol suite incorporated as its integral part. In 1979, DARPA funded UC Berkeley to implement TCP/IP in its BSD Unix. Following the alpha and beta releases of 4.1BSD, the first stable version of 4BSD, 4.2BSD, was released in 1983. At the same year, DARPA mandated TCP/IP be adopted throughout ARPANET. Since then TCP/IP networking facilities have become an indispensable part of Unix, and sites on ARPANET and Internet in its early stages had been almost exclusively running Unix. In contrast, until the release of Windows95[®] in the fall of 1995, the dominant PC operating systems DOS/Windows3.x had no integrated TCP/IP networking functions, and TCP/IP support was realised by running add-on programs on top of DOS.

Because BSD Unix was the first operating system with TCP/IP integrated and it is widely available in source form, its TCP/IP implementation has been referenced by other operating systems, including Linux, which, in terms of deployment base, is the most popular clone of Unix. Linux was written according to POSIX specifications without using existing Unix code. For licensing and copyright consideration, we chose Linux as the operating system to implement ST-PD TCP.

Following a brief description of data structures of existing TCP/IP implementation in Section 5.1, the implementation of ST-PD TCP in Linux is detailed in Section 5.2. Section 5.3 outlines the experimental environment. The construction of a data link emulator, which facilitates the flexible configuration of the testbed network, is also described in Section 5.3. Experimental results are presented in Section 5.4. The chapter is summarised in Section 5.5.

5.1 Data Structures of TCP/IP Implementation in Linux Kernel

The implementation of the networking subsystem in Linux kernel directly referenced its counterpart in 4BSD. A unified networking interface, known as a *socket*, is used to access multiple protocol suites known as the Address Family (AF) built into a kernel. The kernel architecture of 4BSD networking subsystem is described in the Appendix.

The types of protocol address families integrated into a Linux kernel are decided during the configuration time when the kernel is first built. Every boot time, when the socket interface is initialised, each protocol address family built into the kernel is registered with the socket interface. The initialisation routines for each supported protocol address family are then called. The actual association between the socket interface and a particular protocol address family is decided when an application makes a socket creation system call, during which the name of the address family is passed as a parameter. In this section, data structures related to the TCP/IP operations are described.

Application programs use the system call `sys_socket(int family, int type, int protocol)` to create a new socket. If the address family is supported by the kernel, a new **socket** structure is allocated to represent the BSD socket interface. Further a structure called **sock** is created to represent the underlying address family. The two structures of **socket** and **sock** are cross-linked as shown in Figure 5.1, so that data can be easily passed between the socket interface and the specific address family.

The **sock** structure contains a pointer that points to the structure **proto**, which holds a set of pointers that point to the socket operation routines as shown in Figure 5.1. In **sock**, the data structure of a specific transport layer protocol is held in the *union* **tp_pinfo**, and the data structure of a specific network layer protocol is held in the *union* **net_pinfo**. If TCP is specified when creating the socket interface, the place of **tp_pinfo** will be filled by the structure **tcp_opt**, as shown in Figure 5.2.

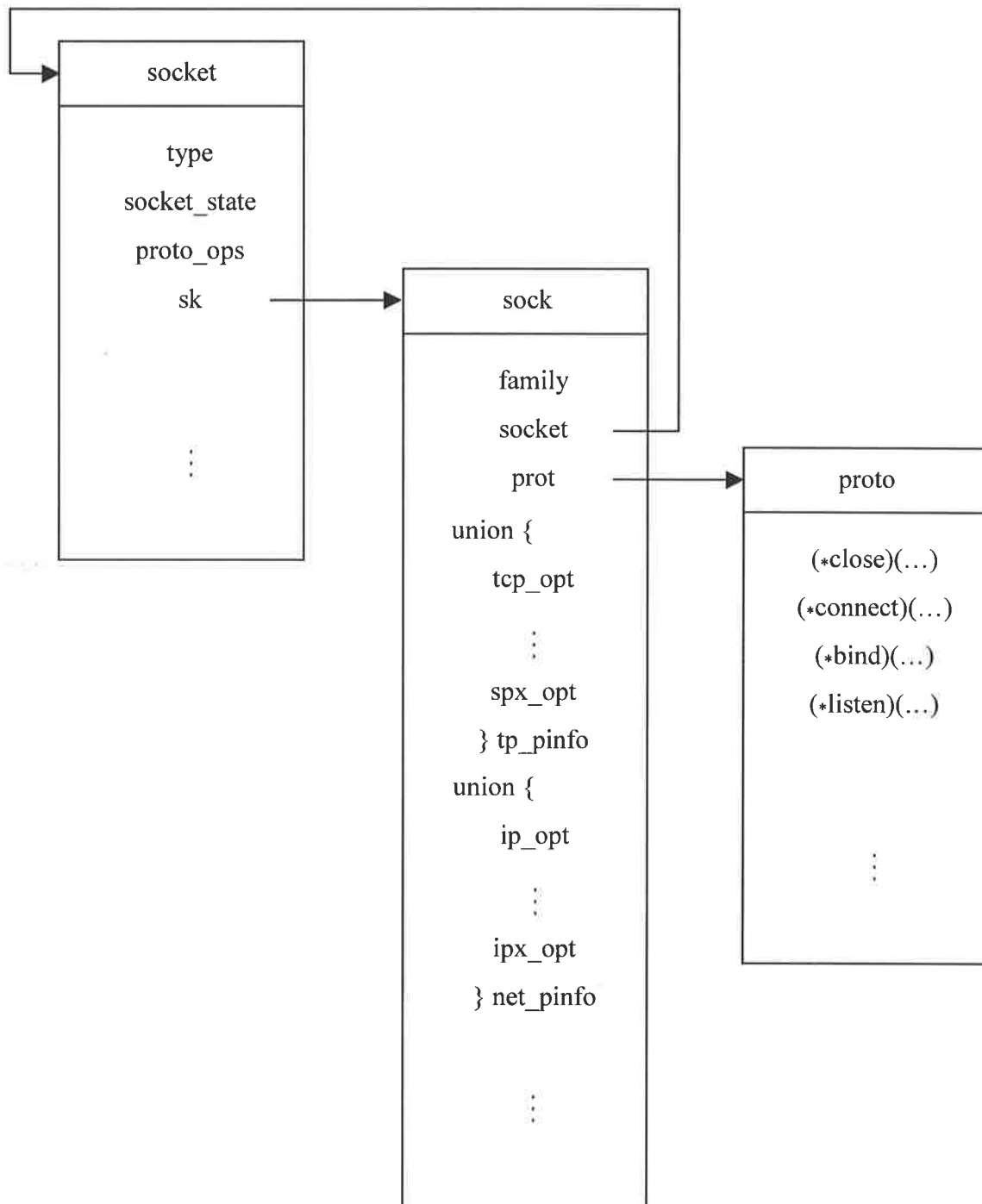


Fig. 5.1 Socket data structures

All state variables specific to TCP operations are kept in the structure **tcp_opt**. The names of these variables are self-explaining. For example, the entry of *snd_nxt* in **tcp_opt** stores the sequence number of the next segment to be transmitted and the entry of *snd_cwnd* in **tcp_opt**

stores the congestion window size of the current TCP connection. The name convention of the TCP state variables are adopted from RFC793 [54] and RFC1122 [55].

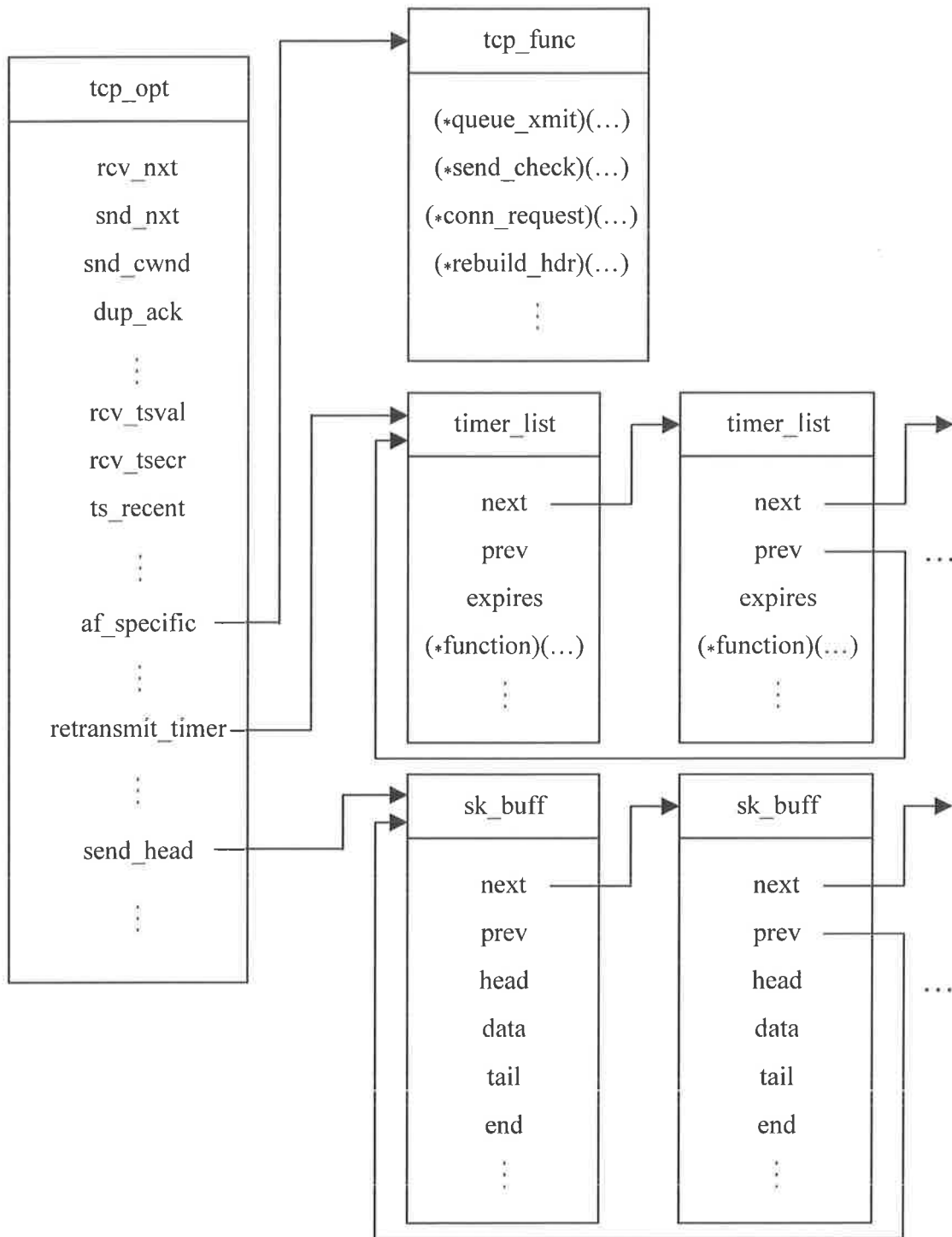


Fig. 5.2 Structure of `tcp_opt`

The entry of *af_specific* in **tcp_opt** points to the structure **tcp_func**, which holds a set of pointers pointed to TCP specific operation routines.

The entry of *retransmit_timer* in **tcp_opt** contains a pointer pointing to the structure **timer_list**, which is a double-linked list. The entry of *expires* in **timer_list** holds the expiration value of the retransmit timer; the entry of *(*function)(...)* in **timer_list** points to the routine that TCP should conduct if the retransmit timer expires.

The start location of user data to be transmitted by TCP is kept in the entry of *send_head* in **tcp_opt**. A pointer is placed in *send_head* pointing to the structure **sk_buff**, which is a double-linked list. The entries of *head* and *end* in **sk_buff** point to the start and end addresses of the buffer respectively; the entries of *data* and *tail* in **sk_buff** point to the start and end addresses of the data, ie. excluding protocol headers and tails.

The double-linked list structure of **timer_list** and **sk_buff** facilitates the arbitrary inserting and removing operations on that list.

5.2 Implementation of ST-PD TCP in Linux Kernel

Compared with existing TCP congestion control schemes, ST-PD TCP needs extra state variables to perform the self-tuning PD control. The following members are added to the structure **tcp_opt** shown in Figure 5.2 to represent these extra state variables:

- *rtt_min* ← the minimum round-trip time the TCP connection experienced so far.
- *err_prev* ← the previous control error, which is needed by the derivative part of the PD controller.
- *kp* ← the control gain of the proportional part of the PD controller.
- *kd* ← the control gain of the derivative part of the PD controller.

The existing congestion control routine in Linux kernel is replaced by the following two routines:

- **void fuzzy_tuning(struct tcp_opt *tp)** ← update the PD control gains $tp \rightarrow kp$ and $tp \rightarrow kd$ of the TCP connection according to the fuzzy tuning method described in subsection 3.4.3.2.
- **void pd_adjust_cwnd(struct tcp_opt *tp)** ← adjust the congestion window size $tp \rightarrow cwnd$ of the TCP connection according to the PD control principle described in subsection 3.4.3.1.

8 bits	signed char __s8
	unsigned char __u8
16 bits	signed short __s16
	unsigned short __u16
32 bits	signed int __s32
	unsigned int __u32
64 bits	signed long __s64
	unsigned long __u64

Tab. 5.1 Data types supported by Linux kernel

The implementation of ST-PD TCP in Linux kernel is relatively straightforward, except that the following issues must be addressed first:

- The PD control function described in section 3.4 is built upon a set of floating-point arithmetic operations. However, there is no floating-point data type in the kernel space. The data types that Linux kernel supported are integers (including char), as listed in table 5.1. Therefore floating-point arithmetic operations must be supported in the kernel space before ST-PD TCP can be implemented.
- The effectiveness of ST-PD TCP operations depend on the accuracy of the measured round-trip time (RTT). However, the granularity of the timestamp, from which the RTT is calculated, is 10ms for Linux running on Intel X86 platform. Such a temporal granularity is too coarse for ST-PD TCP to function effectively in highspeed or local area networking environments. For example, in a 10Mbps LAN, the round-trip time

for forwarding a 1000-byte-sized packet and returning its 40-byte-sized acknowledgment is less than 6ms. Obviously in such networking environments, the granularity of the timestamp is not fine enough to measure the actual RTTs.

The following subsections describe the ways that floating-point arithmetic operations are enabled and the timestamp resolution is refined in detail.

5.2.1 Software Implementation of Floating-point Arithmetic Operations

There are two ways to enable floating-point arithmetic operations in Linux kernel. One approach is to embed in Linux kernel the assembly code, which directly access the Floating-point Processing Unit (FPU). The other way is software implementation of floating-point arithmetic operations. The direct FPU accessing solution has the following drawbacks:

- ❑ Not all platforms on which Linux runs have a FPU.
- ❑ Even if different platforms do have FPUs, the assembly programming languages for distinct FPU architectures are unlikely to be compatible.

Based on the portability consideration, the software implementation approach is adopted in this experiment.

To provide the necessary floating-point arithmetic operations for ST-PD TCP, a tailored version of *SoftFloat* [56] is integrated into the Linux kernel. *SoftFloat* is a software implementation of arithmetic operations that conforms to the IEC/IEEE Standard for Binary Floating-point Arithmetic, and it is distributed in source form.

There are four floating-point precision types defined in *SoftFloat*: **float32** (single precision, stored in a 32-bit word), **float64** (double precision, stored in a 64-bit word), **float80** (extended double precision, stored in a 80-bit word), and **float128** (quadruple precision, stored in a 128-bit word). For the purpose of providing floating-point arithmetic operations for ST-PD TCP, single precision is adequate. Therefore only the floating-point type **float32** is added to the Linux kernel.

A subset of arithmetic operations implemented in *SoftFloat* are integrated into the Linux kernel, which are listed as follows:

- **float32** *int32_to_float32*(__s32) ← Converting a 32-bit signed integer to a 32-bit floating-point format.
- **__s32** *float32_to_int32*(float32) ← Converting a 32-bit floating-point format to a 32-bit signed integer.
- **float32** *float32_add*(float32, float32) ← Add operation in 32-bit floating-point formats.
- **float32** *float32_sub*(float32, float32) ← Subtract operation in 32-bit floating-point formats.
- **float32** *float32_mul*(float32, float32) ← Multiply operation in 32-bit floating-point formats.
- **float32** *float32_div*(float32, float32) ← Divide operation in 32-bit floating-point formats.
- **float32** *float32_round_to_int*(float32) ← Rounding to the nearest integer value but still in a 32-bit floating-point format.
- **__s32** *float32_lt*(float32, float32) ← *Less than* comparison operation between two 32-bit floating-point formats.
- **__s32** *float32_le*(float32, float32) ← *Less than or equal* comparison operation between two 32-bit floating-point formats.

The last two comparison operations are needed by the fuzzy logic controller.

5.2.2 Refining Timestamp Resolution

In Unix (including Linux) operating systems, one timer in the hardware is configured to generate regular interrupts. The period of this timer becomes the basic time unit on which the

kernel operates. Every time the timer expires, the kernel updates its software clock and dispatches the next due event. The value of the software clock is stored in the global variable *jiffies* that can be accessed by all kernel routines as a temporal reference.

In the Linux operating systems running on Intel X86 platform, the timer is programmed to interrupt 100 times per second, therefore the granularity of *jiffies* is 10 milliseconds. Every time a TCP source node transmits a packet, it reads the current value of *jiffies* and uses it as the timestamp of the packet. When the TCP destination node receives the packet, it sends an acknowledgment back with the original timestamp included. Once the TCP source node receives the acknowledgment, it read the current value of *jiffies* again, and calculates the current round-trip time of the TCP connection.

In existing TCP implementations, the calculated round-trip time is used to determine the timeout value for the retransmit timer, therefore the resolution of the round-trip time is not crucial. However, in ST-PD TCP, the measured round-trip time is used by a self-tuning PD controller to adjust the congestion window size, thereby the coarse-grained round-trip time will affect the accuracy of the congestion control function.

To improve the measured round-trip time resolution, time read from the real-time clock rather than *jiffies* is used to timestamp outgoing packets in ST-PD TCP.

The routine **void do_gettimeofday(struct timeval *t)** in the Linux kernel reads the real-time clock and stores the reading in the structure pointed by *t*. The structure **timeval** has two members: *tv_sec* and *tv_usec*. After the routine is called, *t->tv_sec* stores the seconds passed since the beginning of 1970, and *t->tv_usec* stores the microseconds. To fit in the 32-bit space allocated for storing timestamp, *t->tv_sec* and *t->tv_usec* are combined into a 32-bit unsigned integer *ts_now*,

$$ts_now = (t \rightarrow tv_sec \& 0xFFFF) \cdot 100000 + t \rightarrow tv_u\ sec / 10 \quad (5.1)$$

Equation (5.1) shows that only the most recent 65535 (0xFFFF) seconds in *t->tv_sec* is used and the last digit in *t->tv_usec* is discard in calculating *ts_now*. Therefore the timestamp constructed this way has a resolution of 10 microseconds and wraps very 65535 seconds

(18.2 hours). Restrained by the 32-bit space for the timestamp storage, the above measure is aimed to strike a balance between the two contradictory requirements of high resolution and long wrapping period.

Through the procedures described above, the resolution of measured round-trip time increases 1000 times to 10 microsecond in ST-PD TCP. Despite the fact that the new timestamp wraps only once every 18.2 hours, anti-wrapping measures were implemented to handle the rare event of timestamp wrapping.

5.3 Experiment Environments

5.3.1 Testbed Network Configuration

The testbed network topology is shown in Figure 5.3, which consists of three nodes and two links. The configuration of these nodes and links is listed as follows.

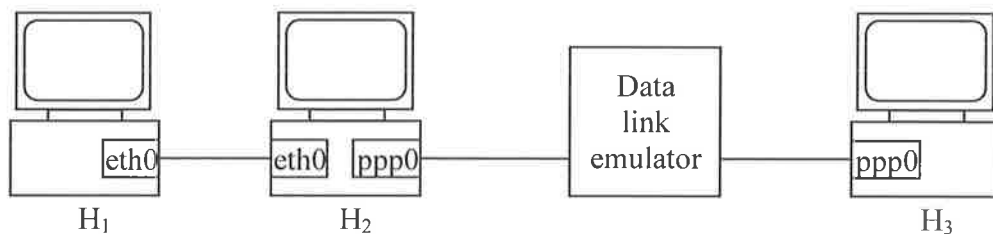


Fig. 5.3 Experiment setting

- **Hardware:** Each node has an Intel Celeron[®] 500Mhz CPU and 64MB memory. The nodes H₁ and H₂ are connected via a pair of RealTek[®] 8139 fast ethernet cards, which have a maximum transmission speed of 100Mbps. The nodes H₂ and H₃ are connected through a serial point-to-point link. The serial point-to-point link has a maximum transmission speed of 115.2Kbps when using a direct NULL-Modem connection. Restricted by data processing capacity of the microcomputer on which the data link emulator is built, the transmission speed of the serial point-to-point link is limited to 38.4Kbps when the data link emulator is used.

- **Software:** Each host runs two versions of Linux. One is the standard release of Linux-2.2.17 in which the TCP implementation adopts the Reno congestion control mechanism. The other is the experimental version based on Linux-2.2.17 in which the ST-PD TCP described in Section 3.4 is implemented. The kernel of Linux-2.2.17 includes the device driver for RealTek® 8139 fast ethernet card. The Point-to-Point Protocol (PPP) is used as the link layer protocol to carry IP packets over the serial link. The node H₂ has the IP forward feature enabled, so that it can act as a router as well as an end node.

5.3.2 Methods for Experiment Data Collection

The widely used TCP traffic capture tool *tcpdump* [57] is used to collect raw experiment data. *tcpdump* simply records the time every packet passed the node on which *tcpdump* is running and the protocol header of the passing packet. A sample TCP traffic trace file dumped by *tcpdump* is shown in Figure 5.4.

```
18:02:52.507306 H1.3490 > H3.1053: P 40261:40361(100) ack 1 win 32704 <nop,nop,timestamp 123558 1156494> (DF)
18:02:52.547241 H3.1053 > H1.3490: . ack 37261 win 31900 <nop,nop,timestamp 1156498 123437> (DF)
18:02:52.547312 H1.3490 > H3.1053: P 40361:40461(100) ack 1 win 32704 <nop,nop,timestamp 123562 1156498> (DF)
18:02:52.587242 H3.1053 > H1.3490: . ack 37361 win 31900 <nop,nop,timestamp 1156502 123441> (DF)
18:02:52.587306 H1.3490 > H3.1053: P 40461:40561(100) ack 1 win 32704 <nop,nop,timestamp 123566 1156502> (DF)
18:02:52.627243 H3.1053 > H1.3490: . ack 37461 win 31900 <nop,nop,timestamp 1156506 123445> (DF)
```

Fig. 5.4 A sample TCP traffic trace collected by *tcpdump*

The first line of Figure 5.4 shows that at the time of 18:02:52.507306 a packet was transmitted from the port 3490 of the node H₁ to the port 1053 of the node H₃. The payload size was 100 bytes which begins with the sequence number of 40261. The second line of Figure 5.4 shows that at the time of 18:02:52.547241 an acknowledgment arrived from the port 1053 of the node H₃ to the port 3490 of the node H₁. The acknowledgment indicated that the node H₃ has received in-order data up to the sequence number of 37260.

The window size of 31900 is advertised by the receiving side to the sending side, which indicates the available buffer at the receiving side. The number of data that the sending side can transmit without further acknowledgments is limited by the smaller one of the two windows: the congestion window *cwnd* maintained at the sending side and the window *win* advertised by the receiving side[†].

The first element of each line is the time recorded by *tcpdump* based on reading the clock of the host on which *tcpdump* is running. There are also two timestamp entries each line, which are extracted from the TCP header of each passing packet. For example, the timestamp of 123588 in the first line of Figure 5.4 is the kernel jiffy at which the packet is transmitted. The second timestamp of 1156494 in the first line is a turn-around (echoed) timestamp value in its previously received packet.

All experiment results presented in this chapter were extracted from TCP traffic traces collected by *tcpdump*.

5.3.3 Implementation of Data Link Emulator

To test ST-PD TCP performance over a wide range of networking environments, a data link emulator is preferred to enable the propagation delay and transmission error characteristics of a data link to be set to a range of values. The advantage of using a data link emulator is that it can behave like a physical link in real-time, but without the expense and inconsistency that are inherent with testing on real networks.

A data link emulator is constructed by using a single-chip microcomputer M30620FP and a 4-channel RS-232 drivers/receivers chip MAX233A, as shown in Figure 5.5.

[†] In reality, a receiver advertised window size is always larger than a sender maintained congestion window size. In the cases involved very large delay-bandwidth product of TCP connections, such as TCP over GEO satellite links or TCP over extreme high-speed links, TCP has option of scaling the receiver advertised window size to ensure that TCP throughput is only subjected to the congestion window maintained at senders.

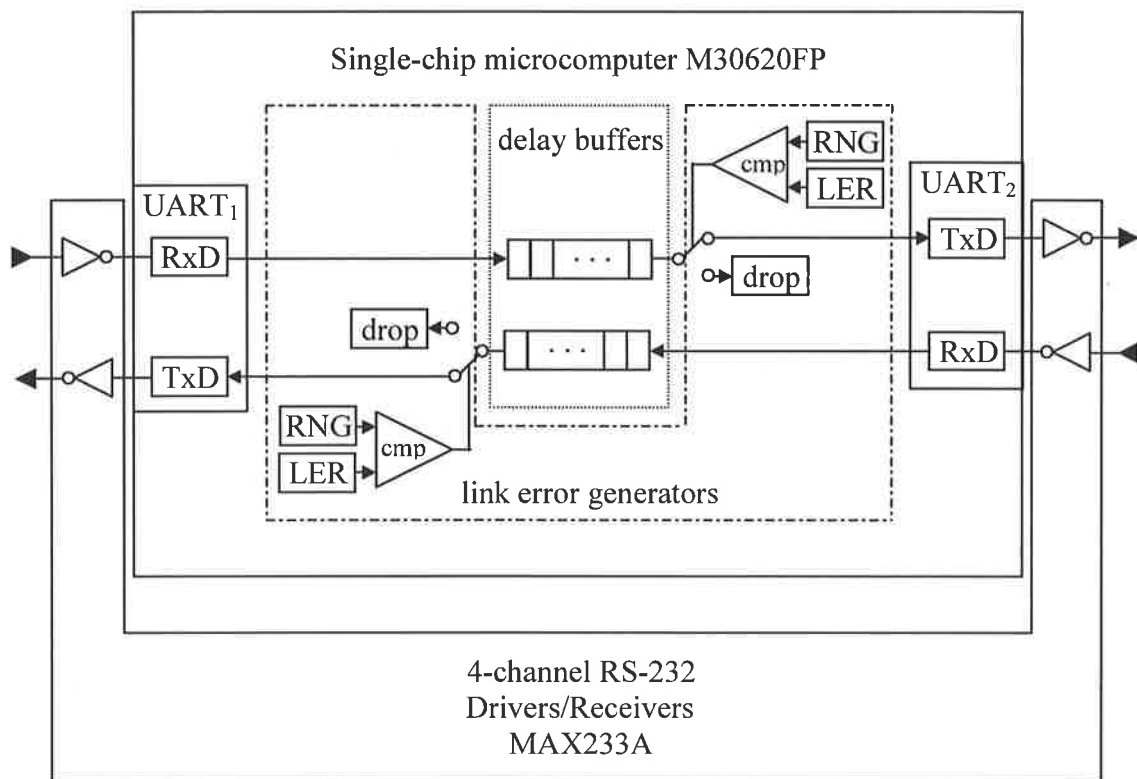


Fig. 5.5 Block diagram of data link emulator

The chip MAX233A is used to convert the voltage levels between CMOS and RS-232 interfaces. The functions of link delays and link errors are implemented in the single-chip microcomputer M30620FP. The data link emulator shown in Figure 5.5 can be divided into three functional blocks:

- 1) The UART₁ and UART₂ in M3060FP and the 4-channel RS-232 drivers/receivers in MAX233A constitute two full-duplex RS-232 interfaces.
- 2) Two blocks of on-chip RAM of the single-chip microcomputer M30620FP are allocated as the delay buffers, one for each direction.
- 3) Two independent Random Number Generators (RNGs) form the cores of the link error generators, one for each direction. The generated random number is compared with the preset Link Error Rate (LER) to decide whether to send the data to the outgoing TxD buffer or to drop it.

The following describes the organisation of the delay buffer and the construction of the random number generator used in the data link emulator in detail.

The organisation of the delay buffer: The single-chip microcomputer M30620FP has 10K bytes of on-chip RAM, which are mapped to the address space ranging from 0x00400 to 0x02BFF when configured into the single-chip operation mode, as shown in Figure 5.6 [58]. The Special Function Register (SFR) area accommodates the control registers for peripheral devices, such as serial I/Os. Among the 10K bytes of RAM, the area from 0x00400 to 0x004FF is allocated to store general variables used by the data link emulator, the area from 0x00500 to 0x0187F is allocated as the delay buffer for one direction and the area from 0x01880 to 0x02BFF is allocated as the delay buffer for the other direction.

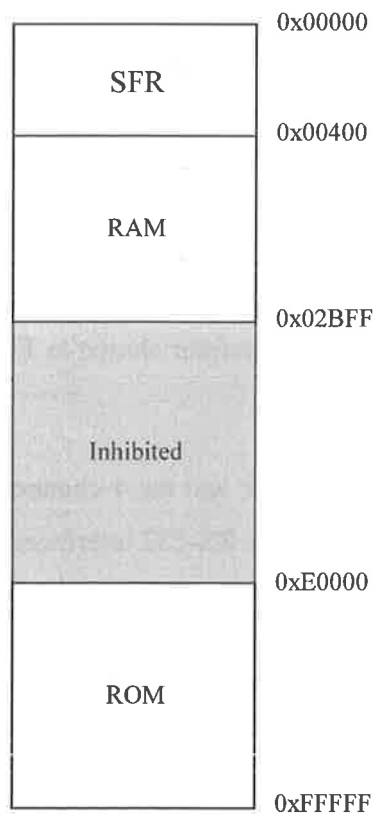


Fig. 5.6 Address area map of internal RAM and Flash ROM of M30620FP

The most intuitive way of emulating link propagation delay is shown in Figure 5.7, in which the data from the incoming RxD buffer is put into the memory B_0 , the data in the memory B_n is put into the outgoing TxD buffer, and a timer is programmed to trigger the shifting operation of the data in the delay buffer periodically. Assuming the period of the timer is T , then the emulated propagation delay is $n \cdot T$.

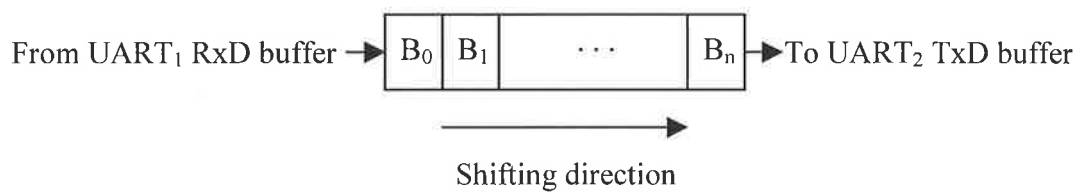


Fig. 5.7 Link propagation delay emulation

However, the data shifting described above requires the completion of n memory copy operations (ie. $B_{n-1} \rightarrow B_n$, $B_{n-2} \rightarrow B_{n-1}$, ..., $B_0 \rightarrow B_1$) within T . As n increases, the microcomputer may not be able to complete these operations within that period.

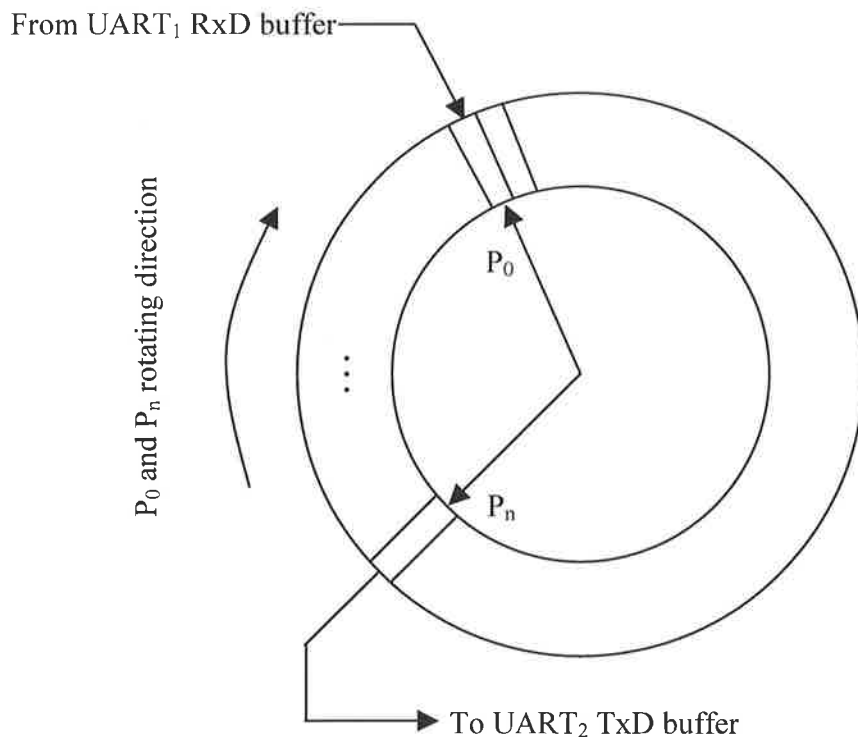


Fig. 5.8 Circular delay buffer structure

To overcome this, a circular delay buffer structure was adopted as shown in Figure 5.8, in which the data from incoming RxD buffer is put into the memory pointed by P_0 , the data in the memory pointed by P_n is put into the outgoing TxD buffer. Instead of shifting the data in the delay buffer, the two pointers P_0 and P_n are shifted one position every time the timer interrupts. During each of the pointer shifting operations, the allocated memory block boundaries are checked to ensure that the lowest and highest physical addresses of the memory block wrap correctly.

The construction of the random number generator: as there is no built-in random number generator circuit in the microcomputer M30620FP, the data link emulator uses a software implementation of a family of random number generators called Linear Congruential Generator (LCG).

All random number generators are based upon specific mathematical algorithms, which are repeatable and sequential [59]. LCG is most commonly used for generating random[†] integers based on the following equation,

$$x_{n+1} = (a \cdot x_n + c) \mod m \quad (5.2)$$

In equation (5.2), a , c and m are constant integers. Together with the initial seed x_0 , the entire sequence is determined by the multiplier a , the additive constant c , and the modulus m . Following the convention described in [60], the random number sequence generated this way is denoted as $LCG(a, c, m, x_0)$.

The integer numbers generated by $LCG(a, c, m, x_0)$ fall into the range $[0, m)$, and the maximum period of the sequence is m . A ideal pseudorandom number sequence should possess the properties of long period and uniformity [59].

Following the recommendation in [61], $LCG(a, c, m, x_0)$, where $a = 16807$, $c = 0$, $m = 2^{31} - 1 = 2147483647$, which is a prime number, and $x_0 = 1$, are used by the data link emulator to generate random numbers.

[†] Strictly, the numbers generated by all random number generators are pseudorandom, as they have the appearance of randomness, but nevertheless exhibit repeatable patterns.

The pseudorandom number sequence generated by $LCG(16807,0,2^{31}-1,1)$ has a period of 2,147,487,647. Figure 5.9(a) shows the autocorrelation function of the first 20,000,000 numbers of this sequence. For comparison purposes, Figure 5.9(b) shows the autocorrelation function of the first 20,000,000 numbers generated by the uniform random number function in Matlab®. The mean of the first 20,000,000 normalised numbers is 0.4991 and the variance is 0.0831, which approach the theoretical values of $\frac{1}{2}$ and $\frac{1}{12}$.

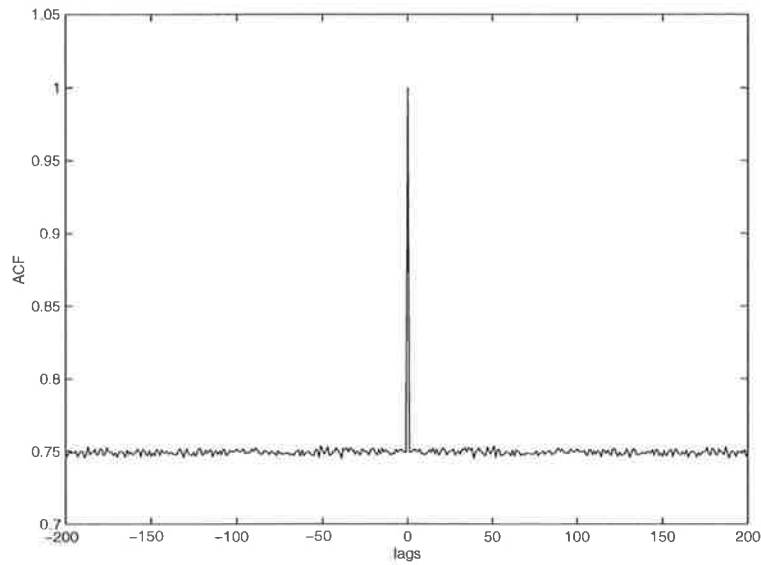


Fig. 5.9(a) ACF of $LCG(16807,0,2^{31}-1,1)$ sequence

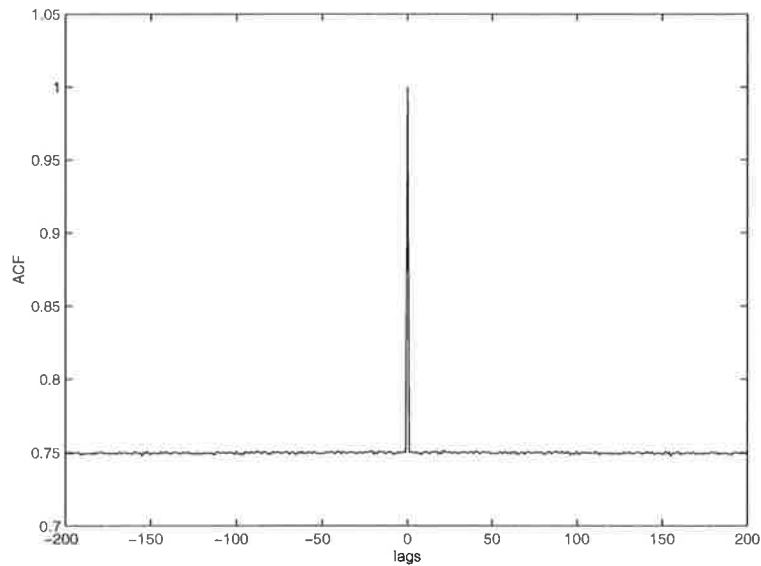


Fig. 5.9(b) ACF of uniformly distributed sequence generated by Matlab

Figure 5.10 is the pair plot of the first 20,000 normalised numbers of the sequence generated by $LCG(16807, 0, 2^{31} - 1, 1)$. The degree of uniformity that the unit square is covered by the pairs of (R_n, R_{n+1}) is another indication of the randomness of the generated pseudorandom sequence.

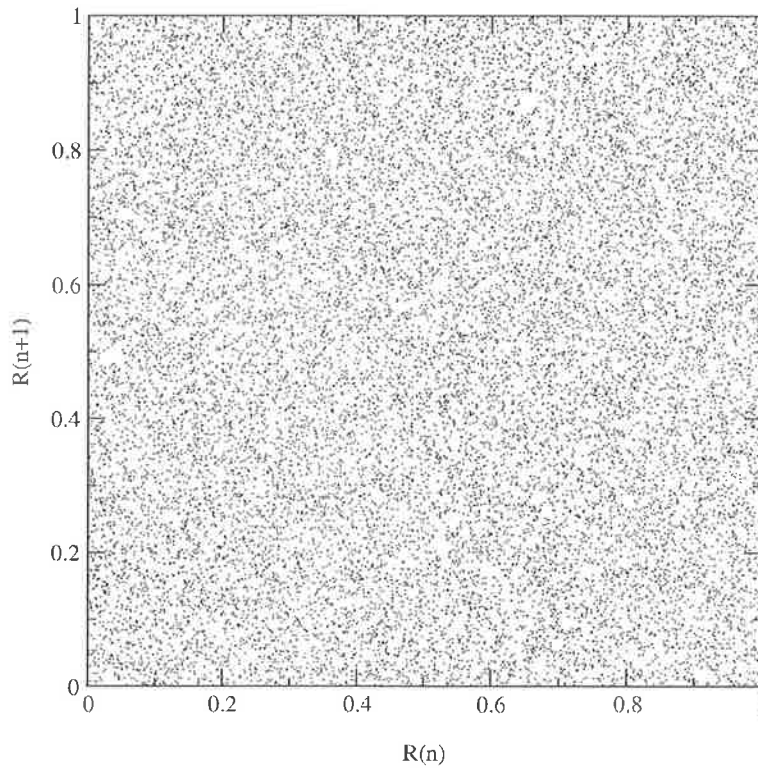


Fig. 5.10 Pair plot of the $LCG(16807, 0, 2^{31} - 1, 1)$ sequence

The very long period, the near theoretical values of the mean and variance, the autocorrelation function and pair plot show that the pseudorandom number sequence generated by $LCG(16807, 0, 2^{31} - 1, 1)$ should be adequate for the purpose of emulating uniformly distributed link transmission errors.

5.4 Experiment Results

5.4.1 Results of Experiments over Lossless Links

In the first set of experiments, a single TCP connection was created between node H_1 and node H_3 . Node H_1 used this TCP connection to transmit 100,000 bytes of data to node H_3 . After the data transfer was completed, the TCP connection was torn down. This set of experiments consisted of using a 115.2kbps 0ms-delayed, 38.4kbps 80ms-delayed and 38.4kbps 250ms-delayed serial link respectively to connect the nodes H_2 and H_3 . The objective of this set of experiments is to test how ST-PD TCP adapts to different bandwidth-delay products of TCP connections. During those TCP connections' life time, *tcpdump* was running on the node H_1 to dump all departure and arrival traffic traces. Figures 5.11- 5.19 were drawn from the data extracted from those traffic traces.

Figure 5.11(a) and (b) show the traffic traces of TCP Reno connection and ST-PD TCP connection respectively when the bottleneck link has the capacity of 115.2kbps and zero propagation delay. The solid lines are the traces of transmitted data sequence numbers departed from source node H_1 . The dashed lines are the traces of acknowledged data sequence numbers arrived at source node H_1 .

In Figure 5.11(a), there are several places where the acknowledged sequence trace (the dashed line) do not change with time, which indicate that duplicate acknowledgments arrived at source node H_1 and further imply that packet losses have occurred. These packet losses happened around the beginning, the 6.5th second and the 11th second of the TCP Reno connection. After experiencing these packet losses, the transmitted sequence trace (the solid line) moved downwards, which indicates that source node H_1 was forced to retransmit the lost packets. This feature demonstrates that even when transmitting data over lossless links, TCP Reno still generates packet losses due to its reliance on flooding the bottleneck node's buffer to probe the available bandwidth. The reason that the TCP Reno source causes buffer overflows at the beginning of the data transfer is that its congestion window grows exponentially (*slow start* phase) during that period.

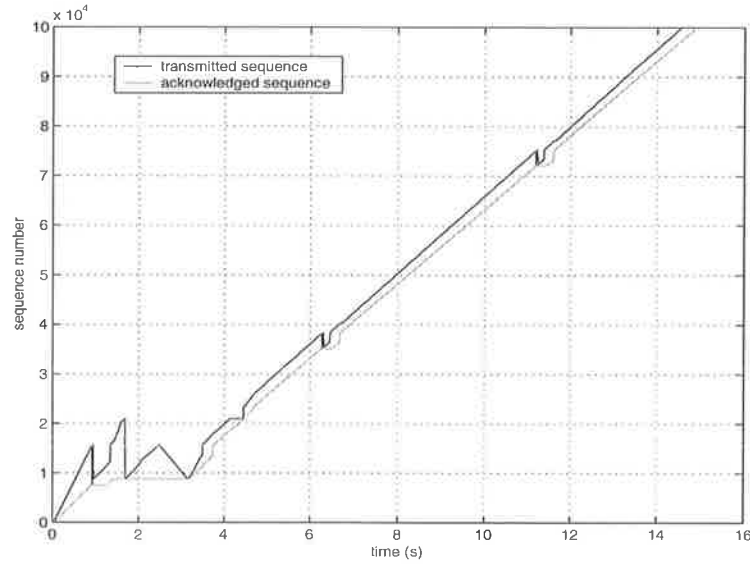


Fig. 5.11(a) Sequence traces of TCP Reno over 115.2kbps 0ms-delayed serial link

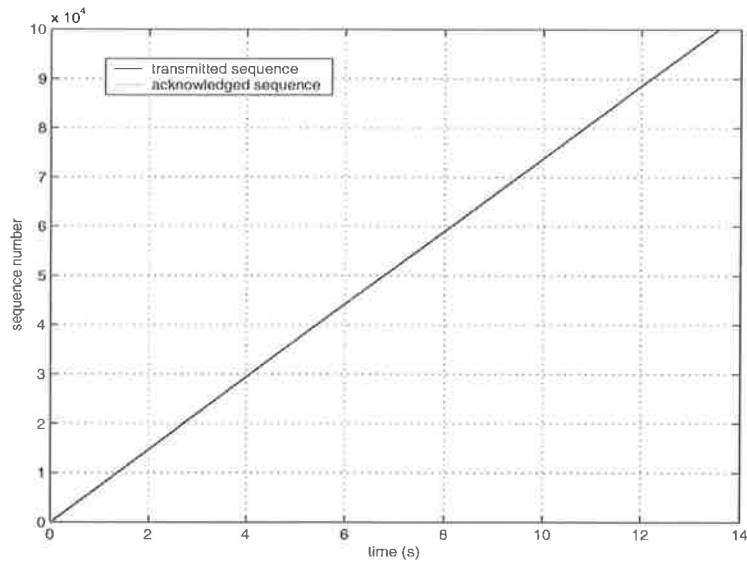


Fig. 5.11(b) Sequence traces of ST-PD TCP over 115.2kbps 0ms-delayed serial link

The horizontal distances between the transmitted and acknowledged packet sequence traces in Figure 5.11(a) and (b) reflect round-trip times of these packets. The closeness between the transmitted and acknowledged packet traces in Figure 5.11(b) shows that ST-PD TCP avoids unnecessary queuing delay by keeping the buffer occupancy at the bottleneck node around the designed level. The round-trip time of each packet transmitted by TCP Reno and ST-PD TCP is shown in Figure 5.12(a) and (b) respectively.

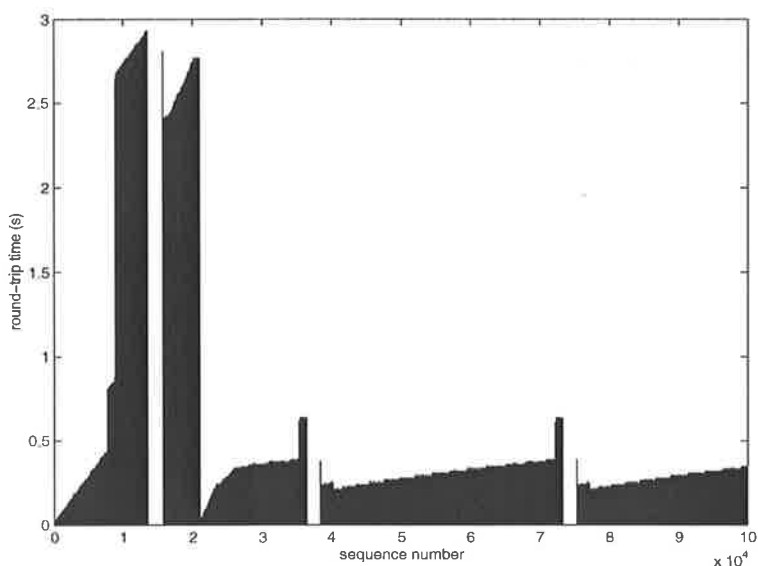


Fig. 5.12(a) RTT trace of TCP Reno over 115.2kbps 0ms-delayed serial link

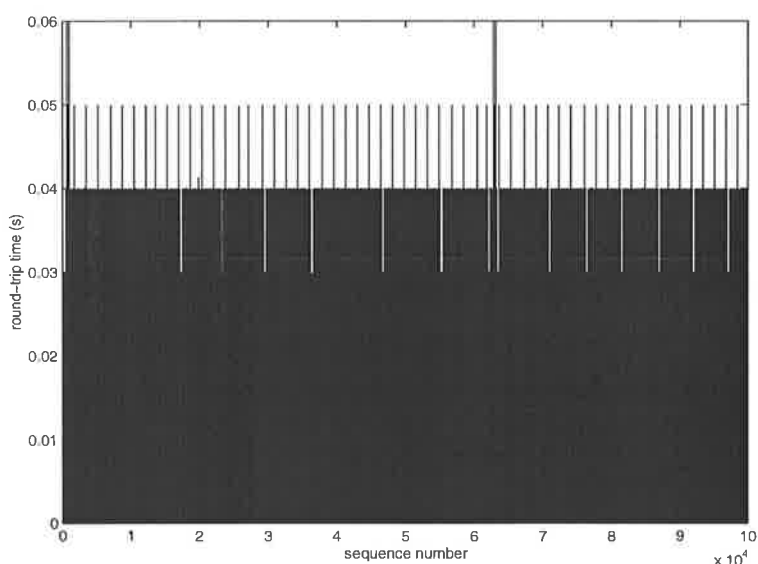


Fig. 5.12(b) RTT trace of ST-PD TCP over 115.2kbps 0ms-delayed serial link

The gaps in Figure 5.12(a) indicate that those packets arrived at the destination node as out-of-order packets. They are buffered at the destination node without being acknowledged. When the missing packet arrives, only the highest sequence number of received in-order packets is acknowledged. Figure 5.12(a) also shows the periodical buffer overflow pattern under the congestion control of TCP Reno: as TCP Reno keeps increasing its congestion window size, more packets are queued at the bottleneck node, and more queuing delay

occurs. As a result the RTT increases steadily. Once the buffer at the bottleneck node overflows, the congestion window is cut back to drain the packets in transit, and the RTT is also reduced. The cycle of congestion window growth and cut continues until the data transfer is finished.

From a given RTT value, the bottleneck node's buffer occupancy at that time can be derived based on the following equation,

$$RTT = T_{tx}^{(f)} + T_{dly}^{(f)} + T_{que}^{(f)} + T_{tx}^{(r)} + T_{dly}^{(r)} + T_{que}^{(r)} \quad (5.3)$$

where $T_{tx}^{(f)}$, $T_{dly}^{(f)}$ and $T_{que}^{(f)}$ are packet transmission time, the electrical signal propagation delay and queuing delay in the forward direction, and $T_{tx}^{(r)}$, $T_{dly}^{(r)}$ and $T_{que}^{(r)}$ are packet transmission time, the electrical signal propagation delay and queuing delay in the reverse direction respectively. T_{tx} is determined by the packet size and the link capacity, and T_{dly} is decided by the distance the electrical signal travels. In this set of experiments, the packet size is 152 bytes and the acknowledgment is 52 bytes, therefore there is no queue built up in the reverse direction, ie. $T_{que}^{(r)}$ can be removed from equation (5.3).

For example, in Figure 5.12(a), transmitting a 152-byte packet and its 52-byte acknowledgment over a 115.2kbps link takes about 13.194ms[†] and 4.512ms[†], therefore the $T_{tx}^{(f)}$ and $T_{tx}^{(r)}$ are 13.194ms and 4.512ms respectively[‡]. Given the RTT is 0.3s in Figure 5.12(a), then the queuing delay in the forward direction is 0.282s at that time, which is equivalent to 21 packets of the size of 152 bytes waiting to be transmitted by the 115.2kbps link. From the bottleneck node's buffer occupancy, the congestion window size of the TCP Reno source at that time is estimated to be 22 (21 packets are queuing plus one in transit).

ST-PD TCP attempts to keep the bottleneck node's buffer occupancy around one packet level, therefore the ideal RTT under the congestion control of ST-PD TCP should include one packet worth of queuing delay. In Figure 5.12(b), transmitting a 152-byte packet and its 52-byte acknowledgment over a 115.2kbps takes 13.194ms and 4.514ms respectively, one

[†] Sending 8-bit data over the serial link actually requires transmitting 10 bits (1 start bit, 8 data bits, 0 parity bit and 1 stop bit)

packet worth of queuing delay is 13.194ms, and the propagation delay is zero, therefore the ideal RTT in Figure 5.12(b) is 0.031s. Figure 5.12(b) confirms that ST-PD TCP effectively keeps the buffer occupancy of the bottleneck node around the designed level.

Figure 5.13(a) and (b) show the traffic traces of TCP Reno connection and ST-PD TCP connection respectively when the bottleneck link has the capacity of 38.4kbps and 80ms propagation delay.

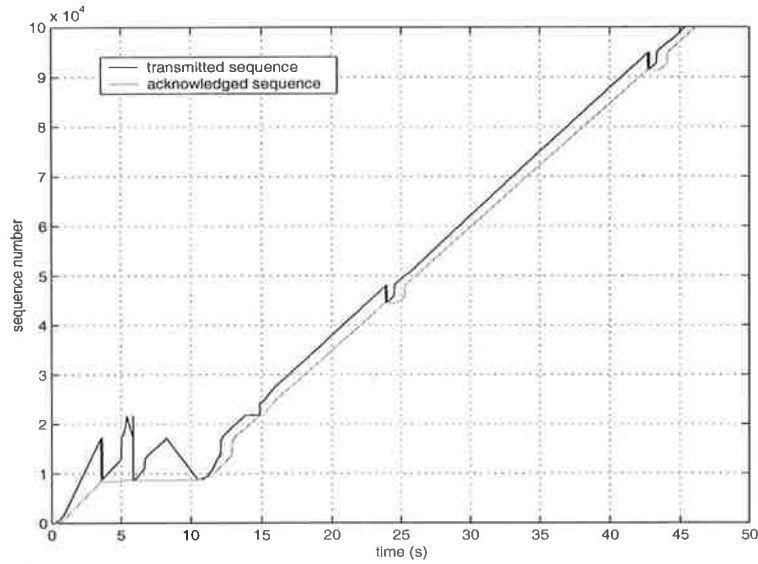


Fig. 5.13(a) Sequence trace of TCP Reno over 38.4kbps 80ms-delayed serial link

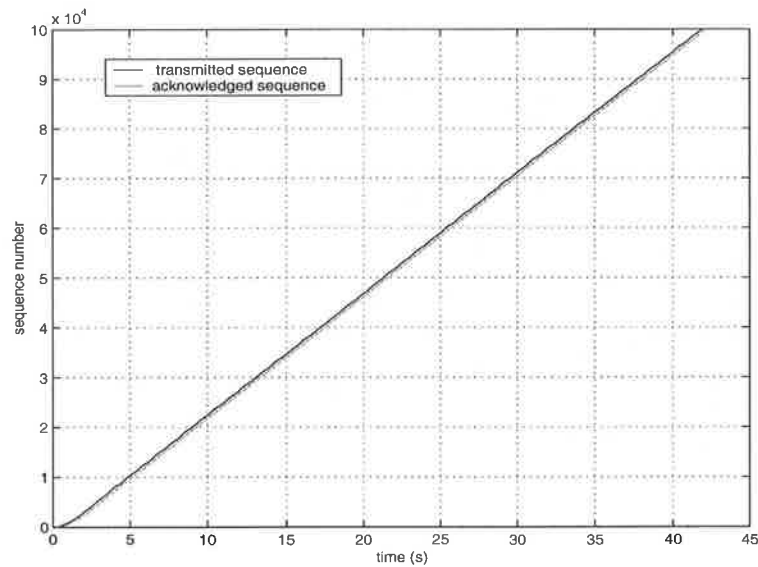


Fig. 5.13(b) Sequence trace of ST-PD TCP over 38.4kbps 80ms-delayed serial link

‡ Packet and its acknowledgment transmission times over the 100Mbps ethernet link are omitted

The horizontal distances (RTTs) between the transmitted and acknowledged packet sequence traces in Figure 5.13(a) and (b) are shown in Figure 5.14(a) and (b) respectively.

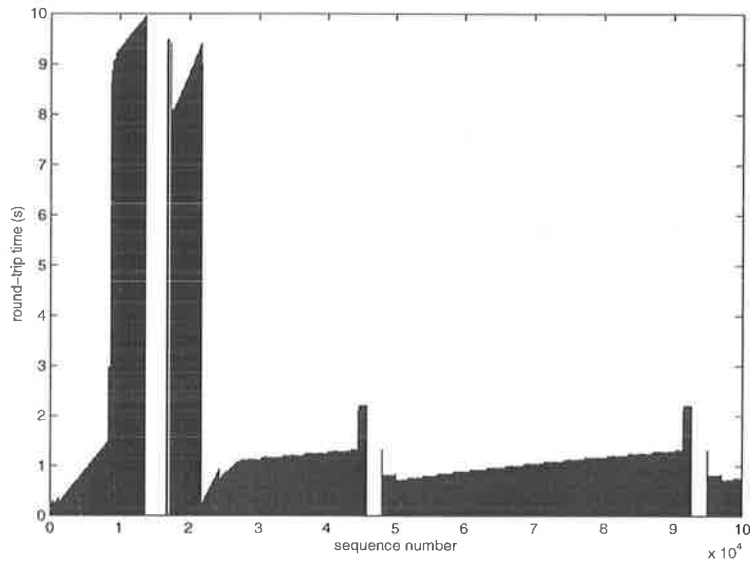


Fig. 5.14(a) RTT trace of TCP Reno over 38.4kbps 80ms-delayed serial link

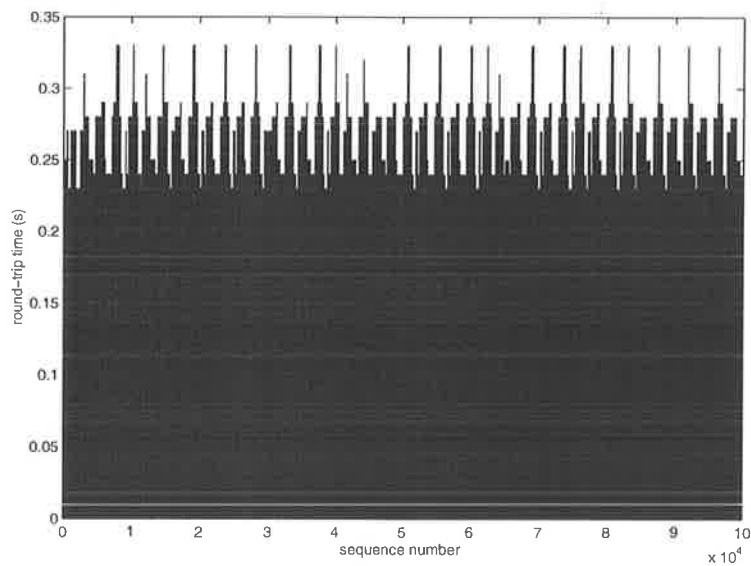


Fig. 5.14(b) RTT trace of ST-PD TCP over 38.4kbps 80ms-delayed serial link

Figure 5.15(a) and (b) show the traffic traces of TCP Reno connection and ST-PD TCP connection respectively when the bottleneck link has the capacity of 38.4kbps and 250ms propagation delay.

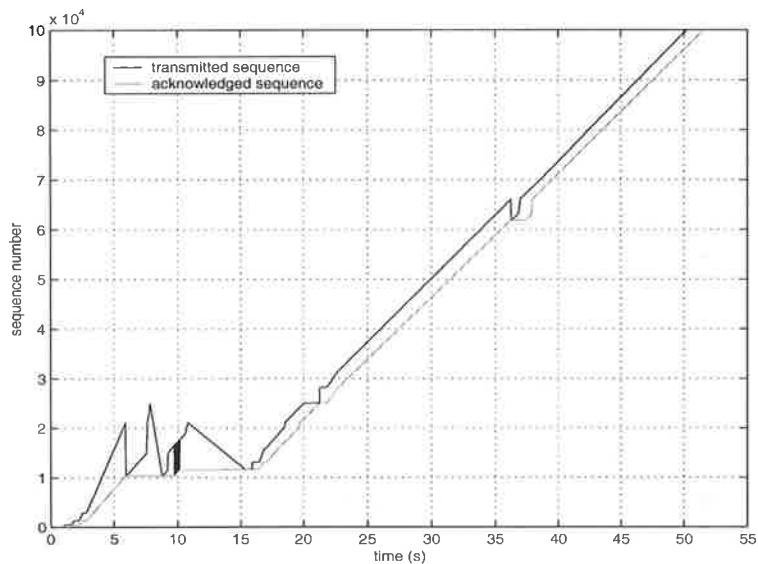


Fig. 5.15(a) Sequence trace of TCP Reno over 38.4kbps 250ms-delayed serial link

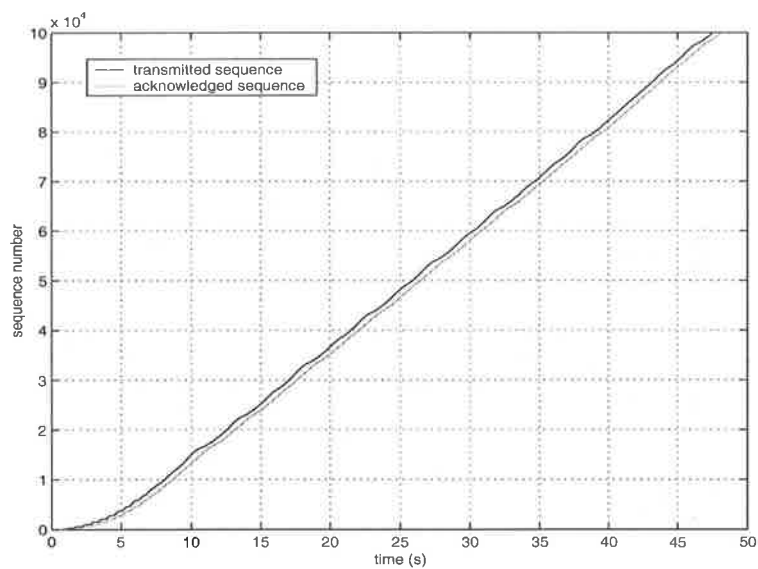


Fig. 5.15(b) Sequence trace of ST-PD TCP over 38.4kbps 250ms-delayed serial link

The horizontal distances (RTTs) between the transmitted and acknowledged packet sequence traces in Figure 5.15(a) and (b) are shown in Figure 5.16(a) and (b) respectively.

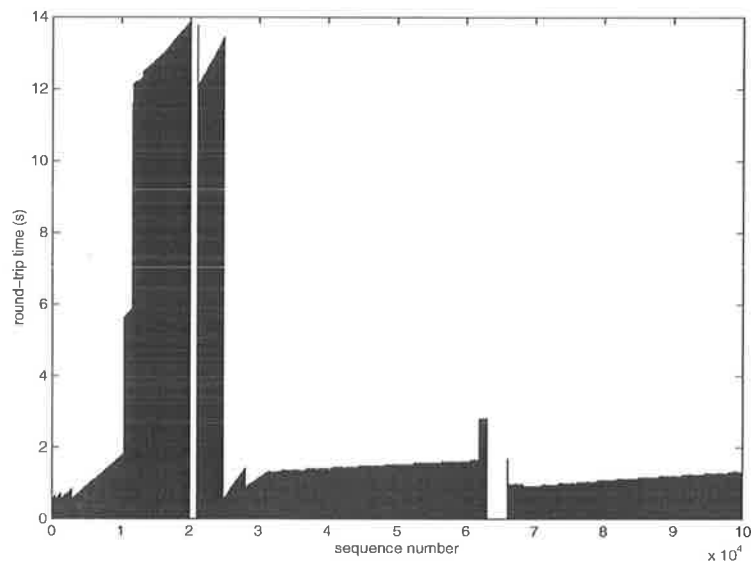


Fig. 5.16(a) RTT trace of TCP Reno over 38.4kbps 250ms-delayed serial link

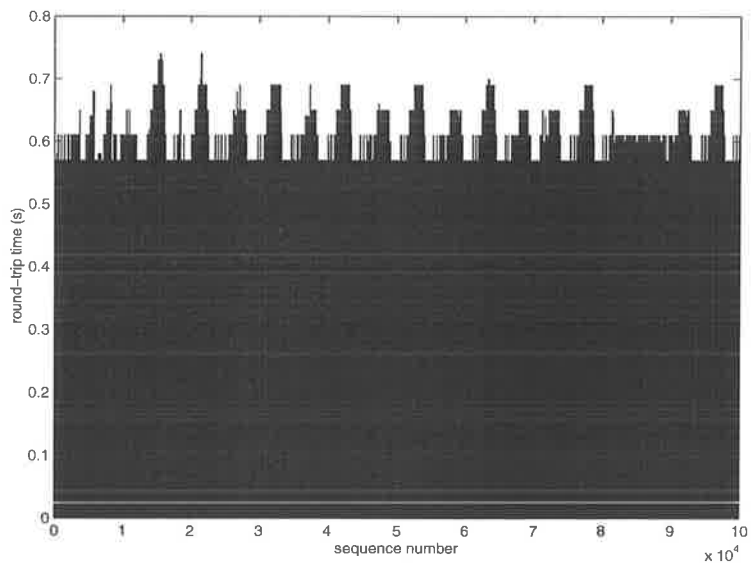


Fig. 5.16(b) RTT trace of ST-PD TCP over 38.4kbps 250ms-delayed serial link

In the same way that the ideal RTT was calculated in Figure 5.12(b), the ideal RTT in Figure 5.14(b) and Figure 5.16(b) was calculated to be 0.253s and 0.593s respectively. Figures 5.12(b), 5.14(b) and 5.16(b) confirm that ST-PD TCP effectively keeps the buffer occupancy at the bottleneck node around the designed level.

In Figures 5.11(b), 5.13(b) and 5.15(b), the increase of both the transmitted and acknowledged sequences are steady and consistent. The traces of the transmitted sequence never move downward, which indicate that packet loss never occurs. The feature in these figures demonstrates that ST-PD TCP is able to probe the available bandwidth without causing buffer overflows along its connection.

The ability of ST-PD TCP to avoid buffer overflows directly contributes to its throughput improvement. Comparing Figure 5.11(a) and (b), Figure 5.13(a) and (b), Figure 5.15(a) and (b), it can be seen that in all three circumstances, ST-PD TCP spends less time on transmitting the 100,000-byte data than TCP Reno does.

Figures 5.17 to 5.19 show the throughput of Reno and ST-PD TCP connections over the bottleneck link that had a variety of delay-bandwidth product. The throughput appears to exceed the maximum link speed during short periods because of out-of-order packets. The out-of-order packets were buffered at the receiving side without being discarded. Once the missing packet arrived, those buffered packets were retrieved immediately to become the in-sequence packets.

The throughput shown in these figures is calculated as raw throughput. It incorporates the following factors: (i) Sending 8-bit data over the serial link actually requires transmitting 10 bits (1 start bit, 8 data bits, 0 parity bit and 1 stop bit). (ii) The packet size is set to be 152 bytes in this experiment, of which the payload size is 100 bytes and TCP/IP protocol header size is 52 bytes.

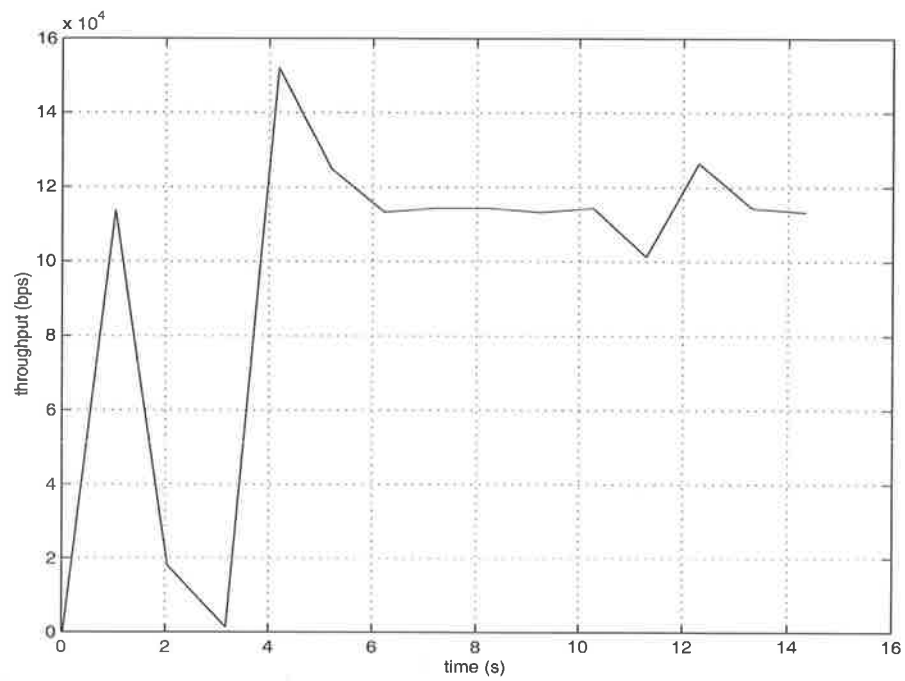


Fig. 5.17(a) Throughput of TCP Reno over 115.2kbps 0ms-delayed serial link

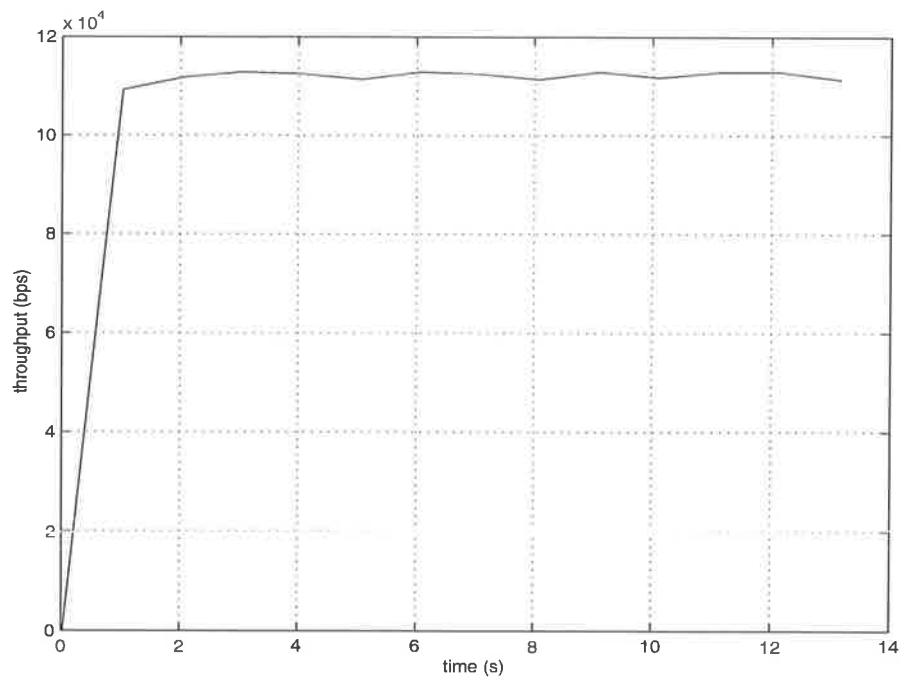


Fig. 5.17(b) Throughput of ST-PD TCP over 115.2kbps 0ms-delayed serial link

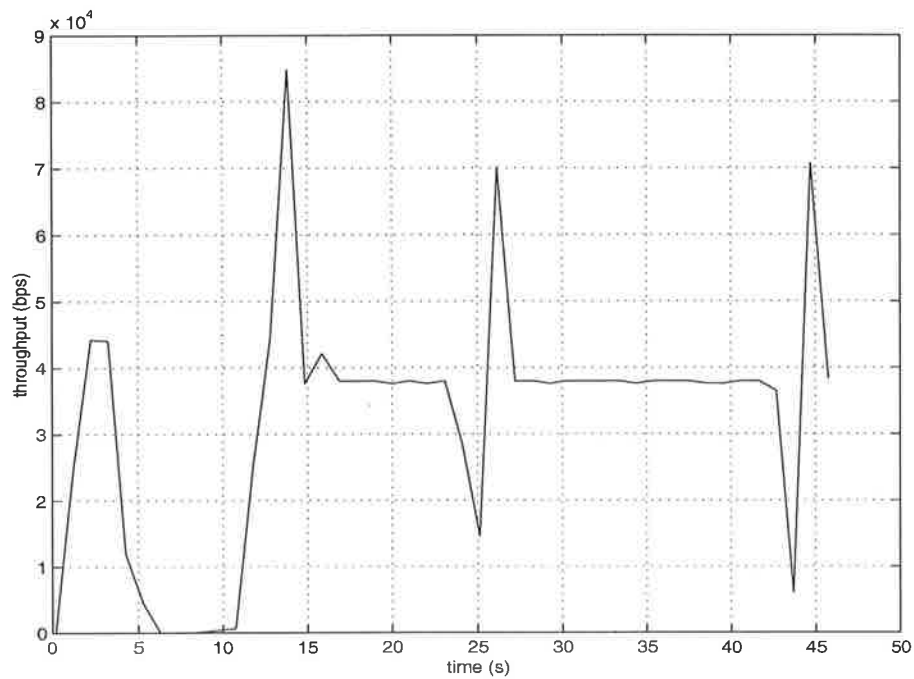


Fig. 5.18(a) Throughput of TCP Reno over 38.4kbps 80ms-delayed serial link

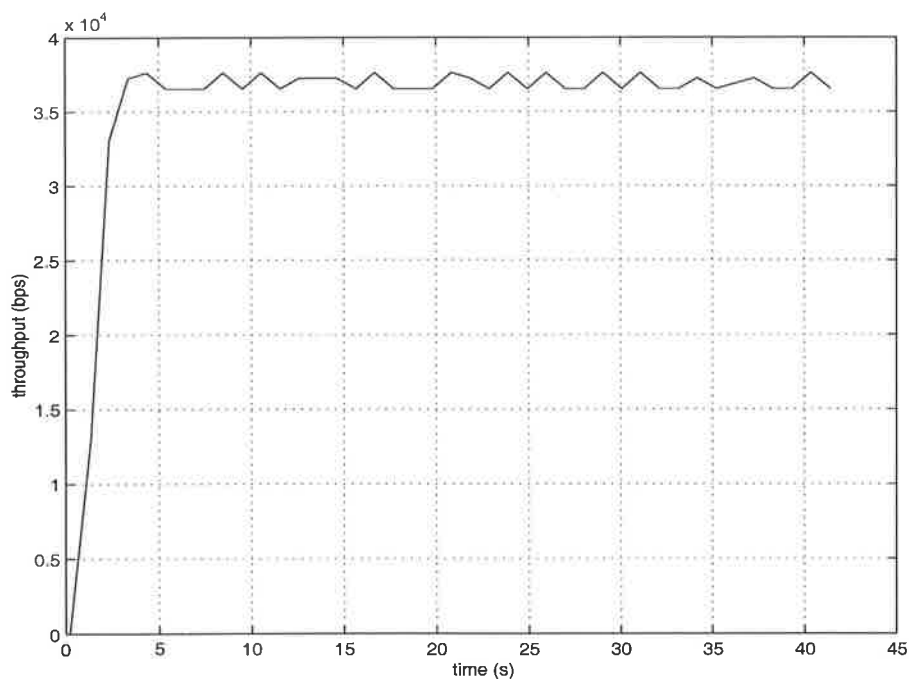


Fig. 5.18(b) Throughput of ST-PD TCP over 38.4kbps 80ms-delayed serial link

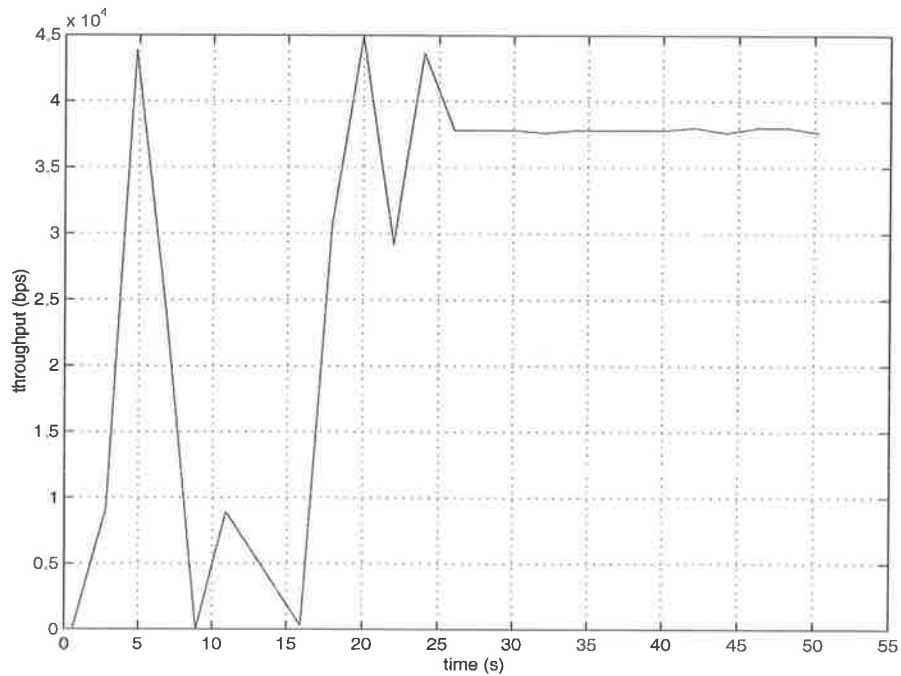


Fig. 5.19(a) Throughput of TCP Reno over 38.4kbps 250ms-delayed serial link

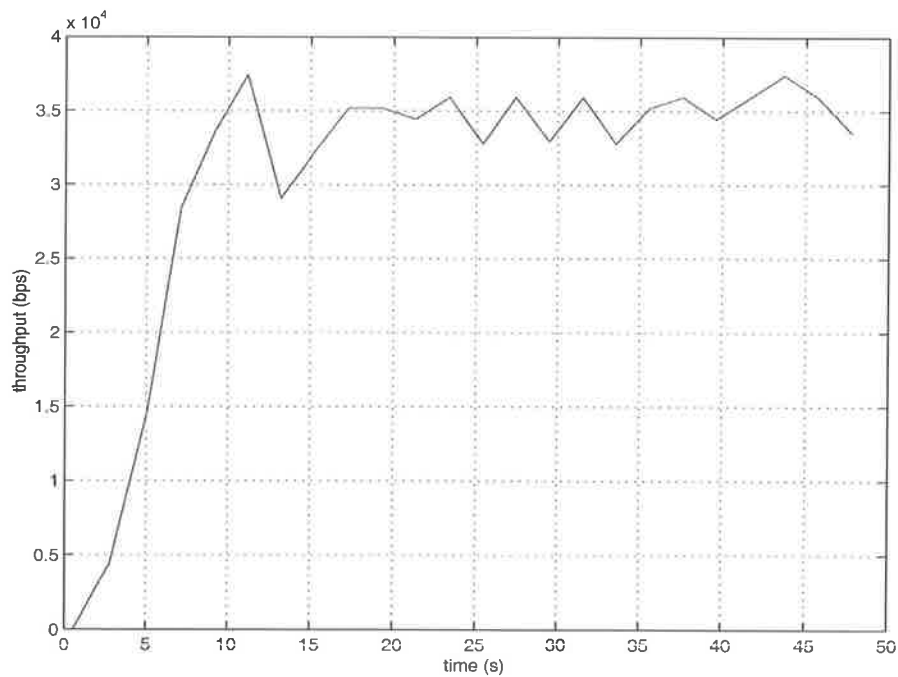


Fig. 5.19(b) Throughput of ST-PD TCP over 38.4kbps 250ms-delayed serial link

Compared with TCP Reno in the all three circumstances, ST-PD TCP is able to achieve consistent throughput during the connection life time.

In the second set of experiments, a TCP connection was created between node H_1 and node H_3 . Node H_1 used this TCP connection to transmit 100,000 bytes of data to node H_3 . A second TCP connection was created between node H_2 and node H_3 20 seconds after the start of the first TCP connection. Node H_2 used this TCP connection to transmit 100,000 bytes of data to node H_3 . A 38.4kbps 180ms-delayed serial link was used to connect the nodes H_2 and H_3 . The objective of this set of experiments is to test how ST-PD TCP reacts to available bandwidth variations. The program *tcpdump* was running on the node H_3 to dump all departure and arrival traffic traces. Figures 5.20 - 5.22 were drawn from the data extracted from those traffic traces.

Figures 5.20 and 5.21 show the arrived and acknowledged packet sequence traces of Reno and ST-PD TCP respectively. Figure 5.22 shows the throughput achieved by Reno and ST-PD TCP.

From Figures 5.20(a), 5.21(a) and 5.22(a), it can be seen that under the congestion control of TCP Reno, it took the first connection H_1 - H_3 108 seconds and second connection H_2 - H_3 80 seconds to complete the 100,000-byte data transfer. Even if the second connection H_2 - H_3 started 20 seconds later than the first connection H_1 - H_3 , it still finished the data transfer 8 seconds earlier.

Figures 5.20(b), 5.21(b) and 5.22(b) however show that under the congestion control of ST-PD TCP, it took both the first connection H_1 - H_3 and second connection H_2 - H_3 around 65 seconds to complete the 100,000-byte data transfer. It is demonstrated in Figure 5.22(b) that when a new connection joined in, the existing connection swiftly reduced its throughput to let the bottleneck bandwidth be equally shared, and when an existing connection terminated, the other connection quickly grasped the available bandwidth.

Compared with TCP Reno, ST-PD TCP not only more efficiently utilised the bottleneck link (ie., it took ST-PD TCP much less time to finish each of the 100,000-byte data transfers than TCP Reno did), but also shared the bottleneck bandwidth evenly between the two competing connections.

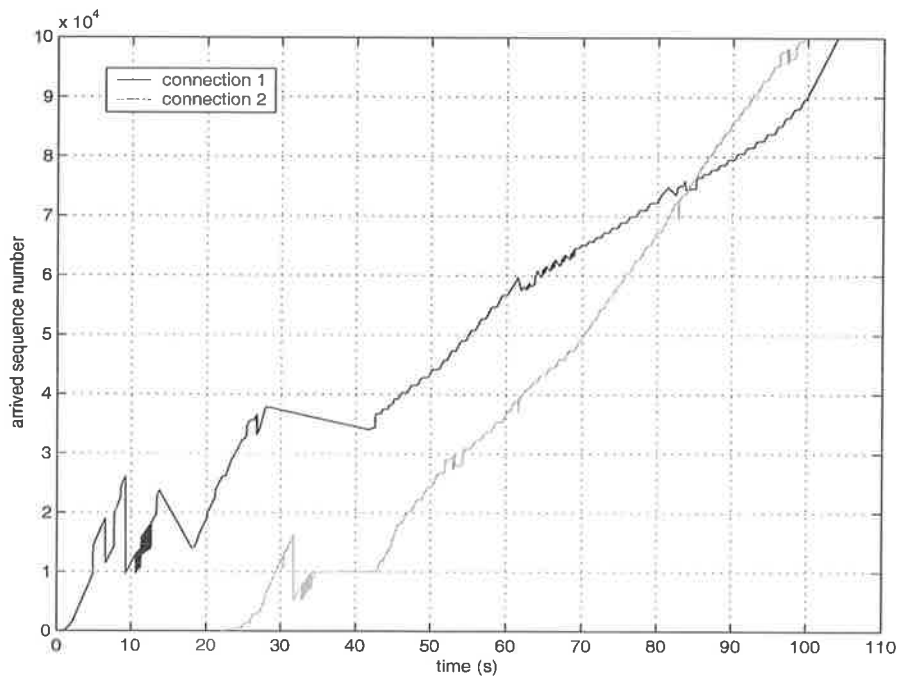


Fig. 5.20(a) Arrived packet sequence traces of two competing TCP Reno connections

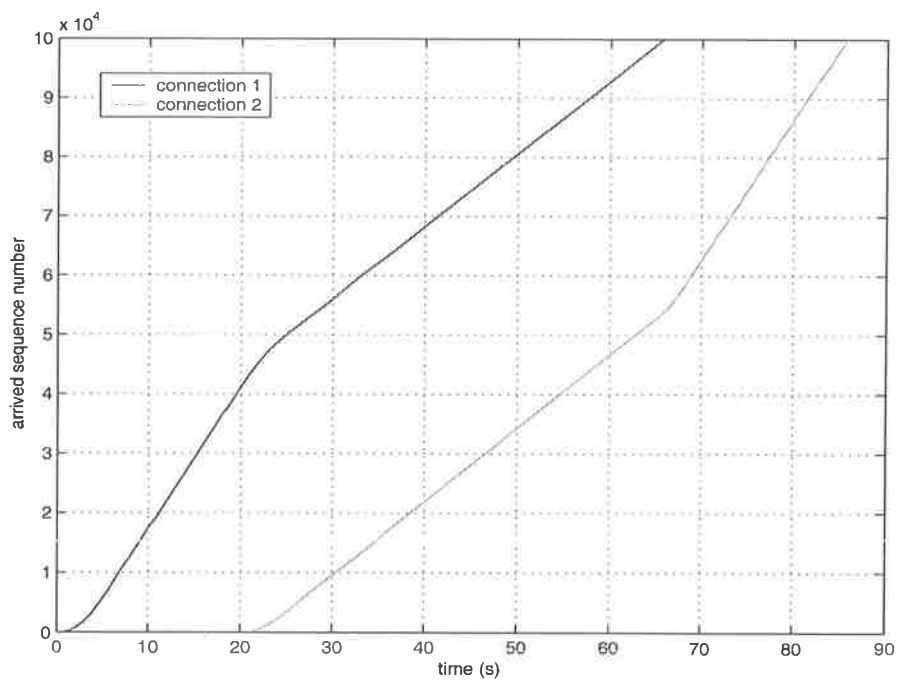


Fig. 5.20(b) Arrived packet sequence traces of two competing ST-PD TCP connections

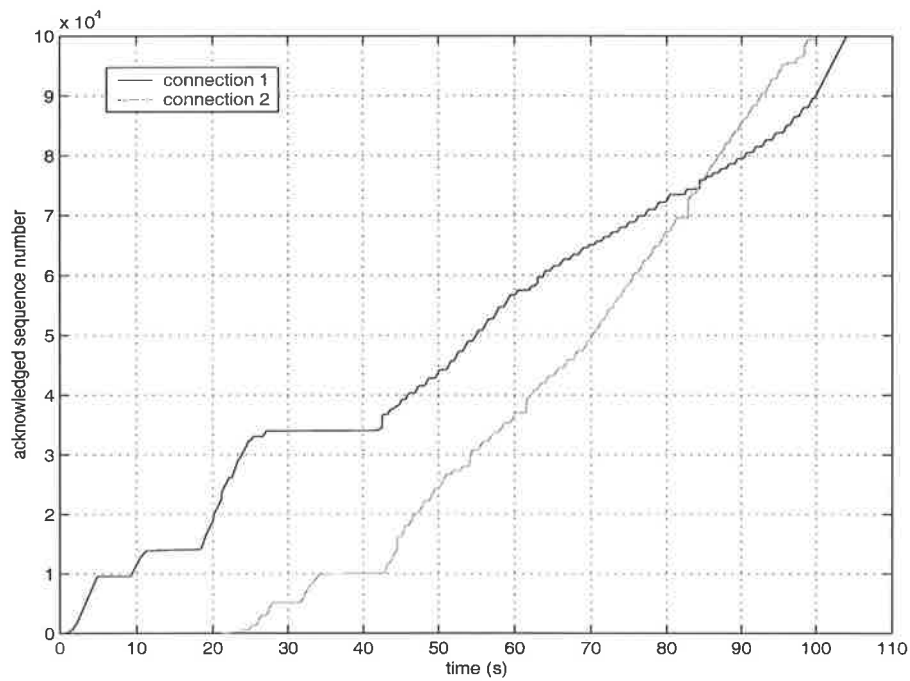


Fig. 5.21(a) Acknowledged packet sequence traces of two competing TCP Reno connections

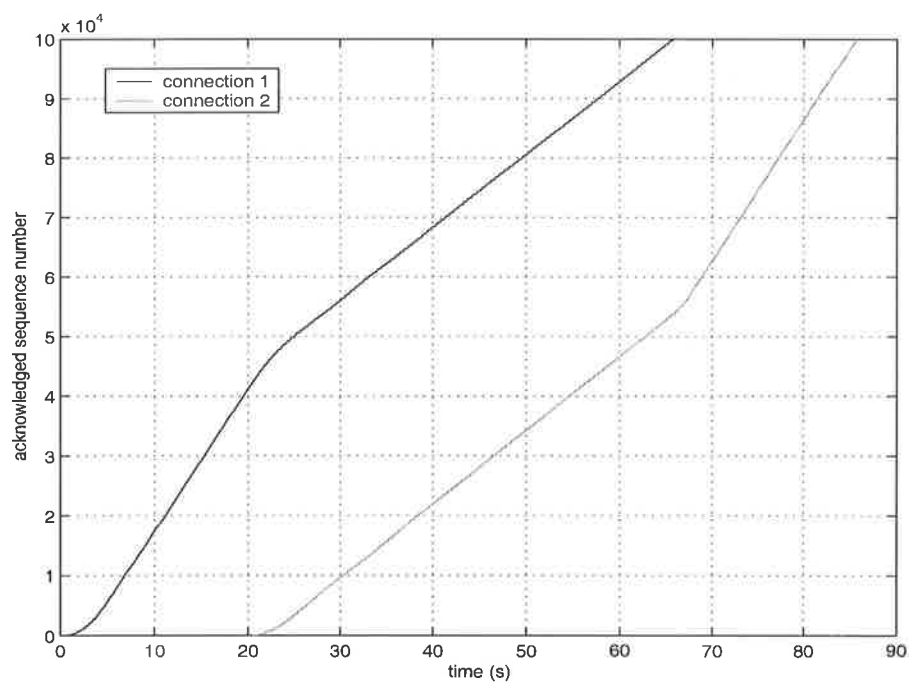


Fig. 5.21(b) Acknowledged packet sequence traces of two competing ST-PD TCP connections

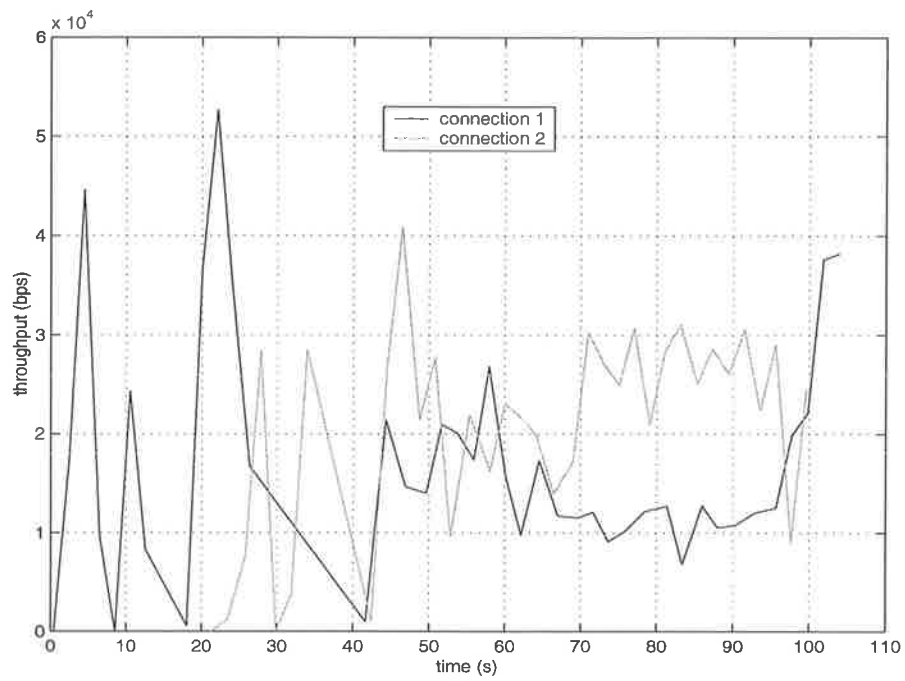


Fig. 5.22(a) Throughput of two competing TCP Reno connections

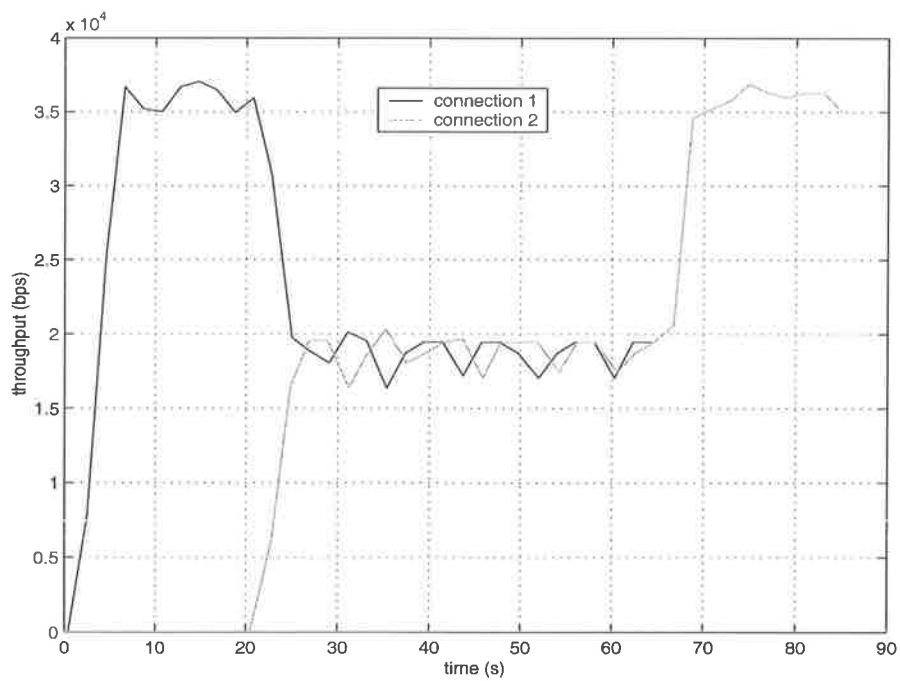


Fig. 5.22(b) Throughput of two competing ST-PD TCP connections

5.4.2 Results of Experiments over Lossy Links

In this set of experiments, a single TCP connection was created between node H_1 and node H_3 . Node H_1 used this TCP connection to transmit 100,000 bytes of data to node H_3 . The 38.4kbps 80ms-delayed and 38.4kbps 250ms-delayed serial links with packet error ratio of 10^{-1} and 10^{-2} were used to connect the nodes H_2 and H_3 . The objective of this set of experiments is to investigate how unreliable transmission media affect the throughput of different TCP implementations. Traffic traces were collected by *tcpdump* run on the node H_1 . Figures 5.23 - 5.26 were drawn from the data extracted from those traffic traces.

Figure 5.23(a) and (b) compare the acknowledged packet sequence traces and throughput between Reno and ST-PD TCP over 38.4kbps 250ms-delayed serial link with a packet error ratio of 10^{-1} . Under such a link condition, it took ST-PD TCP around 90 seconds to transfer the 100,000-byte data, while the same amount of data took TCP Reno more than 180 seconds to transfer.

Figure 5.24(a) and (b) compare the acknowledged packet sequence traces and throughput between Reno and ST-PD TCP over 38.4kbps 250ms-delayed serial link with a packet error ratio of 10^{-2} . Under such a link condition, it took ST-PD TCP around 49 seconds to transfer the 100,000-byte data, while the same amount of data took TCP Reno more than 65 seconds to transfer.

Figure 5.25(a) and (b) compare the acknowledged packet sequence traces and throughput between Reno and ST-PD TCP over 38.4kbps 80ms-delayed serial link with a packet error ratio of 10^{-1} . Under such a link condition, it took ST-PD TCP around 60 seconds to transfer the 100,000-byte data, while the same amount of data took TCP Reno more than 74 seconds to transfer.

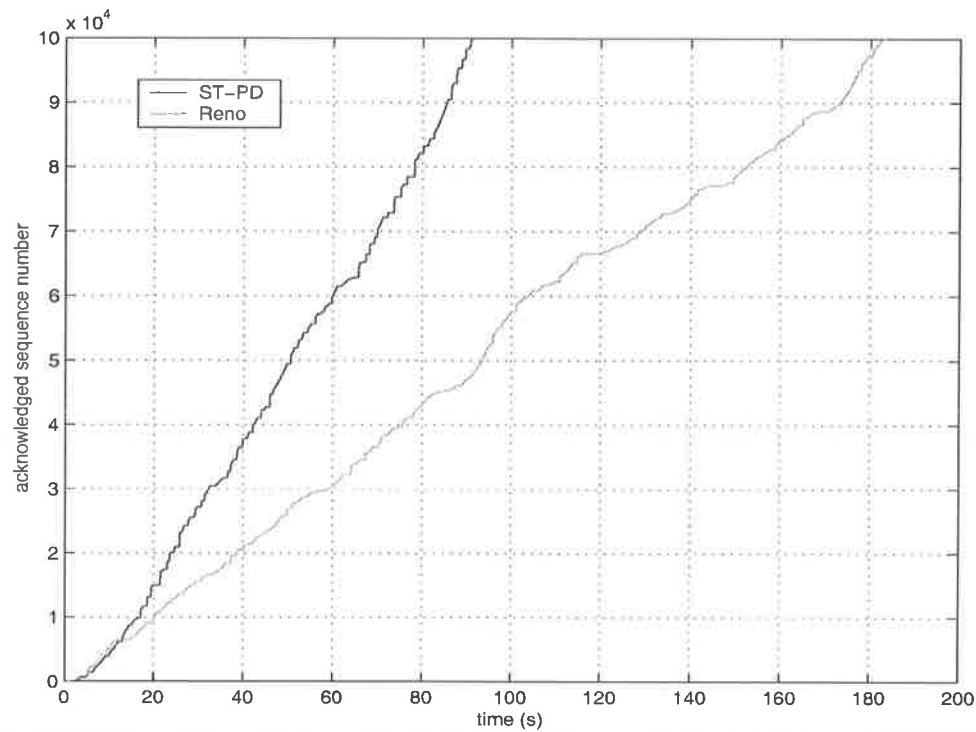


Fig. 5.23(a) Comparison of acknowledged packet sequence traces between Reno and ST-PD TCP over 38.4kbps 250ms-delayed serial link with packet error ratio of 10^{-1}

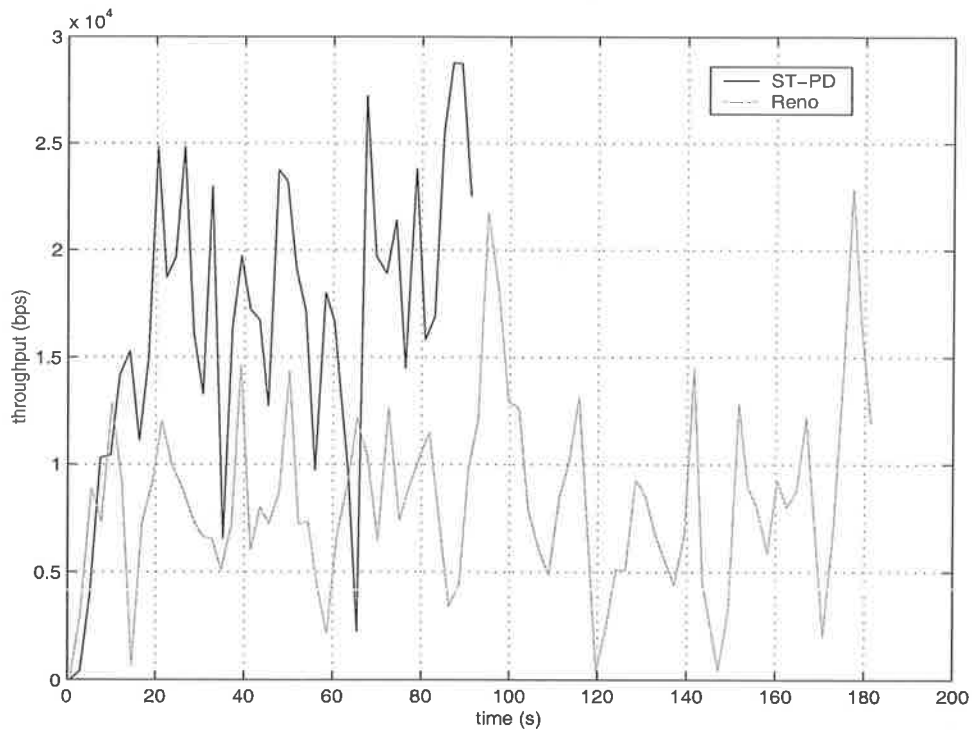


Fig. 5.23(b) Comparison of throughput between Reno and ST-PD TCP over 38.4kbps 250ms-delayed serial link with packet error ratio of 10^{-1}

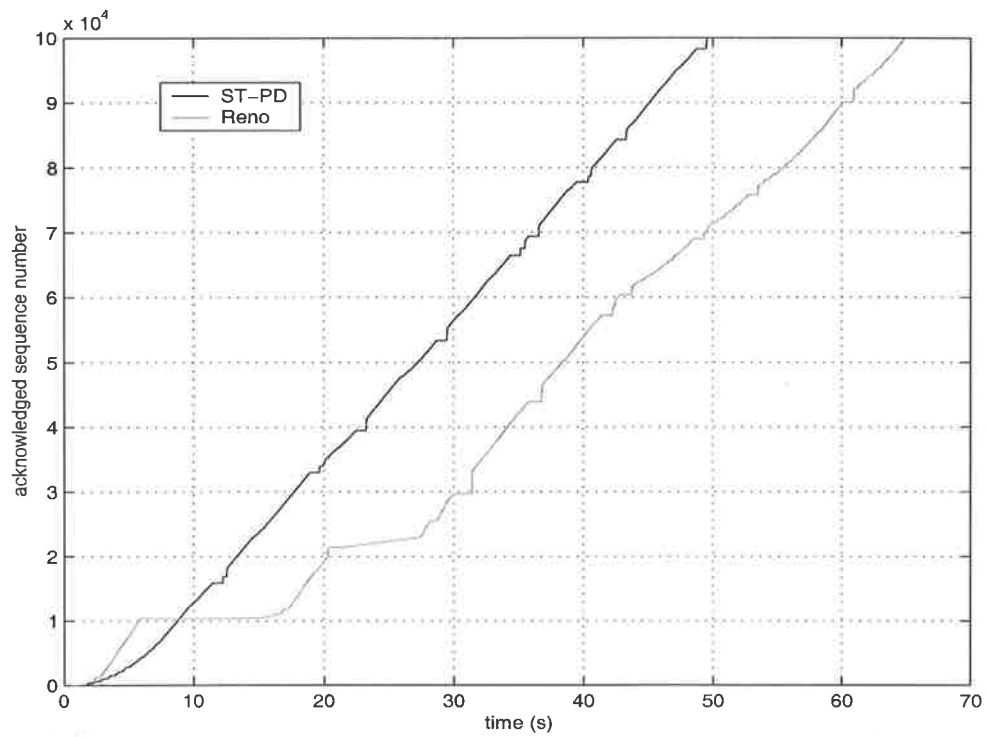


Fig. 5.24(a) Comparison of acknowledged packet sequence traces between Reno and ST-PD TCP over 38.4kbps 250ms-delayed serial link with packet error ratio of 10^{-2}

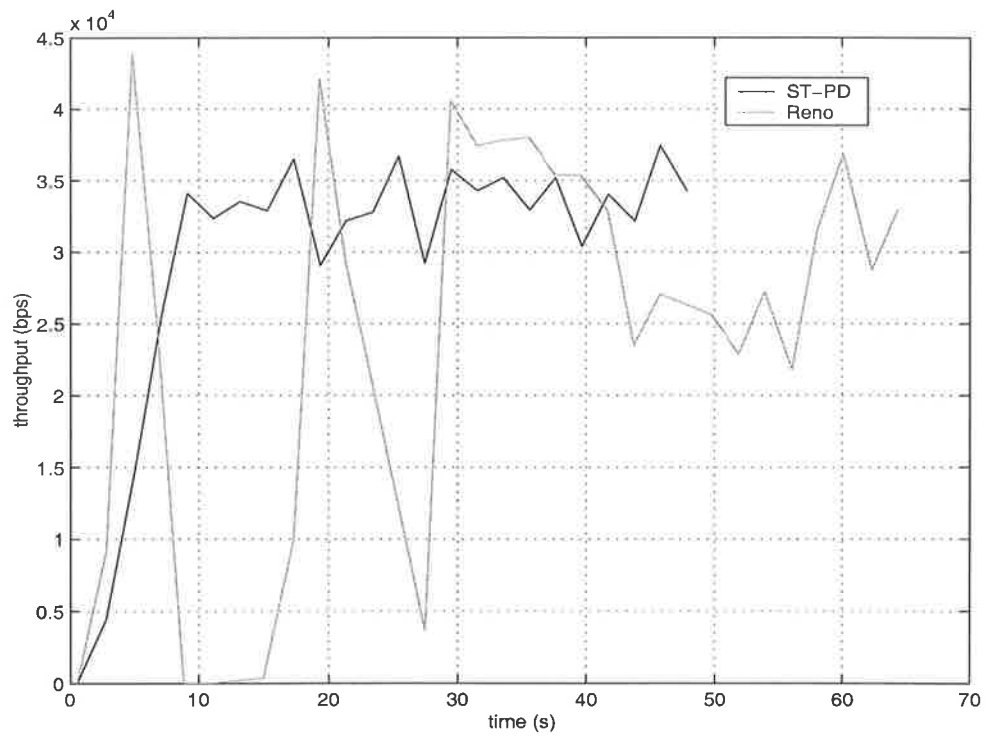


Fig. 5.24(b) Comparison of throughput between Reno and ST-PD TCP over 38.4kbps 250ms-delayed serial link with packet error ratio of 10^{-2}

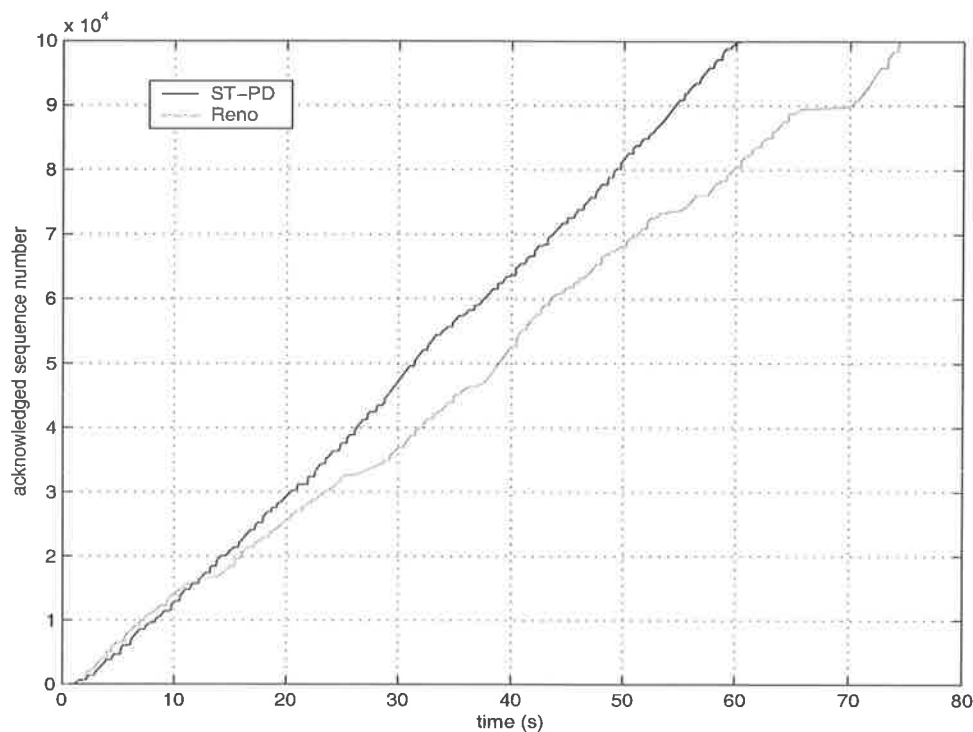


Fig. 5.25(a) Comparison of acknowledged packet sequence traces between Reno and ST-PD TCP over 38.4kbps 80ms-delayed serial link with packet error ratio of 10^{-1}

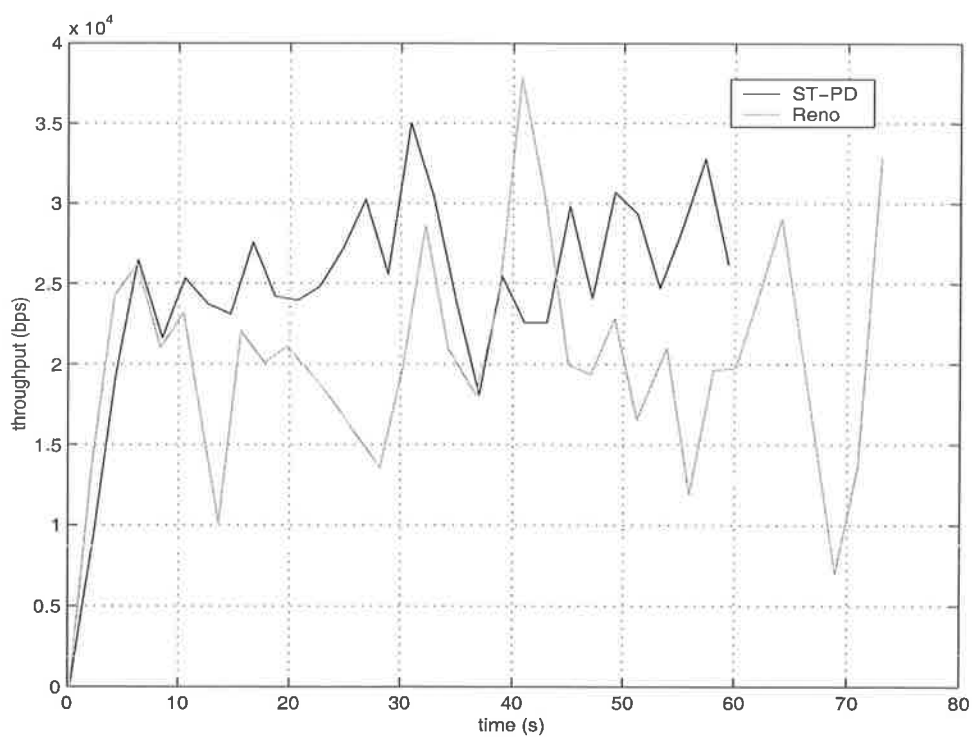


Fig. 5.25(b) Comparison of throughput between Reno and ST-PD TCP over 38.4kbps 80ms-delayed serial link with packet error ratio of 10^{-1}

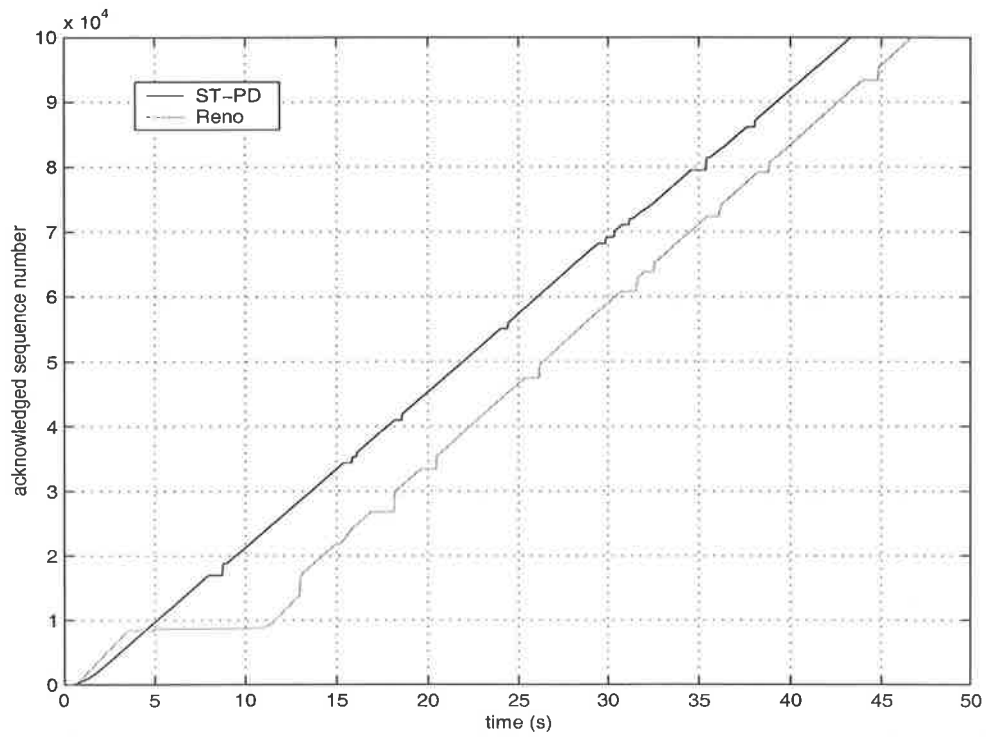


Fig. 5.26(a) Comparison of acknowledged packet sequence traces between Reno and ST-PD TCP over 38.4kbps 80ms-delayed serial link with packet error ratio of 10^{-2}

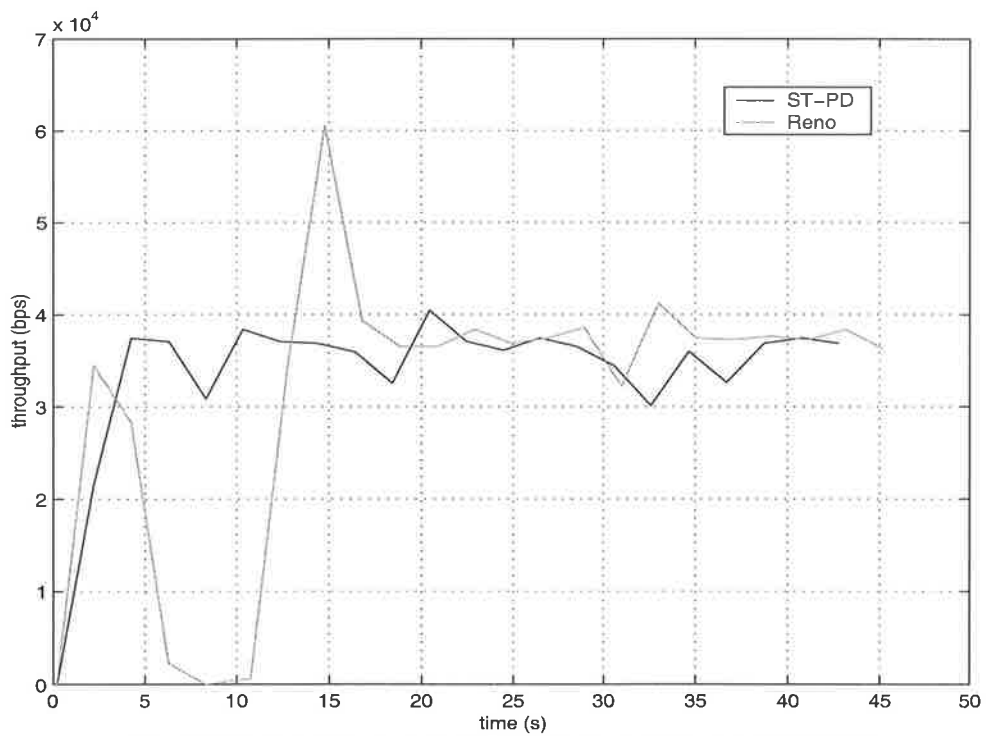


Fig. 5.26(b) Comparison of throughput between Reno and ST-PD TCP over 38.4kbps 80ms-delayed serial link with packet error ratio of 10^{-2}

Figure 5.26(a) and (b) compare the acknowledged packet sequence traces and throughput between Reno and ST-PD TCP over 38.4kbps 80ms-delayed serial link with a packet error ratio of 10^{-2} . Under such a link condition, it took ST-PD TCP around 43 seconds to transfer the 100,000-byte data, while the same amount of data took TCP Reno more than 47 seconds to transfer.

Comparing the results of the experiments conducted over the 38.4kbps 80ms-delayed and 250ms-delayed links, it can be seen that the larger the delay-bandwidth product of a TCP connection, the more the throughput deterioration is suffered by TCP Reno over lossy links. This is because once the congestion window is cut by half as a result of packet losses, it takes longer for TCP Reno to ramp up its congestion window size again over a connection with a large delay-bandwidth product compared to that with a small delay-bandwidth product.

For example, to transfer 152-byte-sized packets over the 38.4kbps 250ms-delayed link, the congestion window size must be kept above 15 packets in order to avoid starving the link. Assuming the congestion window size is cut to 7 due to a packet corruption over the lossy link, it will take TCP Reno 8 RTTs to recover the congestion window size to 15 again, provided there is no more packet transmission error during such a period. While transferring the same sized packets over the 38.4kbps 80ms-delayed link, the congestion window size only needs to stay above 6 packets to fully utilise the link. It will take TCP Reno much fewer RTTs to restore its congestion window size to 6 after a congestion window cut.

Note that the link speed of the data link emulator is limited to 38.4kbps. If the link speed reaches a few megabits per second, the performance degradation of TCP Reno caused by unreliable transmission media will deepen.

5.5 Summary

In the Linux kernel, the state variables belonging to a TCP connection are held in the structure `tcp_opt`. In order to perform the self-tuning PD congestion control formulated in chapter 3, extra members are added to the structure `tcp_opt` to represent the additional state variables required by the self-tuning PD controller. The existing TCP congestion control routines are replaced by a fuzzy tuning routine, which updates the gains of the underlying PD controller according to the perceived bandwidth-RTT product of the TCP connection based on a set of fuzzy logic control rules, and a classic PD control routine, which adjusts the congestion window size based on the proportional and derivative control principle.

To provide the floating-point arithmetic operations necessary for conducting PD control, a software realisation of a minimum set of floating-point arithmetic operations is incorporated into the Linux kernel. Further, the resolution of the timestamp in ST-PD TCP is increased 1000 times to improve the accuracy of the measured RTTs in local and/or high-speed networking environments.

A testbed network was set up to test ST-PD TCP performance. A data link emulator, which constituted a vital part of the testbed network, was constructed to emulate communication links with a wide range of propagation delay and transmission error characteristics.

The results of experiments conducted over a range of lossless links show that ST-PD TCP effectively keeps the bottleneck node's buffer occupancy around the designed level. The RTTs extracted from TCP traffic traces confirm that queuing delays experienced by packets belonging to ST-PD TCP connections hover around the designed point with a very narrow range of deviations. In contrast, the RTTs under the congestion control of TCP Reno increase steadily until no more packets can be buffered at the bottleneck node. Another set of experiments involved two TCP connections competing for the bottleneck link bandwidth show that ST-PD TCP connections react more swiftly to the variations of the available network resource than TCP Reno connections do. Due to its ability to avoid buffer overflows, ST-PD TCP is able to achieve higher throughput than TCP Reno over lossless links.

The results of experiments conducted over a range of lossy links show that compared with the heuristic based congestion control mechanism adopted in TCP Reno, adjusting congestion window based on the PD control principle makes ST-PD TCP less vulnerable to link error induced performance deterioration, and the larger the delay-bandwidth product of a TCP connection, the more advantage ST-PD TCP has over TCP Reno.

Chapter 6

The Coexistence of ST-PD TCP with Existing TCP Deployment

Given the wide deployment base of existing TCP implementations, it is unrealistic to expect that a newly devised TCP can get acceptance by all sites overnight. Any new versions of TCP can only be deployed incrementally. Therefore the ability to coexist with current TCPs becomes a primary considering factor when redesigning TCP. The design of ST-PD TCP is not an exception.

The coexistence of ST-PD TCP with existing systems can be judged according to the following two criteria:

- ❑ **functional compatibility:** existing TCP implementations should be able to communicate with ST-PD TCP without any modifications;
- ❑ **operational congeniality:** the normal operations of TCP connections that use existing TCP implementations should not be adversely affected by the presence of ST-PD TCP connections.

The functional compatibility ensures that applications of ST-PD TCP are not confined to its initial tiny deployment base. The operational congeniality excludes the possibility that the performance improvement achieved by ST-PD TCP connections is at the expense of degrading the performance of connections that use current TCP implementations.

As stated earlier in Chapter 3, functional compatibility is one of the design goals of ST-PD TCP. ST-PD TCP realises this goal in the following scenarios, which enumerate all possible ways that existing TCPs and ST-PD TCP communicate with each other:

- 1) Existing TCPs act as traffic sources, which transmit data, and ST-PD TCP acts as traffic sinks, which transmit acknowledgments;

- 2) ST-PD TCP acts as traffic sources, which transmit data, and existing TCPs act as traffic sinks, which transmit acknowledgments[†];
- 3) ST-PD TCP and existing TCPs act as both traffic sources and traffic sinks, and they transmit both data and acknowledgments.

Now consider the issue of operational congeniality. The performance of existing TCPs and ST-PD TCP is determined by the ways they control their congestion window sizes. ST-PD TCP adjusts its congestion window size aiming to keep the bottleneck node's buffer occupancy around the designed level; while existing TCP implementations keep increasing their congestion window sizes until the bottleneck node's buffer overflows. Therefore, ST-PD TCP connections and the connections that use existing TCP implementations do not affect each other if they do not share the same bottleneck queue. Furthermore, even if they do share a bottleneck queue, ST-PD TCP connections do not unfairly grasp extra network resource from the connections that use current TCP implementations, due to ST-PD TCP's less aggressive approach to occupying the bottleneck node's buffer space. The following simulation was conducted to demonstrate that the performance of a TCP Reno connection is not adversely affected by sharing a bottleneck queue with a ST-PD TCP connection.

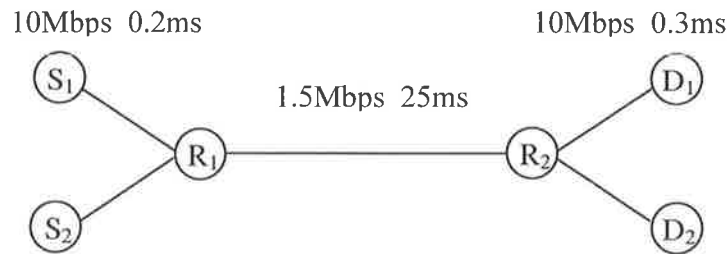


Fig. 6.1 Network configuration of two mixed-version TCP connections

The network configuration is shown in Figure 6.1, in which the link capacities and propagation delays are given. The node R_1 has a buffer space that can accommodate 5 packets of the size of 1000 bytes. The node S_1 uses TCP Reno to transmit an infinitely large file to the node D_1 , and 5 seconds later the node S_2 uses ST-PD TCP to transmit an infinitely

[†] Some TCP implementations adopt a mechanism called delayed acknowledgments, which does not affect the functional compatibility of ST-PD TCP, but for performance consideration, it must disable this feature.

large file to the node D_2 . Since the capacities of the incoming links S_1R_1 and S_2R_1 at the node R_1 exceed the capacity of the outgoing link R_1R_2 , packets belonged to connections S_1D_1 and S_2D_2 will queue at the node R_1 during their data transfers. Figure 6.2(a) compares the throughput of the two TCP connections.

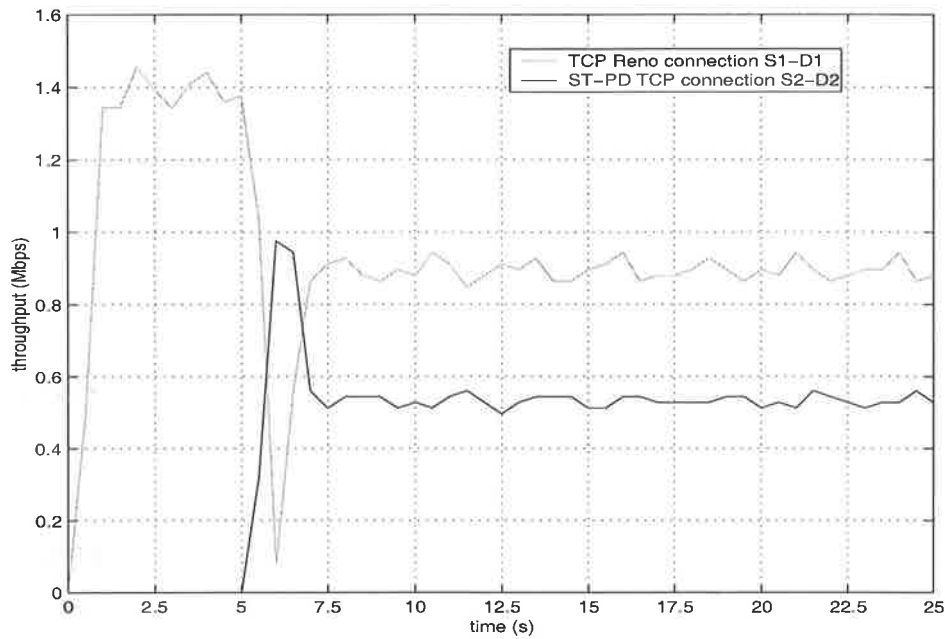


Fig. 6.2(a) Throughput of two TCP connections

Figure 6.2(a) shows that the ST-PD TCP connection gets a lower share of the bandwidth of the link R_1R_2 , when it shares the buffer space of the node R_1 with the TCP Reno connection. This is because ST-PD TCP is sensitive to the bottleneck node's buffer occupancy. If the ST-PD TCP source detects that the bottleneck node's buffer occupancy has increased, it will reduce its congestion window to draw back the bottleneck node's buffer occupancy to the designed level. However packets arriving from the TCP Reno source continue to occupy the bottleneck node's buffer space until the buffer is full. Therefore, for connections that use existing TCP implementations, such as TCP Reno, sharing a bottleneck queue with ST-PD TCP connections does not deteriorate their performance. However for ST-PD TCP connections, sharing a bottleneck queue with connections that use TCP Reno-style congestion control scheme does affect the congestion control function of ST-PD TCP.

To test the performance of ST-PD TCP connection S_2D_2 under the circumstance where more TCP Reno connections share the same buffer with the ST-PD TCP connection, the above simulation was conducted again with a second TCP Reno connection S_3D_3 joining in at the 10th second. The throughput of the three TCP connections, shown in Figure 6.2(b), demonstrates that the throughput of the ST-PD TCP connection S_2D_2 does not deteriorate on average after the second TCP Reno connection S_3D_3 joins in.

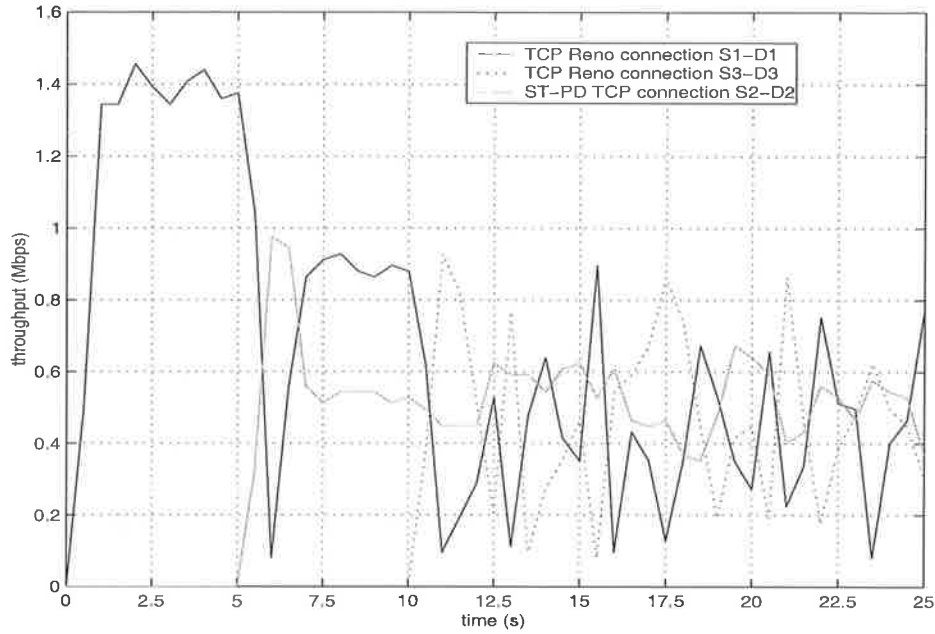


Fig. 6.2(b) Throughput of three TCP connections

Even though the normal operation of ST-PD TCP is affected by the presence of TCP Reno connections, ST-PD TCP connections still outperform TCP Reno connections if the connections consist of lossy links. To demonstrate this, the previous simulation was conducted again under the condition that the bottleneck link R_1R_2 has a packet error probability of 10^{-2} . Given the packet size of 1000 bytes in this simulation, the packet error probability is equivalent to a raw channel BER of 10^{-6} , which is not an uncommon wireless channel error rate.

Figure 6.3(a) shows the throughput of connections S_1D_1 and S_2D_2 , where S_1D_1 is a TCP Reno connection and S_2D_2 is a ST-PD TCP connection. For comparison purposes, Figure 6.3(b)

shows the throughput of connections S_1D_1 and S_2D_2 , where S_1D_1 and S_2D_2 are both TCP Reno connections.

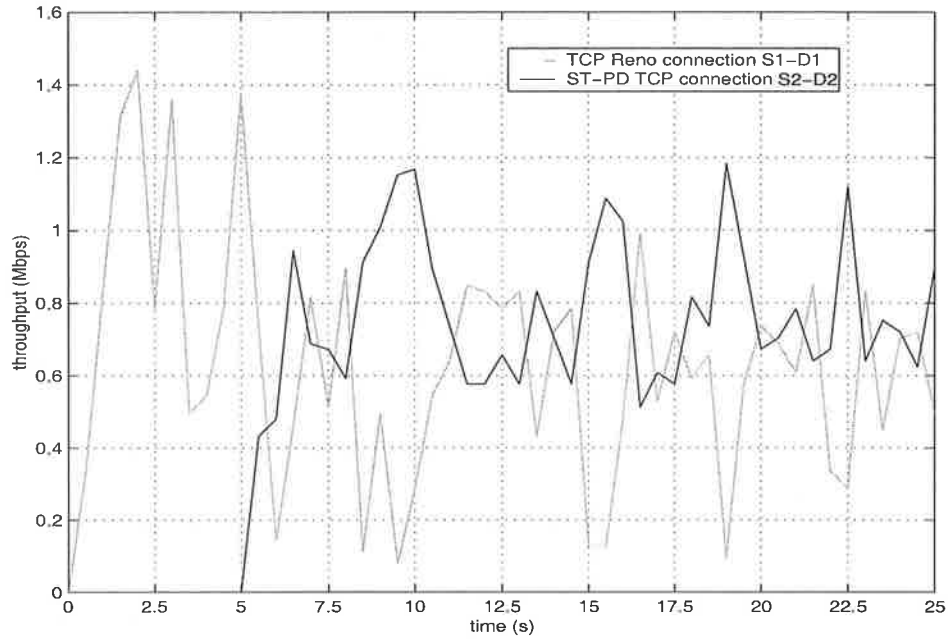


Fig. 6.3(a) Throughput of TCP Reno connection S_1D_1 and ST-PD TCP connection S_2D_2 (link R_1R_2 with packet error ratio of 10^{-2})

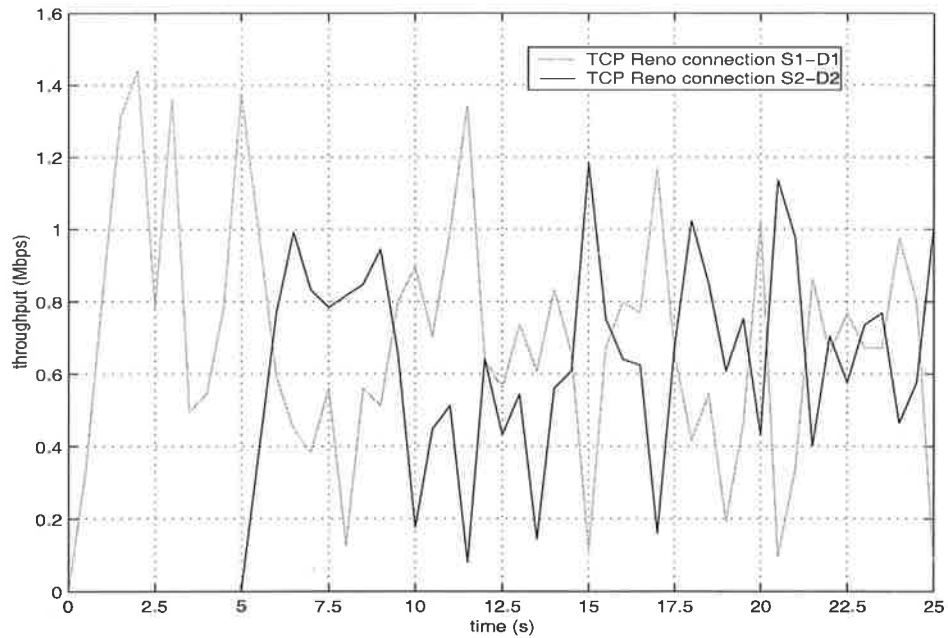


Fig. 6.3(b) Throughput of TCP Reno connection S_1D_1 and TCP Reno connection S_2D_2 (link R_1R_2 with packet error ratio of 10^{-2})

Figure 6.4 compares the in-order delivered packet sequence traces of the connection S_2D_2 , when the source node S_2 uses TCP Reno and ST-PD TCP respectively.

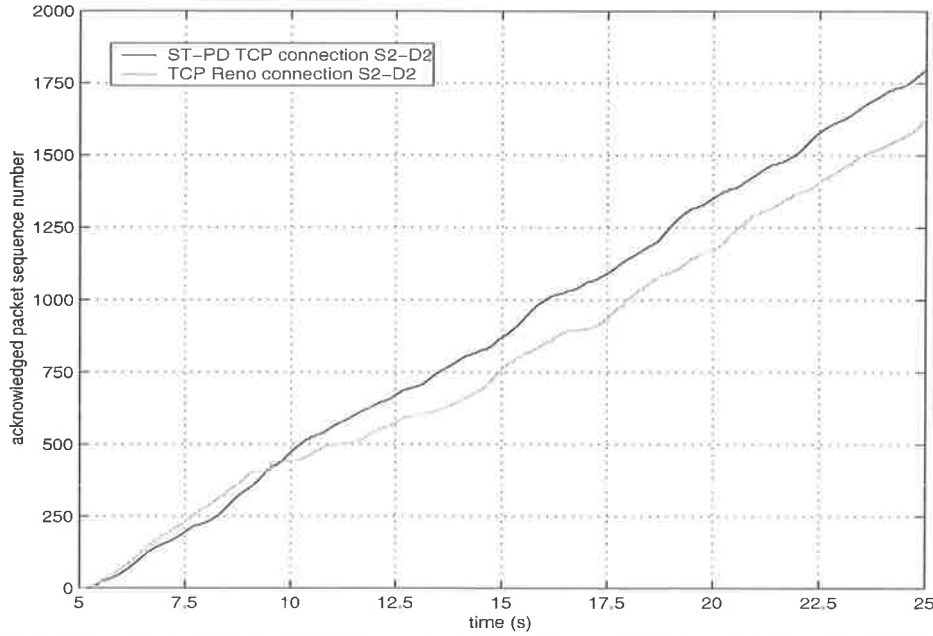


Fig 6.4 A comparison of acknowledged sequence traces of Reno and ST-PD TCP connection S_2D_2 (link R_1R_2 with packet error ratio of 10^{-2})

The acknowledged packet sequence traces in Figure 6.4 indicate that the amount of in-order delivered packets from the node S_2 to the node D_2 during the 20 seconds increases by 10 percent if TCP Reno is replaced by ST-PD TCP at the node S_2 , even though the TCP Reno connection S_1D_1 appears to have a negative effect on the normal operation of ST-PD TCP.

Two factors contribute to the improvement of ST-PD TCP throughput in the above simulation. The fundamental one is that ST-PD TCP is able to conduct congestion control and error control independently. The other factor is only relevant to the specific networking environment, where the lossy link R_1R_2 is shared by both the TCP Reno connection S_1D_1 and the ST-PD TCP connection S_2D_2 . Wireless transmission errors incurred on packets belonging to the TCP Reno connection S_1D_1 force the congestion window growth at the source node S_1 to end prematurely. Thus the bottleneck node's buffer space occupied by packets arriving from the source node S_1 is substantially reduced.

To test the performance of ST-PD TCP connection S_2D_2 under the circumstance where the TCP Reno connection S_1D_1 does not suffer from wireless losses, the above simulation was conducted again with the lossy link moved from R_1R_2 to S_2R_1 . The packet error probability remains at 10^{-2} . Figure 6.5(a) shows the throughput of connections S_1D_1 and S_2D_2 , where S_1D_1 is a TCP Reno connection and S_2D_2 is a ST-PD TCP connection. For comparison

purposes, Figure 6.5(b) shows the throughput of connections S_1D_1 and S_2D_2 , where S_1D_1 and S_2D_2 are both TCP Reno connections.

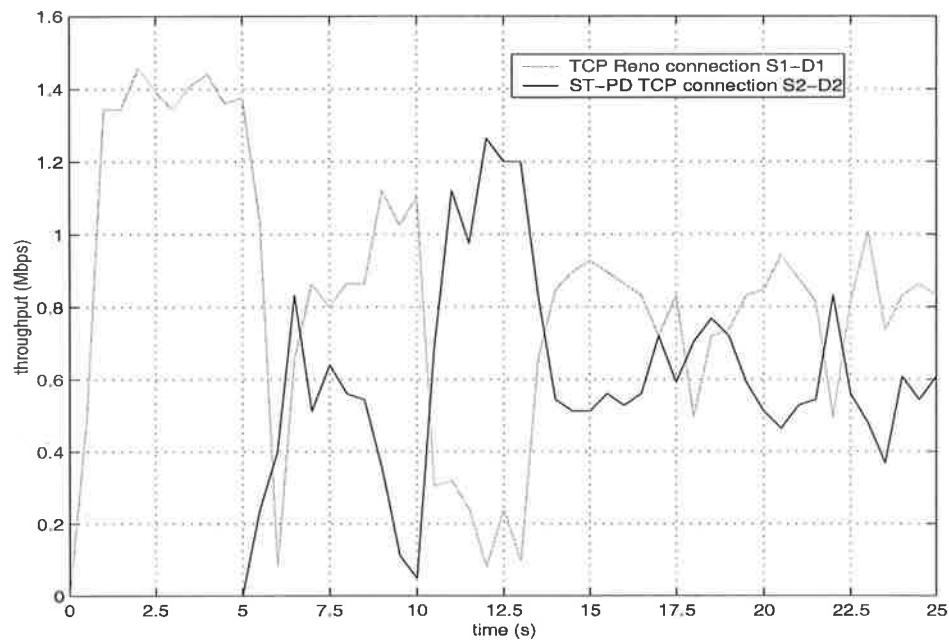


Fig. 6.5(a) Throughput of TCP Reno connection S_1D_1 and ST-PD TCP connection S_2D_2 (link S_2R_1 with packet error ratio of 10^{-2})

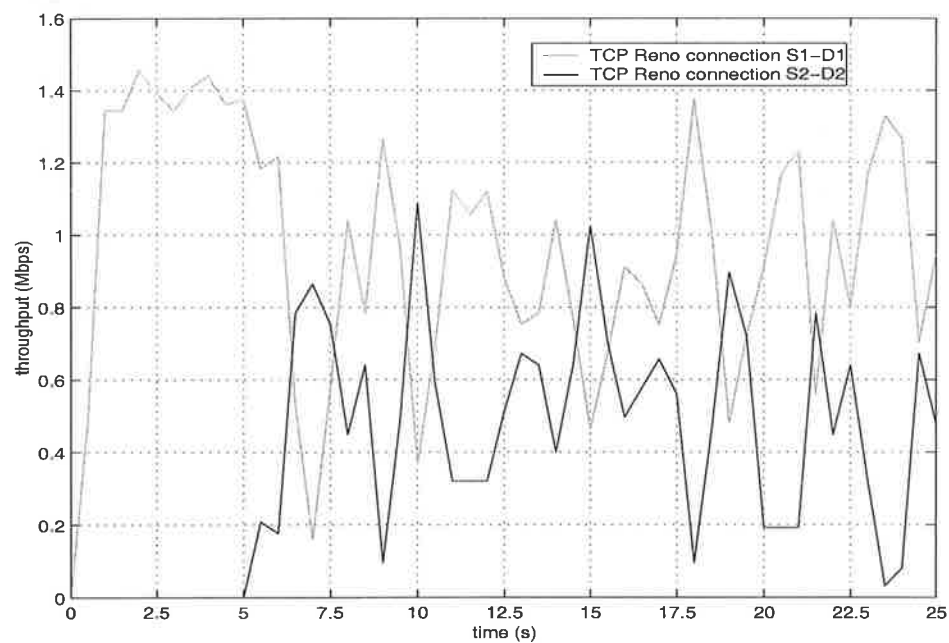


Fig. 6.5(b) Throughput of TCP Reno connection S_1D_1 and TCP Reno connection S_2D_2 (link S_2R_1 with packet error ratio of 10^{-2})

Figure 6.6 compares the in-order delivered packet sequence traces of the connection S_2D_2 , when the source node S_2 uses TCP Reno and ST-PD TCP respectively.

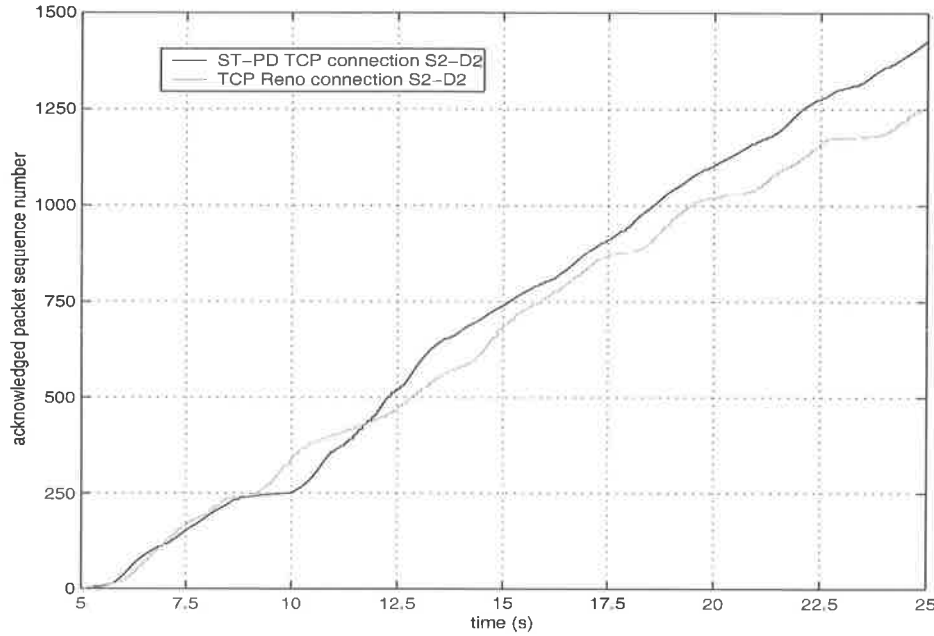


Fig. 6.6 A comparison of acknowledged packet sequence traces of Reno and ST-PD TCP connection S_2D_2 (link S_2R_1 with packet error ratio of 10^{-2})

The acknowledged packet sequence traces in Figure 6.6 show that the amount of in-order delivered packets from the node S_2 to the node D_2 during the 20 seconds increases by 14 percent if TCP Reno is replaced by ST-PD TCP at the node S_2 , in the presence of another TCP Reno connection S_1D_1 which does not suffer from wireless losses but which has an adverse impact on the congestion control operation of ST-PD TCP.

Compared with the results in the previous simulation, the throughput improvement of ST-PD TCP over TCP Reno in this simulation jumps from 10 percent to 14 percent. This is because in the previous simulation, the actual packet error rate incurred over the connection S_2D_2 is less than 10^{-2} since two connections share the lossy link, while in this simulation, the connection S_2D_2 fully experiences a packet error rate of 10^{-2} . As the throughput of ST-PD TCP is less susceptible to wireless losses than that of TCP Reno, the higher loss rate of a wireless link, the more advantage ST-PD TCP has over TCP Reno.

In summary, ST-PD TCP achieves full functional compatibility with existing TCP implementations, and the presence of ST-PD TCP connections do not degrade the performance of other connections that use current TCP implementations.

The performance of ST-PD TCP may or may not be affected by other TCP connections at the same network that adopt Reno-style congestion control mechanism, depending on whether they share a bottleneck queue. If they do not share a bottleneck queue, ST-PD TCP connections can retain all the benefits of performance improvement as described in chapter 3 and 4. This will justify the adoption of ST-PD TCP among millions ordinary Internet users who use dial-up circuits to connect to their Internet Service Providers (ISPs), as the so called “last mile” constitutes, in most occasions, the bottleneck link for these connections.

TCP connections that use a Reno-style congestion control mechanism do have an adverse impact on the normal operation of ST-PD TCP if they share a bottleneck queue. However in situations where there are unreliable transmission links in a network, the performance improvement achieved by ST-PD TCP can outweigh the negative effect on ST-PD TCP performance caused by connections that use existing TCP implementations. Therefore it is still justifiable to replace existing TCP implementations with ST-PD TCP even on a partial deployment basis in any networks that consist of terrestrial wireless or satellite components.

Chapter 7

Conclusions

TCP congestion control enables TCP-based Internet applications to adapt very well to fundamental changes in Internet infrastructure and explosive growth of the size of the Internet. Because of TCP congestion control, applications built on top of TCP can run on backbone infrastructures ranging from 50kbps leased circuits thirty years ago to today's optical fibres with a capacity of gigabits per second, and they can operate in ARPANET comprising a few hundred nodes as well as the global Internet consisting of millions of hosts.

However the network resource utilisation under the congestion control of current TCP implementations is less satisfactory, due to the heuristic approach to the design of congestion control schemes in current TCPs. The majority of TCP implementations adopt the congestion control scheme from TCP Reno, in which the congestion control process consists of two different phases — *slow start* and *congestion avoidance*, and most time the congestion control process stays in *congestion avoidance* phases, except at the beginning of the data transfer and recovery from retransmission timeout induced congestion window cut. During a *congestion avoidance* phase, current TCP implementations increase their congestion window sizes by one segment every round-trip time. The performance of current TCPs is directly affected by the heuristic-based congestion window updating mechanism in the following ways:

- There is no theoretical or practical basis for increasing a congestion window size by one segment every RTT. If a TCP connection has a large bandwidth-RTT product and the bandwidth is severely under-utilised, it is preferable to increase the congestion window size by more than one segment every RTT. On the other hand, if a TCP connection has a small bandwidth-RTT product or a TCP source's transmission rate has approached the available bandwidth, it is desirable to increase the congestion

window size by less than one segment every RTT. A direct consequence of the congestion window linear growth policy is that existing TCP implementations are very sluggish in responding to available bandwidth variations

- In each of the congestion window growing processes, current TCPs do not know whether their congestion window sizes have reached the ideal level and stop the congestion window growing process subsequently. In current TCP implementations, a congestion window growing process only stops when the bottleneck node's buffer has overflowed. At that stage, the network has been overloaded and one or more packets have been lost. As a consequence, existing TCPs are forced to shrink their congestion window sizes substantially to relieve the temporary congestion after experiencing packet losses, followed by another congestion window growing process. Clearly an assumption has been made behind such a congestion control: packet losses can only be caused by buffer overflows. When the assumption no longer holds in a network comprised of unreliable transmission links, the performance of current TCPs deteriorates drastically.
- Current TCPs cannot restrain the bottleneck queue length, therefore the packets transmitted by current TCPs can experience excessive queuing delays. Although most applications that use TCP to transport their data across networks are not sensitive to the latency of the data transfers, there are a number of interactive-oriented applications, such as Telnet and rlogin operating over TCP, which expect prompt response from their corresponding peers.

In this thesis, a self-tuning PD based TCP congestion control scheme is proposed to solve these problems associated with existing TCP implementations, while retaining current TCPs' strengths of adaptability and scalability.

To avoid the shortfalls associated with the existing heuristic approaches, the congestion control scheme in ST-PD TCP is formulated from a control-theoretical perspective. The objective of TCP congestion control is to maximise the network resource utilisation without causing congestion. This goal can be realised by keeping the bottleneck node's buffer occupancy of each TCP connection around a certain level, so that each bottleneck link cannot

be starved and the bottleneck queue size is bounded well within the physical buffer dimension allocated at that node.

The congestion control entity in ST-PD TCP has a clear and well defined control objective — keeping the bottleneck node's buffer occupancy around a designed level. The congestion control entity in ST-PD TCP estimates the current bottleneck node's buffer occupancy from its recent transmission rate and the RTT variations of the TCP connection. The difference between the estimated buffer occupancy and the designed buffer occupancy forms the control error. The congestion control entity in ST-PD TCP applies the proportional and derivative control principle to adjust its congestion window size. This aims to minimise the control error, thereby drawing the bottleneck node's buffer occupancy back to the designed level.

To optimise the PD control performance in ST-PD TCP, a fuzzy logic controller is employed to tune the control gains of the PD controller based on the source node perceived bandwidth-RTT product of the TCP connection. Because of this, the PD-based congestion control scheme in ST-PD TCP works effectively over connections with a broad range of bandwidth-delay product characteristics.

Since the increase/decrease of the congestion window size in ST-PD TCP is proportional to how far the estimated bottleneck node's buffer occupancy has deviated from the designed level, ST-PD TCP source nodes can adapt their transmission rates swiftly to the network load variations. That is, if there is extra available bandwidth, ST-PD TCP connections can quickly grab it. If the network is going to be overloaded, ST-PD TCP connections can cut their congestion window sizes intelligently to reduce the load. The aggressiveness of these congestion window cuts depends on the extent of the load condition. This feature of ST-PD TCP is confirmed by both simulations and experiments, in which several ST-PD TCP connections share the same bottleneck link, where the data transfers are started and ended at different times. The results of these simulations and experiments demonstrate that when new connections join in, the source nodes of existing connections reduce their transmission rates swiftly. When a connection is terminated, the remaining connections quickly grasp the extra available bandwidth.

Because the congestion control entity in ST-PD TCP has a clear control objective, ie. keeping the bottleneck node's buffer occupancy around the designed level, ST-PD TCP source nodes know when to stop their congestion window increase, and thereby avoid buffer overflows. A direct benefit resulting from this feature is that transferring the same amount of data takes ST-PD TCP less time to complete compared with existing TCPs even in wired networking environments. This is confirmed by a set of experiments involved using TCP Reno and ST-PD TCP to transfer 100,000 bytes of data over connections with a range of bandwidth and propagation delay characteristics. In all circumstances, ST-PD TCP connections finish the data transfer somewhat earlier than TCP Reno connections.

The more important implication of ST-PD TCP's ability to avoid buffer overflows is that ST-PD TCP no longer interprets packet losses as signals of network congestion and it does not rely on detecting packet losses to trigger its congestion window cut. The advantage brought about by this feature is that ST-PD TCP is able to operate more efficiently in networks consisting of unreliable transmission links than current TCPs. In the analytical studies to quantify the performance of ST-PD TCP over lossy links, the impact of wireless losses on the congestion window of ST-PD TCP is modelled as disturbances on the control errors estimated by the PD controller. The transient response analysis of the PD controller under such disturbances confirms that the congestion window of ST-PD TCP is less susceptible to wireless losses than the congestion windows of current TCPs, which are either cut by half or reduced to one after experiencing packet losses. Both simulations and experiments show that ST-PD TCP connections achieve significantly higher throughput than TCP Reno connections in wireless networking environments.

To demonstrate that the design of ST-PD TCP does not make any unrealistic assumptions, ST-PD TCP has been implemented into the kernel of Linux-2.2.17 and its performance was measured in a testbed network. The experimental results are consistent with the analytical and simulation studies conducted in both wired and wireless networking environments.

The coexistence of ST-PD TCP with current TCP deployment is just as important as the issues of performance enhancements. ST-PD TCP achieves the full functional compatibility with existing TCPs by allowing them to communicate with ST-PD TCP without requiring

any modifications. Further, the presence of ST-PD TCP connections does not degrade the throughput of connections in the same network that use current TCP implementations.

The throughput of ST-PD TCP connections may or may not be adversely affected by the TCP connections that use Reno-style congestion control schemes, depending on whether they share a bottleneck queue. Even taking the negative effects into account, ST-PD TCP still outperforms current TCPs in networks with unreliable transmission links. This will justify the deployment of ST-PD TCP in networks that contain terrestrial wireless or satellite components.

In summary, there is still plenty of room for the performance of existing TCP implementations to be improved, despite the fact that the TCP/IP protocol suite has served data communications very well. The work presented in this thesis is just an example of improving TCP performance by introducing a new congestion control scheme, while retaining current TCPs' strength of scalability and maintaining the newly devised TCP's functional compatibility and operational congeniality with existing TCP deployment.

Appendix. The Kernel Architecture of 4BSD Unix Networking Subsystem

Although TCP/IP is the first networking protocol suite implemented in Unix kernel, it is not the only one. Other networking protocols such as Novell IPX, X.25 and Appletalk are also supported. Further, a good kernel architecture must facilitate the integration of future emerging networking protocols. As a result, 4BSD provides a general interface to allow network-oriented applications to be developed independent of the underlying communication protocols. The unified interface for accessing kernel networking facilities is known as *socket* in 4BSD. In fact, the socket interface is so general that all interprocess communication facilities are built on it, ie. socket interfaces support interprocess communications both across networks and on the same machine. The networking subsystem architecture of 4BSD is shown in Figure A.1.

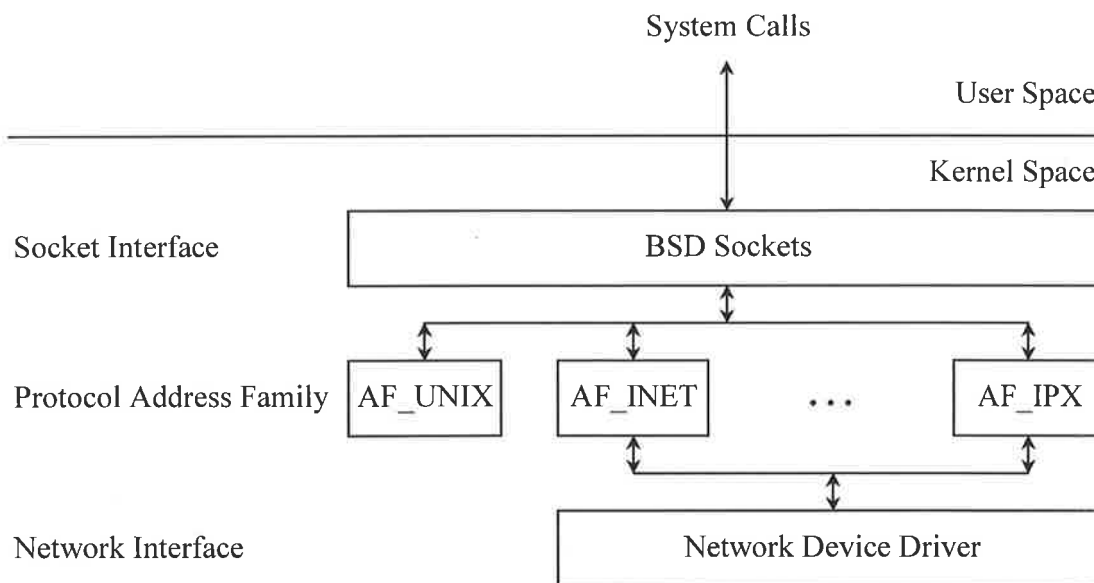


Fig. A.1 The networking subsystem architecture

From Figure A.1, it can be seen that the networking subsystem of 4BSD kernel supports different sets of communication protocols, known as Address Family (AF). TCP/IP is just a

part of the internet protocol address family, AF_INET, as shown in Figure A.1. The socket interface greatly simplifies the development of application programs running on top of the socket layer. If an application program wants to use a different protocol suite, it only needs to pass the name of appropriate protocol address family as a parameter during the *socket* system call. The socket interface also facilitates the integration of new protocols into the kernel.

The socket interface also provides services to interprocess communication programs running on the same machine. The protocol address family delivering such services is AF_UNIX. No physical network device is needed for AF_Unix based applications.

Figure A.2 shows the layered structure of the block AF_INET in Figure A.1.

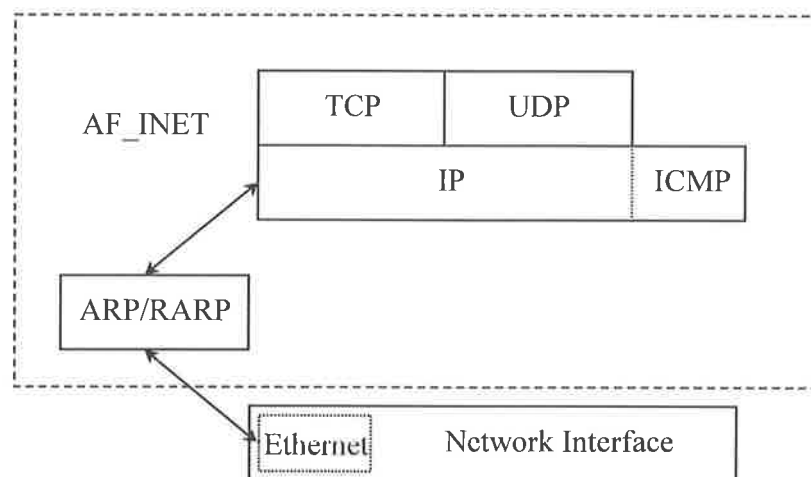


Fig. A.2 The layered structure of AF_INET block

Two types of transport protocols are integrated in AF_INET: TCP provides transport services for connection-oriented applications; and UDP provides transport services for connectionless applications. The network layer protocol in AF_INET is IP. Although the Internet Control Message Protocol (ICMP) uses IP to deliver its messages, it is regarded as a part of the IP. The Address Resolution Protocol (ARP) and the Reverse Address Resolution Protocol (RARP) are used to map between physical ethernet addresses and logical IP addresses. ARP and RARP are part of AF_INET.

REFERENCES

- [1] W. Stallings, "IPv6: The new Internet protocol", *IEEE Commun. Mag.* Vol. 34, pp. 96-108, Jul. 1996
- [2] L. Kleinrock, "Information Flow in Large Communication Nets", RLE Quarterly Progress Report, July 1961.
- [3] L. Roberts, "Multiple Computer Networks and Intercomputer Communication", ACM Gatlinburg Conf., October 1967
- [4] G. Ennis, "Development of the DoD protocol reference model," in *Proc. SIGCOMM'83 Symp.*, 1983
- [5] R. E. Kahn, Communications Principles for Operating Systems. Internal BBN memorandum, Jan. 1972
- [6] V. G. Cerf and R. E. Kahn, "A protocol for packet network interconnection", *IEEE Trans. Comm. Tech.*, vol. COM-22, V5, pp. 627-641, May 1972
- [7] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff, "A brief history of the Internet," *Communications of ACM*, Feb. 1997
- [8] S. J. Leffler, M. K. McKusick, M. J. Karels and J. S. Quarterman, The Design and Implementation of the 4.3BSD UNIX Operating System, Addison –Wesley Publishing Company
- [9] J. S. Quarterman and S. Wilhelm, UNIX, POSIX and Open Systems. Addison-Wesley Publishing Company, Inc. 1993

- [10] Department of Defence, *Military Standard Internet Protocol*, MIL-STD-1777, Aug. 1983
- [11] Department of Defence, *Military Standard Transmission Control Protocol*, MIL-STD-1778, Aug. 1983
- [12] C. Partridge and T. Shepard, "TCP Performance over Satellite Links," *IEEE Network*, vol. 11, no. 5, pp. 44-49, Sept. 1997.
- [13] R. Caceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," *IEEE J. Select. Areas Commun.*, vol. 13, no. 5, pp.850-857, June 1995.
- [14] C. Lefelhocz, B. Lyles, S. Shenker and L. Zhang, "Congestion control for best-effort service: Why we need a new paradigm," *IEEE Network*, vol. 11, no. 1, pp. 10-19, Jan. 1996.
- [15] V. Jacobson, "Congestion avoidance and control," in *Proc. SIGCOMM'88 Symp.*, Aug. 1988, pp. 314-329.
- [16] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Commun. Review*, vol. 26, no. 3, pp. 5-21, July 1996.
- [17] W. Stevens, *TCP/IP Illustrated*, Addison Wesley, 1994.
- [18] L. Brakmo and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE J. Select. Areas Commun.*, vol. 13, no. 8, pp.1465-1480, Oct. 1995.
- [19] Z. Wang and J. Crowcroft "A new congestion control scheme: Slow start and search (Tri-S)," *ACM Computer Commun. Review*, vol. 21, no. 1, pp. 32-43, Jan. 1991.

- [20] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," *ACM Computer Commun. Review*, vol. 19, no. 5, pp. 56-71, Oct. 1989.
- [21] S. Floyd, T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 2582, April, 1999
- [22] Kolarov, A., and Ramamurthy, G.: 'A control-theoretic approach to the design of an explicit rate controller for ABR service', *IEEE/ACM Trans. Netw.*, 1999, 7, (5), pp. 741-753
- [23] Pitsillides, A., Sekerciogly, A., and Ramamurthy, G.: 'Fuzzy backward congestion notification (FBCN) congestion control in asynchronous transfer mode (ATM) networks', in *Proc. IEEE GLOBECOM'95*, Singapore, pp. 280-285
- [24] S. Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Commun. Review*, vol. 24, no.5, pp. 10-23, Oct. 1994.
- [25] W. Qian, and M. Mizumoto, "PID type fuzzy controller and parameters adaptive method," *Fuzzy Sets and Systems*, vol. 78, pp. 23-35, 1996.
- [26] K. Astrom and T. Hagglund, *PID Controllers – Theory, Design and Tuning*, 2nd edn, Instrument Society of America, 1995.
- [27] C. Altrock, "Fuzzy logic and neurofuzzy applications explained," Prentice Hall, 1995
- [28] Online www.isi.edu/nsnam/ns
- [29] RD-LAP Specifications, Motorola
- [30] RAM Mobile Data System Overview, Release 5.2, RMDUS 031-RMDSO-RM, Oct., 1994

- [31] Cellular Digital Packet Data System Specification 1.0, July 19, 1993
- [32] G. Brasche, B. Walke, "Concepts, Services and Protocols of the New GSM Phase 2+: General Packet Radio Service," *IEEE Commun. Mag.* Vol. 34, pp. 94-104, Aug. 1997
- [33] T. Magedanz, "Integration and Evolution of Existing Mobile Telecommunication Systems towards UMTS," , *IEEE Commun. Mag.* Vol. 34, pp. 90-96, Sept. 1996
- [34] B. P. Crow et al, "IEEE 802.11 Wireless Local Area Networks," , *IEEE Commun. Mag.* Vol. 34, pp. 116-127, Sept. 1997
- [35] W. Honcharenko, " Broadband Wireless Access," , *IEEE Commun. Mag.* Vol. 34, pp.20-26, Jan. 1997
- [36] P. Papazian and G. Hufford, "Initial Study of the Local Multipoint Distribution Service Radio Channel," Proc. Wireless'96 conf., Calgary, Alberta, Canada, July 1996 pp. 494-513
- [37] J. Farserotu, R. Prasad, "A Survey of Future broadband Multimedia Satellite Systems, Issues and Trends," *IEEE Commun. Mag.* Vol. 30, pp. 90-98, May 1996
- [38] <http://www.hns.com/spaceway>
- [39] <http://www.skkybridgesatellite.com>
- [40] <http://www.teledesic.com>
- [41] B. Sklar, "Digital Communications: Fundamentals and Applications," Englewood Cliffs, NJ: Prentice Hall, 1988
- [42] T. S. Rappoport, "Wireless Communications," Upper Saddle River, NJ: Prentice Hall, 1996
- [43] B. Sklar, "Rayleigh Fading Channels in Mobile Digital Communication Systems," *IEEE Commun. Mag.* Vol. 34, pp. 136-146, Sept. 1997

- [44] H. Balakrishnan, V. Padmanabhan, S. Seshan and R. Katz, "A comparison of mechanisms for improving TCP performance over wireless links", *IEEE/ACM Trans. Netw.*, vol. 5 pp. 756-769, May 1997
- [45] S. Lin and D. J. Costello, "Error Control Coding: Fundamentals and Applications," Englewood Cliffs: NJ: Prentice Hall 1983
- [46] E. Ayanoglu et al, "AIRMAIL: A link-layer Protocol for Wireless Networks," *ACM/Baltzer Wireless Networks J.*, vol. 1, pp.47-60, Feb. 1995
- [47] S. Nanda, R. Ejzak, and B. T. Doshi, "A Retransmission Scheme for Circuit-mode Data on Wireless Links," *IEEE J. Select. Areas Commun.*, vol. 12, Oct. 1994
- [48] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," *ACM Wireless Networks*, vol. 1, Dec. 1995
- [49] R. Yavatkar and N. Bhagwat, "Improving end-to-end performance of TCP over mobile internetworks," in *Mobile 94 Workshop Mobile Computing Syst. Appl.*, Dec. 1994
- [50] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in Proc. 15th Int. Conf. Distributed Computing Syst., May 1995
- [51] S. Floyd, "TCP and Explicit Congestion Notification," *ACM Computer and Communication Review*, vol 24, pp. 10-23, Oct. 1994
- [52] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," *Internet RFC 1323*, 1992
- [53] R. Braden, "T/TCP — TCP Extensions for Transactions, Functional Specification," *Internet RFC 1644*, 1994
- [54] J. Postel, "Transmission Control Protocol," *Internet RFC793*, 1981

-
- [55] R. Braden, "Requirements for Internet Hosts — Communication Layers," *Internet RFC1122*, 1989
 - [56] J. R. Hauser, SoftFloat Release 2a, December 13, 1998
 - [57] <http://www.tcpdump.org>
 - [58] Mitsubishi Microcomputers M16C/62 Group User's Manual
 - [59] The Computational Science Education Project, "Random Numbers".
 - [60] S. L. Anderson, "Random Number Generators on Vector Supercomputers and Other Advanced Architectures," *SIAM Review*, 32(2), pp. 221-251, 1990
 - [61] S. K. Park and K. W. Miller, "Random Number Generators: Good Ones are Hard to Find," *Transactions of the ACM*, Nov. 1998