ENHANCING TCP PERFORMANCE IN WIDE-AREA WIRELESS NETWORKS USING END-TO-END BANDWIDTH ESTIMATION

BY

KHANDOKER NADIM PARVEZ

A Thesis submitted to
the Faculty of Graduate Studies
In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

Enhancing TCP Performance in Wide-Area Wireless Networks Using End-to-End Bandwidth Estimation

BY

Khandoker Nadim Parvez

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Of

MASTER OF SCIENCE

Supervisor: Prof. Ekram Hossain

ABSTRACT

A comprehensive study on the performances of the basic TCP (Transmission Control Protocol) variants (e.g., TCP Tahoe, TCP Reno, TCP NewReno, SACK TCP, FACK TCP) in wide-area cellular wireless networks is presented. Impacts of variations in wireless channel error characteristics, number of concurrent TCP flows and wireless link bandwidth on the average TCP throughput and fairness performances are investigated. To this end, a new TCP variant, namely, *TCP Prairie*, based on dynamic bandwidth estimation and sender-only modification is proposed for wide-area wireless networks. Two novel mechanisms are incorporated in *TCP Prairie* - one to obtain bandwidth sample under bursty environment and the other to estimate the available bandwidth using the bandwidth samples. The key idea of the protocol is to control the transmission rate at the TCP sender based on bandwidth measured from the rate of returning acknowledgements. An in-depth analysis of the system behavior is presented and compared with another TCP variant (*TCP Westwood*) which also uses a congestion control mechanism based on bandwidth estimation. Simulation results obtained using *ns-2* show that *TCP Prairie* offers superior performance compared to *TCP Westwood*.

Examiners:

Prof. Ekram Hossain, Supervisor, Dept. of Electrical & Computer Engineering

---

Prof. R. D. Mcleod, Member, Dept. of Electrical & Computer Engineering

---

Prof. Jelena Misic, External Examiner, Dept. of Computer Science

Table of Contents

List of Figures

List of Tables

Acknowledgement

I would like to express my heartiest appreciation to my supervisor Professor Ekram Hossain for his continuous support and active guidance during this research work. His supervisions and experiences were vital to shape my ideas into this dessertation. I would like to acknowledge with gratitude the support from TR*Labs*, Winnipeg, during the course of my graduate studies at University of Manitoba.

I am very grateful to Professor R. D. Mcleod and Professor Jelena Misic for serving on my examination committee.

Last but not the least, I would like to extend my sincere thanks to all of my colleagues and friends here in Winnipeg for their friendship and cooperation.

Dedication

To my parents and brother who are the inspirations of all my efforts.

Chapter 1

Introduction

TCP is the transport layer protocol used along with the unreliable network layer protocol IP (Internet Protocol) in todays Internet for non-real-time applications. Since TCP/IP is the standard network protocol stack on the Internet, its use over the next-generation wireless mobile networks is a certainty. The use of TCP/IP as the network protocol stack in the future generation wireless networks will leverage the rapidly evolving Internet technology in the wireless domain and enable to provide seamless wide-area Internet service to mobile users [1]. Efforts in developing IP-centric wireless networks are ongoing and architectures and protocols for supporting multimedia traffic are evolving ([1]-[6]).

Transport protocol segments are transmitted in a wireless network as radio link level frames over the air-interface. In case of transmission to a mobile from a fixed host in the Internet, LLC (Link Level Control) protocol divides TCP segments into radio frames for transmission over the wireless link. Therefore, the performance of TCP will depend on the performance and service provided by the underlying radio link layer. In this thesis, however, we deal with the end-to-end (or transport-level) approaches while investigating the performance of TCP in a wide-area wireless network. High utilization of wireless link bandwidth and at the same time fairness towards competing TCP traffic are required in a wireless packet data network.

1.1 TCP (Transmission Control Protocol)

Transport layer protocol such as TCP operates on an end-to-end basis and it's performance is one of the most critical issues in data networking over wireless links. A transport layer protocol is responsible for managing end-to-end flow and congestion

control, providing reliability, security and QoS (Quality of Service) [7]. In OSI model, application protocol data unit (APDU) is passed down to the transport layer which is then called the transport service data unit (TSDU). A transport protocol header is added to this TSDU to form a transport protocol data unit (TPDU) and then it is passed to the network layer as a network service data unit (NSDU)[1].

TCP is a connection-oriented reliable transport protocol consisting of three phases of operations: connection setup, data transfer, and connection termination. The connection set up procedure uses a three-way handshake where the connection is established after both ends of the connection inform each other of the set up process. The connection termination procedure uses a four-way handshake where both the active and the passive terminators send their own set of two-way handshake messages. Again, TCP is a bi-directional transport protocol that can transmit and receive at the same time (i.e., within the same connection set up). TCP is a byte-oriented transport protocol since it passes data to its peer in a byte by byte manner. For error control, TCP uses sequence number, cumulative positive acknowledgements with piggyback option for successfully transmitted packets and retransmission-based end-to-end error recovery.

TCP uses a congestion control mechanism (to ensure network stability and prevent congestion collapse) which is interwined with a window-based flow control mechanism. The TCP sender detects a packet-loss either by the arrival of several duplicate cumulative ACKs (Acknowledgements) or by the absence of an acknowledgement during a timeout interval and it attributes the packet loss to network congestion. Upon detection of a packet loss at the TCP sender, the congestion control/avoidance mechanism is triggered which reduces the transmission window size multiplicatively and/or increases the retransmission timer exponentially, and consequently, the throughput is reduced.

The implementation of the TCP flow/congestion control mechanism is based on two sender-side state variables, namely, *congestion window (cwnd),* $W(t)$ and the *slow-start threshold (ssthresh),* $W_{th}(t)$ ([8]-[11]). During connection set up, the receiver advertises a maximum window size W_{max} and the TCP sender is not allowed

[1]In the Internet community, both NPDU and TPDU are known as *packet* even though in OSI terminology a *segment* is referred to as a TPDU. The terms 'packet' and 'segment' are used interchangeably in this thesis.

to have more than $min\,(W_{max}, W(t))$ unacknowledged data packets outstanding at any given time t. For a new connection, $W(t)$ is generally initialized to 1. The basic window adaptation mechanism (which is triggered by ACKs) in all currently available TCP implementations is as follows:

i. If $W(t) < W_{th}(t)$ each ACK causes $W(t)$ to be incremented by 1. This is the *slow-start* phase.

ii. If $W(t) \geq W_{th}(t)$ each ACK causes $W(t)$ to be incremented by $\frac{1}{(int)W(t)}$. This is the *congestion avoidance* phase.

In case of timeout, TCP source updates $W(t)$ and $W_{th}(t)$ as follows: $W_{th}(t+) = \lceil \frac{W(t)}{2} \rceil$ and $W(t+) = 1$ and then starts retransmitting from the first lost packet. However, when the TCP source receives a cumulative acknowledgement from the TCP sink acknowledging the already transmitted packets, it immediately starts transmission after the highest ACKed packet. After a timeout the sender also updates the RTO (Retransmission Time Out) value using exponential backoff. The backoff continues until an ACK is received for a packet transmitted exactly once.

TCP sinks can accept packets out of order but deliver them only in sequence to the user and they generate immediate ACKs. If a sink receives a packet out of order, it issues an ACK immediately for the last in order packet that was received. If a packet is lost (after a stream of correctly received packets), then the receiver keeps sending ACKs (called duplicate ACKs) with the sequence number of the first lost packet even if packets transmitted after the lost packet are correctly received.

1.2 TCP in Wide-Area Cellular Wireless Networks

While the "traditional" TCP has been tuned for the last two decades to optimize its performance in wired networks where congestion is the main cause of any packet loss, it performs poorly in wireless environments due to the following factors:

- *Transmission losses on wireless link*: While in the wired transmission media the bit error rates are of $O(10^{-6}) - O(10^{-8})$, wireless media suffer significantly higher bit error rates such as $O(10^{-3}) - O(10^{-1})$ [12]. Therefore, the packet loss rate in a wireless link is an order of magnitude higher than that in a wired link.

In a typical wireless scenario packet loss rate can be as high as 10% depending on channel fading and interference conditions and user mobility.

When a packet is lost, TCP performs end-to-end recovery by retransmitting the lost packet and frequent end-to-end retransmissions may reduce end-to-end throughput significantly. Again, any packet loss in the wireless link is misinterpreted by the TCP sender as a congestion loss and it triggers the congestion control mechanism which reduces the sender's transmission window size resulting in reduced end-to-end throughput. Therefore, in a wired-cum-wireless environment, random packet losses in the wireless link cause the TCP sender to underestimate the available network bandwidth and thereby reduces the application layer performance.

Also, the location and time-dependent channel errors may impact the achieved throughput fairness among multiple concurrent TCP flows[2] and this would presumably be different for the different TCP variants. Again, frequent channel errors and subsequent retransmissions may result in inefficient use of the limited battery power in the mobile devices.

- *Wireless link delays*: Presence of limited bandwidth wireless links in an end-to-end path generally results in increased end-to-end transmission delays (and hence longer round-trip times (RTTs)). Although 2.5G and 3G cellular wireless networks (e.g., EGPRS, UMTS, IMT-2000) will have increased bandwidth, factors such as link asymmetry due to different uplink and downlink bandwidth, bandwidth oscillation due to dynamic resource allocation, stronger FEC (Forward Error Correction) coding, interleaving, radio link level recovery will result in large BDP (Bandwidth-Delay Product)[3] wide-area cellular wireless networks [13].

  Since the rate of increase in the congestion window size at the TCP sender is proportional to the rate of incmoing ACKs, the congestion window may increase at a much lower rate in the presence of wireless links and thus reduce the end-to-end throughput. In fact, the throughput of a TCP connection has been shown to vary as the inverse of the connections RTT. Given a packet loss rate p, the

[2]The terms 'flow' and 'connection' are used interchangeably.

[3]The bandwidth-delay product for a TCP connnection refers to the product of the round trip delay (T) for the connection and the capacity of the bottleneck link (μ) in its path.

maximum sending rate for a TCP sender is γ *bytes/sec*, for $\gamma \leq \frac{1.5*\sqrt{2/3}*B}{T*\sqrt{p}}$ where B is the packet size and T is the round trip time [14]. The inherent TCP bias against flows with longer RTT results in throughput unfairness among flows traversing the same number of hops but having different number of wireless links in the end-to-end path.

- *User mobility*: During a TCP session when a mobile user in a wireless cellular network moves from one cell to another, all necessary information must be transferred from the previous base station (BS) to the new base station which might cause a short duration of disconnection (typically of the order of several hunderds of milliseconds) during which no transmission takes place. Similar to that in the case of link errors, delays and packet losses during this "handoff" scenario trigger TCP congestion control mechanisms which results in reduced end-to-end throughput [15].
- *Short-lived TCP flows*: Data services which are more transactional than streaming in nature (e.g., web browsing on a smart phone, email) usually involve transmission of a rather small amount of data, and therefore, tend to require short TCP connection duration. It is possible that the entire transmission is completed while the TCP sender is in the slow-start phase and thus resulting in the under utilization of the network capacity.

1.3 Objective of the Work

The objectives of the work presented in this thesis are as follows:

- Investigate the effectiveness of the different end-to-end TCP strategies in a wide-area wireless networks under different wireless link conditions (e.g., random error, correlated error).
- Develop efficient end-to-end TCP for wired-wireless networks.
  The motivation behind this is the fact that when IPSEC (IP Security) or other security mechanisms are employed to encrypt IP payloads, true end-to-end techniques based on transport level modifications of TCP would be required for wide-area wireless networks. Also, when the data and the ACKs traverse different paths (e.g., in satellite networks), non-end-to-end solutions may cause

serious problems, while end-to-end solutions are applicable over any type of networks and links.

1.4 Thesis Outline

Subsequent chapters of this thesis are organized as follows:

- **Chapter 2** provides a unified study of different transport layer protocol approaches to TCP design for wide-area wireless networks. Performances of the different basic TCP variants including the newer variants such as TCP SACK (Selective Acknowledgment), TCP Vegas, TCP FACK (Forward Acknowledgement) are evaluated in a wide-area cellular network in terms of TCP throughput and fairness (in the case of multiple competing TCP flows).
- **Chapter 3** summarizes the different transport level approaches for TCP enhancements. Both the end-to-end and non-end-to-end (mainly base station centric) approaches are considered.
- **Chapter 4** analyzes the behavior of *TCP Westwood* which is a recently proposed TCP variant based on end-to-end bandwidth estimation. In particular, the deficiency of the bandwidth estimation technique in *TCP Westwood* is revealed, and the impact of the erroneous bandwidth estimation on the the throughput and friendliness behavior of *TCP Westwood* are analyzed.
- In **Chapter 5**, we propose a new TCP variant, namely, *TCP Prairie* that solves the problem of erroneous bandwidth estimation in *TCP Westwood* by using a novel bandwidth estimation technique. The efficacy of the bandwidth estimation technique in *TCP Prairie* is revealed and the throughput, fairness, friendliness performances are analyzed.
- **Chapter 6** summarizes the contributions of the thesis and discusses a few directions for possible future research.

Chapter 2

Performance of Basic TCP Variants in Wide-Area Wireless Networks

A comprehensive study on the performances of the basic TCP variants (e.g., TCP Tahoe, TCP Reno, TCP New-Reno, SACK TCP, FACK TCP) in wide-area cellular wireless networks is presented. Average throughput and fairness performances of TCP are investigated under varying channel error rate and wireless link bandwidth. The maximum achievable throughput under window-based end-to-end transmission control is also evaluated.

2.1 Related Work

Prior research closest to this work is that reported in [16], where the throughput performances of TCP Tahoe, OldTahoe, Reno and New-Reno in a localized wireless network were investigated under random packet losses using a stochastic model. The effects of coarse TCP timeout and the number of duplicate acknowledgements on TCP throughput performance were particularly emphasized. More general findings were reported in [17], where performance of TCP was investigated for a wide-area wireless network, particularly, for a network with high bandwidth-delay product and random packet losses. It was observed that random loss could lead to significant throughput

degradation when the product of the packet loss (p) probability and the square of the bandwidth-delay product (i.e., $(\mu T)^2$, μ is the bandwidth and T is the delay) is larger than one. Also, TCP's unfairness towards connections with higher round trip delays was reported. A packet level performance comparison among TCP Tahoe, Reno, New-Reno and SACK in a wide area wired-Internet environment was presented in [18] considering scenarios where the number of packet drops in a transmission window varies from one to four.

2.2 Basic TCP Variants

2.2.1 TCP Tahoe

TCP Tahoe uses *slow-start*, *congestion avoidance* and and *fast retransmit* mechanisms [9]. During *slow-start*, the congestion window increases exponentially (by one for each acknowledgement received) until it reaches the slow-start threshold (*ssthresh*), and during *congestion avoidance* the congestion window increases linearly by one per round trip time (RTT). The TCP sender goes into the *fast retransmit* mode when it receives *tcp_rexmt_thresh* (which is usually set to 3) number of duplicate acknowledgements. During *fast retransmit*, the sender retransmits the lost segment and enters into the *slow-start* phase by setting the congestion window to 1 and *ssthresh* to the half of congestion window. In addition, it forgets all outstanding data transmitted earlier [18]. When the loss is due to sporadic channel error, switching to *slow-start* mode causes the throughput to fall.

2.2.2 TCP Reno

TCP Reno [10] is similar to TCP Tahoe except that in addition to *fast retransmit*, it also includes the *fast recovery* mechanism for a single segment loss. When the TCP sender receives *tcp_rexmt_thresh* number of duplicate acknowledgements, instead of switching to *slow-start* after *fast retransmit*, TCP Reno enters into *fast recovery*. During *fast recovery*, the sender sets *ssthresh* to the half of the congestion window and the new congestion window to the new *ssthresh* plus the number of duplicate acknowledgements received. TCP Reno remains in *fast recovery* until the lost segment

which triggered the *fast retransmit* has been acknowledged. When the sender receives new acknowledgement(s), it exits *fast recovery* and resets the congestion window to *ssthresh* and thereby moves into congestion avoidance.

In case of congestion loss, the *fast recovery* mechanism keeps the average congestion window size high resulting in better throughput performance compared to TCP Tahoe. During *fast recovery*, each new duplicate acknowledgement increases the congestion window size by one. Although TCP Reno works fine for single loss, in case of multiple losses from the same transmission window the performance suffers since it exits *fast recovery* and enters into it again in a repeated fashion or goes to timeout.

2.2.3 TCP New-Reno

TCP New-Reno uses an augmented *fast recovery* mechanism where, unlike TCP Reno, *fast recovery* continues until all the segments which were outstanding during the start of the *fast recovery*, have been acknowledged [19]. This strategy helps to combat multiple losses without entering into *fast recovery* multiple times or causing timeout. In this case, a *partial acknowledgement*[1] is considered as an indication that the segment following the acknowledged one has been dropped from the same transmission window (or flight), and therefore, TCP New-Reno immediately retransmits the other lost segment indicated by the *partial acknowledgement* and remains in *fast recovery*. It takes one round trip time to detect each lost segment and to retransmit it.

2.2.4 SACK TCP

In SACK TCP, the receiver sends acknowledgements with SACK (Selective Acknowledgement) option when it receives out of order segments due to loss or out of order delivery [20]. The SACK option field contains a number of SACK blocks, where each SACK block reports a non-contiguous set of data that has been received and queued. The first block in SACK option reports the most recently received block.

The SACK TCP sender is an intelligent extension of that in TCP Reno. It only modifies the *fast recovery* mechanism of TCP Reno keeping the other mechanisms

[1]This refers to a new acknowledgement received during *fast recovery* which acknowledges some but not all of the packets that were outstanding at the start of the *fast recovery* phase.

unchanged. Similar to New-Reno, it can handle multiple packet losses from the same flight. It has a better estimation capability for the number of outstanding segments. The acknowledgement with SACK option enables the sender to determine explicitly which segments have been received or have been lost. To keep track of the acknowledged and lost segments, it maintains a data structure called *scoreboard.* Whenever the sender is allowed to transmit based on the congestion window size and the number of outstanding segments, it consults the *scoreboard* and transmits the missing segments. If there is no missing segment to retransmit, it transmits new segments. When a retransmitted segment is dropped, the sender detects it by a retransmit timeout. In case of timeout it retransmits the segment and enters into the *slow-start* phase.

SACK TCP maintains a variable called *pipe* to keep track of the number of outstanding segments. For each retransmission or new transmission, the sender increases *pipe* by one and for each received acknowledgement, it decreases *pipe* by one. For each received partial acknowledgement, *pipe* is decreased by two with the assumption that the original packet and the retransmitted packet have left the network. Since *pipe* represents the amount of outstanding data, the sender transmits only when *pipe* is less than the congestion window.

2.2.5 FACK TCP

FACK (Forward Acknowledgement) TCP is a variant of SACK TCP with a modified *fast recovery* mechanism [21]. It uses a better technique for estimating the number of outstanding segments. For this, it introduces two new variables *snd.fack* and *retran_data*, where *snd.fack* represents the forward most data held by the receiver and *retran_data* represents the size of the retransmitted data outstanding in the network.

In non-recovery states *snd.fack* is updated using the acknowledgement number in the TCP header and is equivalent to *snd.una* [8]. But when a SACK block is received during error reocovery, it is updated to the highest sequence number received by the receiver plus one regardless of the number of the intermediate dropped segments. Therefore, the outstanding data is estimated as *(snd.nxt - snd.fack) + retran_data.*

For each retransmission, the size of the *retran_data* is incremented by the corresponding segment size. When retransmitted segment leaves the network it is decreased

by the same size. FACK TCP compares it's estimate for outstanding data against the congestion window size and decides on the number of transmissions. A timeout is forced in the case of loss of a retransmitted segment on the assumption that the congestion is persistent. It enters into the *fast recovery* mode when the sender receives *tcp_rexmt_thresh* number of duplicate acknowledgements or (*snd.fack* - *snd.una* > *tcp_rexmt_thresh* $*MSS$)[2]. In the case that several segments are lost but fewer than *tcp_rexmt_thresh* number of duplicate acknowledements have been received, the latter condition triggers the *fast recovery* phase sooner. Similar to New-Reno and SACK, FACK TCP terminates it's recovery phase upon receiving acknowledgement for all the segments that were outstanding during transition to *fast recovery*.

2.2.6 TCP Vegas

TCP Vegas [22] comes with a proactive congestion control mechanism in which network congestion is predicted based on the estimated *expected throughput* and the *actual throughput*. Expected throughput is estimated as *WindowSize/BaseRTT*, where *BaseRTT* is the minimum of all the measured RTTs and *WindowSize* is the size of the current congestion window. The actual throughput is calculated from the RTT for a tagged segment and the number of segments transmitted within that RTT. TCP Vegas compares the difference between the expected and the actual throughput against two thresholds α and β (where $\alpha < \beta$) and adjusts the transmission window accordingly. If the difference is smaller than α, the congestion window is increased linearly (by one per RTT) under the assumption that there is unutilized bandwidth available in the network. On the other hand, if the difference is larger than β, the congestion window is decreased, while if the difference lies between α and β the congestion window remains unchanged.

TCP Vegas has a fine grained timer expiry calculation mechanism to support early switching to *fast retransmit*. For this, the sender reads and records the system clock each time a segment is transmitted. When an acknowledgement arrives, it reads the clock again and calculates the fine grained RTT. TCP Vegas uses this fine-grained RTT estimate to calculate RTO. For each duplicate acknowledgement received, the

[2]MSS refers to the *Maximum Segment Size*.

Table 2.1. *Differences among the basic TCP variants.*

TCP Variant	Sender mechanism(s)
Tahoe	No *fast recovery*
Reno	Uses *fast recovery* to recover from single packet drops
New-Reno	Uses *fast recovery* to recover from multiple packet drops
SACK	Uses *fast recovery* to recover from multiple packet drops, better outstanding data estimation mechanism and larger retransmission pool
FACK	Similar to SACK, but uses a better outstanding data estimation mechanism and switches to *fast recovery* faster
Vegas	Uses proactive mechanism to control congestion window, switches to *fast retransmit* faster due to the fine-grained timer

sender checks the *Vegas expiry*[3] and if Vegas expiry occurs, the sender switches to *fast retransmit*. Similar to the other TCP variants, it switches to *fast retransmit* when it receives *tcp_rexmt_thresh* number of duplicate acknowledgements. Also, the sender switches to *slow-start* whenever the usual timeout occurs.

2.2.7 Comparison Among the Basic TCP Variants

Although the TCP senders for all the basic TCP variants rely on using binary positive acknowledgement, similar RTO estimation and timeout mechanisms, they use different congestion avoidance mechanisms. The major differences among the basic TCP variants are shown in Table 2.1.

The main difference between TCP Tahoe and TCP Reno is that the latter uses the *fast recovery* mechanism while the former does not. Although TCP Reno can

[3]Timeout caused by this fine-grained timer is referred to as 'Vegas expiry'.

handle single drop in a flight efficiently, it cannot handle multiple packet drops in a single flight very well. Tahoe does not memorize outstanding data when it switches to slow-start (due to multiple duplicate ACKs), but Reno does. Both TCP New-Reno and SACK TCP can handle multiple losses in a single flight more efficiently.

SACK TCP is different from TCP New-Reno in that SACK TCP can retransmit selectively. While TCP New-Reno retransmits starting from the segment corresponding to the duplicate acknowledgement or partial acknowledgement, SACK TCP retransmits this segment along with other missing segments from it's *scoreboard.* That is, the retransmission pool for a New-Reno sender can have at most one segment regardless of the number of packet drops. Therefore, it takes one RTT to recover each packet loss. On the other hand, since the retransmission pool for a SACK sender can have many lost segments, the recovery process becomes faster. Also, due to the *pipe* variable, SACK TCP has a better estimation of the number of outstanding segments.

The main advantage of FACK TCP over SACK TCP is that, by using *retran_data* and *snd.fack* the former performs a better estimation of the number of outstanding segments than the latter. Generally, the *pipe* estimation for FACK TCP becomes smaller compared to that for SACK TCP which allows the former to transmit more segments. Also, using the information from SACK block, sometimes it switches to *fast recovery* before receiving *tcp_rexmt_thresh* number of duplicate acknowledgements which is helpful when the transmission window size is small or the number of duplicate acknowledgements is too few to trigger *fast retransmit.*

By using *expected throughput*, *actual throughput* and the threshold parameters α, β, TCP Vegas can have a better estimation of the available bandwidth compared to the other TCP variants. Due to its own fine-grained timer management, TCP Vegas can switch to *fast retransmit* earlier which contributes to it's performance improvement. Also, since it reduces the congestion window to 3/4 instead of 1/2 during *fast retransmit* (when the segment that triggered *fast retransmit* has not been transmitted more than once), it helps to combat losses due to the sporadic wireless channel errors more efficiently.

2.3 Analysis of System Dynamics and Performance Evaluation

2.3.1 Simulation Model and Performance Measures

System dynamics under the different basic TCP variants, namely, TCP Tahoe, Reno, New-Reno, SACK, FACK and Vegas are investigated using *ns-2* [23] under varying wireless channel bandwidth, round trip delay, number of concurrent TCP connections and different wireless channel error characteristics in a wired-cum-wireless scenario where the mobile nodes act as TCP sinks. Also, two variants of New-Reno – one with *TCP-aware link level retransmission* (TLLR) and the other with *delayed acknowledgement* (DACK)-capable TCP sink, are considered.

It is assumed that the slowly varying shadow and path losses in a wireless link are perfectly compensated and that the channel quality variations due to multipath fading remain uncompensated. For a broad range of parameters, the packet error process in a mobile radio channel, where the multipath fading is considered to follow a Rayleigh distribution, can be modeled using a two-state Markov chain with transition probability matrix M_c [24]: $M_c = \begin{bmatrix} p & 1-p \\ 1-q & q \end{bmatrix}$ where p and 1-q are the probabilities that the jth packet transmission is successful, given that the (j-1)th packet transmission was successful or unsuccessful, respectively.

Also, the fading envelope is assumed to not change significantly during transmission of a TCP packet (which is of duration, say, t_{tcp}). The channel variation for each of the mobiles is assumed to be independent and determined by the two parameters p and q which, in turn, are determined by $f_d t_{tcp}$ [4], the normalized Doppler bandwidth and P_E, the average packet error probability. P_E describes the channel quality in terms of *fading margin* F [5]. Different values of P_E and $f_d t_{tcp}$ result in different degree of correlation in the fading process. When $f_d t_{tcp}$ is small, the fading process is highly correlated; on the other hand, for large values of $f_d t_{tcp}$, the fading process is almost independent. For a certain value of the average packet error rate P_E, the

[4]Here, $f_d = v/\lambda$ = mobile speed/carrier wavelength. The value of t_{tcp} determines the minimum fade duration.

[5]It refers to the maximum fading attenuation which still allows correct reception of a packet.

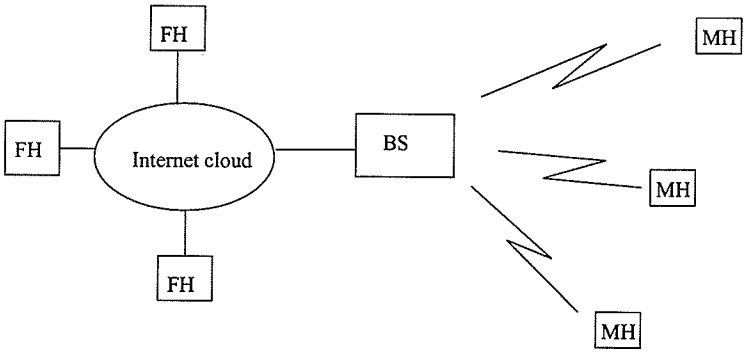

Figure 2.1. *Simulation topology for wide-area cellular wireless network.*

error burst-length increases as user mobility (and hence $f_d t_{tcp}$) decreases.

The network topology used in the simulation is shown in Fig. 2.1 (FH $\equiv$ Fixed host, MH $\equiv$ mobile host, BS $\equiv$ base station/wireless router). The wirless link is considered as bottleneck link and its delay is considered negligible. The round trip time is 60ms. TCP segment size is 1KB. The different performance measures are: *average per-flow throughput (γ)* (i.e., average amount of data successfully transmitted to a TCP sink per unit time), *energy consumption (e)* (average amount of energy dissipated in a TCP sink for successful reception of 1 MB of data) and *throughput fairness (f)*. For n concurrent TCP flows, the fairness index f is calculated as follows[6]:

$$f = \frac{(\sum_{i=1}^{n} \gamma_i)^2}{n \times \sum_{i=1}^{n} \gamma_i^2}. \tag{2.1}$$

To calculate the energy consumption at the mobile nodes (primarily due to transmission of acknowledgements and reception of TCP packets), the ratio of transmisson power and receive power wattage is assumed to be 3 for all the simulation scenarios described in this chapter.

[6]This is similar to the fairness function used in [25] to quantify the fairness in a shared resource system with n users: $F = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2}$ (where x_i is the ith user's throughput).

2.3.2 Simulation Results and Analysis

2.3.2.1 Single TCP flow with unifrom Wireless Loss (Scenario 1)

The wireless link bandwidth is assumed to be 2 *Mbps*. In this scenario, for the *delayed acknowledgemnt* case, the interval for delaying the acknowledgement is assumed to be 10 *ms*. The simulation run time is taken to be 1000 *seconds*.

Under random error conditions, TCP New-Reno and SACK TCP provide remarkable performance improvement over TCP Tahoe and TCP Reno (Fig. 2.2). Although TCP Tahoe and TCP Reno exhibit similar performance, TCP Reno performs slightly better than TCP Tahoe when the error rate is smaller than 1.5%.

Analysis of the system events such as the total number of timeouts (TO), total number of *fast retransmit/fast recovery* (FR), number of *fast retransmits* due to multiple packet drop in a transmission window (MD), number of segments newly transmitted during *fast recovery* (SND), and average of the transmission window sizes measured at the epoches of transitions to *fast retransmit* (FLT) reveal that under random error conditions the system dynamics for both TCP New-Reno and TCP SACK are more or less similar (Table 2.2), and therefore, their long-term performances are observed to be fairly close. FACK TCP is observed to perform better than each of TCP New-Reno and SACK TCP. TCP Vegas shows significant performance improvement over FACK TCP. However, the throughput performance of each of the above basic TCP variants is inferior to the performance of TCP New-Reno with TLLR (Fig. 2.2).

For both TCP Tahoe and TCP Reno, the number of timeouts increases with increasing error rate, and the difference in the number of timeouts for these two cases during a certain observation interval is observed to be small. As the error rate increases, Reno suffers due to multiple packet drops in the same flight and also the number of timeouts increases compared to the number of *fast retransmits*. This offsets Reno's gain due to *fast recovery* and makes the performance closer to that of Tahoe.

As is observable from Table 2.2, at 1% error rate, for SACK TCP there are 115 events of *fast recovery* due to multiple packet drops (within a transmission window) among the total of 1252 *fast recovery* events, while for TCP New-Reno, among the total of 1237 *fast recovery* events 129 *fast recovery* events are due to multiple packet drops. Therefore, the ratio of *multiple drop fast recovery* to total *fast recovery* is

Table 2.2. *Fast retransmit/recovery analysis under random error loss (for 1000* second *simulation run)*

TCP variant	TO	FR	MD	SND	FLT
	Error rate = 1%				
Tahoe	53	1056			
Reno	100	1181	110	1909	11.59
New-Reno	27	1237	129	2013	11.92
SACK	25	1252	115	2435	11.88
FACK	17	1365	137	6668	12.27
TLLR	0	14	1	2	15.00
Vegas	22	1552 1072E			13.04
	Error rate = 10%				
Tahoe	691	295			
Reno	692	277	65	359	5.55
New-Reno	705	315	88	442	5.55
SACK	685	298	83	341	5.51
FACK	630	392	125	426	5.28
TLLR	20	95	58	134	13.94
Vegas	546	550 334E			5.89

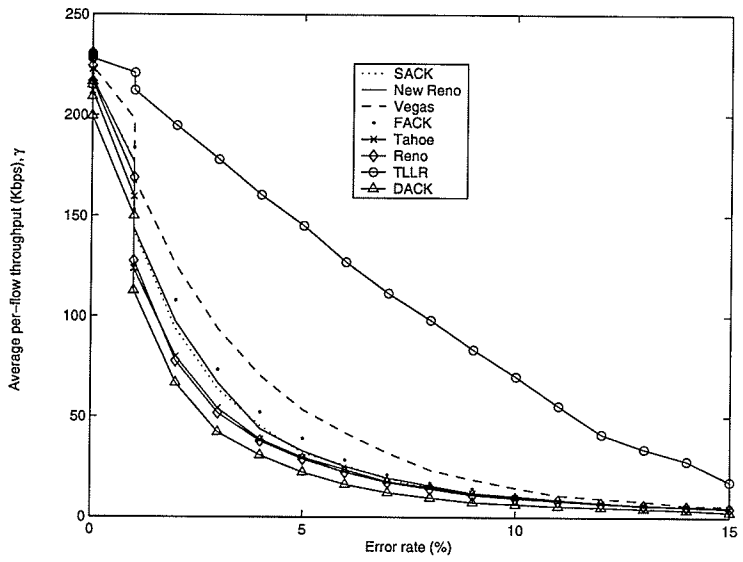

Figure 2.2. *Throughput performances of different TCP variants (scenario 1).*

smaller for SACK TCP. This observation holds for other error rates as well. In this case, SACK TCP cannot exploit the advantage of selective retransmission more effectively (compared to TCP New-Reno).

Again, under random error conditions, SACK TCP may not be able to exploit the advantage of better *pipe* estimation. Referring to Table 2.2, for 1% error rate SACK TCP transmits on the average 1.94 (= 2435/1252) new segments during *fast recovery*, while TCP New-Reno transmits on the average 1.63 (= 2013/1237) new segments. This small difference may not result in significant performance improvement for SACK TCP as compared to TCP New-Reno. As the error rate increases, the difference may become even smaller.

The *delayed acknowledgement* (DACK) variant of TCP New-Reno is observed to perform the worst, which is primarily due to its slow response to transmission failure.

Due to the better estimation of the number of outstanding segments, FACK TCP performs better than SACK TCP under all error conditions. At 1% and 10% error rate, for FACK TCP the number of new segments transmitted during *fast recovery* is observed to be 6668 and 426, respectively, while for TCP SACK, the numbers are 2435 and 341 (Table 2.2). With increasing error rate, as the effect of timeout becomes more dominant, the number of new segments transmitted during *fast recovery* decreases.

The performance improvement in TCP Vegas is mainly due to its two unique features – Vegas expiry and congestion window reduction by a factor of 3/4. The Vegas expiry mechanism causes it to react very fast to segment loss. As can be observed in Table 2.2, for 1% error rate, during 1000 *second* of simulation Vegas experiences 1552 *fast retransmit*s among which 1072 were detected early by the Vegas fine-grained expiry mechanism (in Table 2.2, 'E' refers to fast retransmit detected by Vegas expiry). The *Vegas expiry* mechanism also reduces the number of timeouts and it is observed that among all the TCP variants it experiences the smallest number of timeouts.

The TCP New-Reno with TCP-aware link level retransmission (TLLR) scheme (e.g., snoop protocol [26]) offers the best performance (Fig. 2.2) due to the fact that the 'snoop agent' at the link level eliminates most of the fast retransmits and timeouts. For example, with TLLR, for 10% error rate, the number of fast retransmits and the number of timeouts are 20, and 95, respectively, while for New-Reno they are 705 and 315, respectively. As a result, the average flight size is always very high (e.g., 13.94 for 10% error rate) which is close to the assumed maximum transmission window size of 15, and consequently, better throughput performance is achieved.

Regarding energy consumption for transmission of a fixed amount of data (1 MB) to a mobile node using the different TCP variants, it is observed that TCP Vegas requires the highest amount of energy (Fig. 2.3). This is due to the fact that, TCP Vegas results in the highest number of *fast retransmit*s. For TCP New-Reno with the DACK option, the energy consumption is observed to be the lowest, which is due to it's conservative acknowledgement transmission policy. But as the error rate increases, energy consumption due to reception of the retransmitted segments becomes more dominant compared to that due to transmission of acknowledgements, and consequently, energy consumption for TCP New-Reno with DACK tends to be similar to that for the other TCP variants (Fig. 2.3).

In fact, in the high error rate regime none of the end-to-end TCP mechanisms performs well in a wide-area cellular wireless environment. For example, with 10% error rate, the achieved thoroughput is about 10 $Kbps$ compared to 217 $Kbps$ in the ideal case. This is due to the inability of TCP to differentiate between wireless loss and congestion loss.

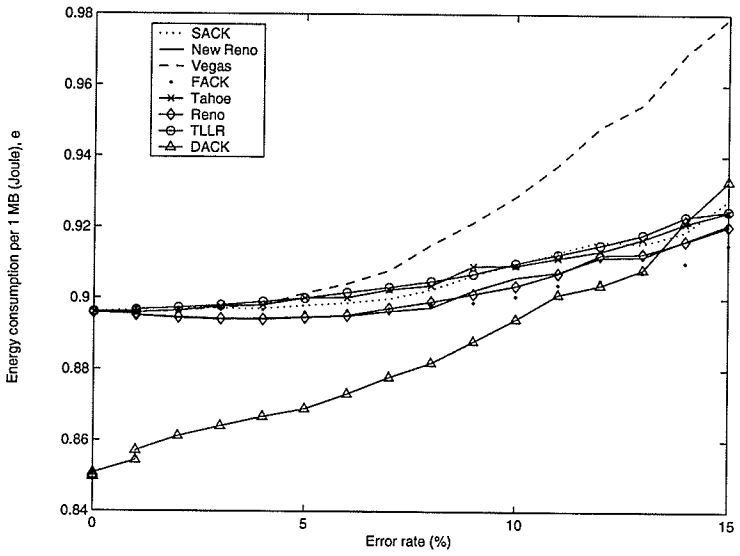

Figure 2.3. *Energy consumption for the different TCP variants (scenario 1).*

2.3.2.2 Multiple Flow with Uniform Wireless Loss (Scenario 2)

The wireless link bandwidth is assumed to be 10 *Mbps* and there are 5 concurrent flows. Throughput and energy consumption per flow under multiple concurrent TCP connections are similar to those in case of a single connection (Figs. 2.4-2.5). But the average per-flow throughput is observed to be slightly higher than the single flow case (scenario 1) when the error rate is not too high (e.g., $\leq 5\%$).

In this case, we also observe the achieved throughput fairness among the different competing TCP connections. All of the TCP variants provide good long-term fairness and it is observed that even for error rate as large as 9%, the fairness index lies above 0.99. For error rate larger than 14%, the fairness index reduces to 0.98 (Fig. 2.6). Therefore, it can be concluded that the random wireless channel errors do not impact the TCP throughput fairness for the different TCP variants remarkably.

2.3.2.3 Single Flow with Uniform Wireless Loss (High bandwdith bottleneck)(Scenario 3)

The wireless link bandwidth is assumed to be 10 *Mbps* in this case. This scenario is considered to compare the throughput performance in the case of multiple concurrent

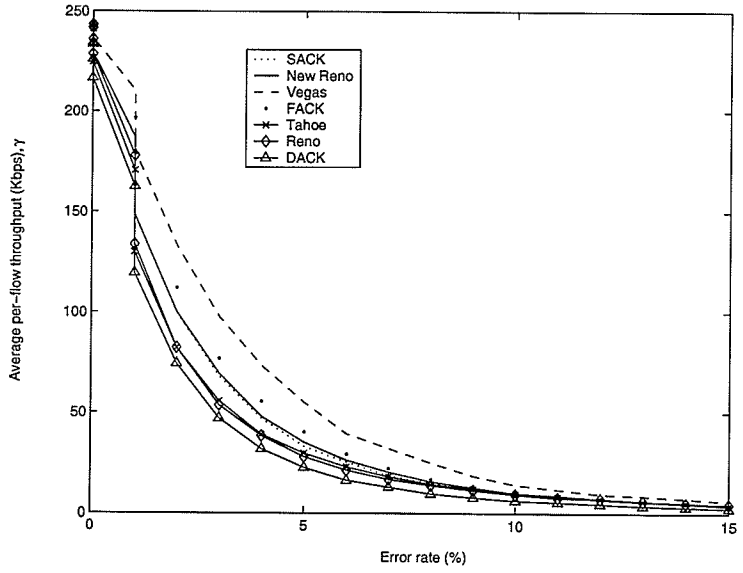

Figure 2.4. *Throughput performances of different TCP variants (scenario 2).*

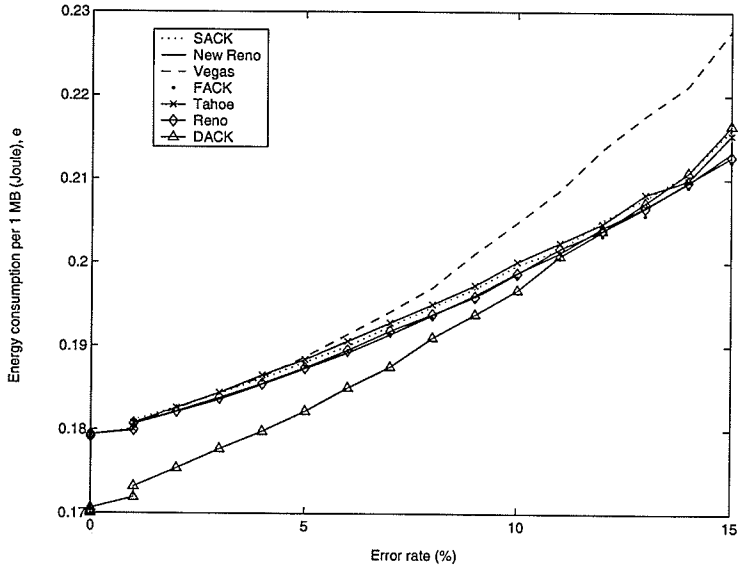

Figure 2.5. *Energy consumption for the different TCP variants (scenario 2).*

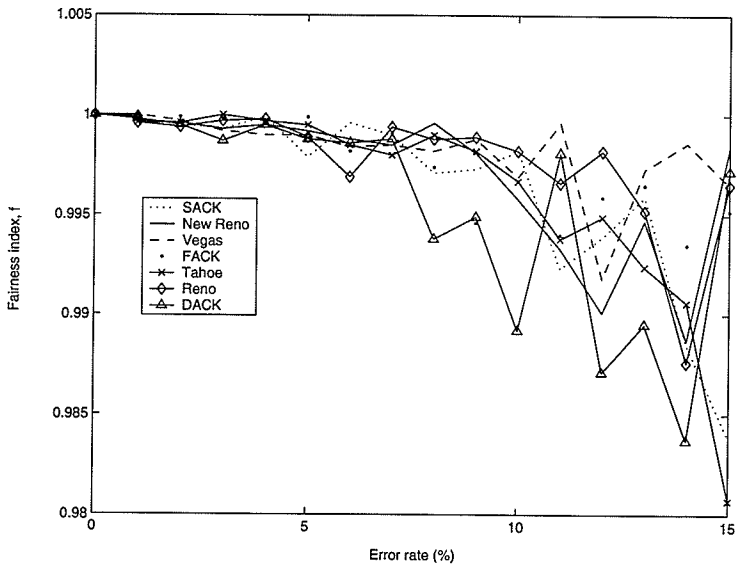

Figure 2.6. *Throughput fairness for the different TCP variants (scenario 2).*

TCP flows with that in the case of a single TCP flow occupying the entire bandwidth. The average throughput of this case (which is expected to be five times that for a flow in scenario 2) becomes close to the per-flow throughput in scenario 2 as the error rate increases beyond 1% (Fig. 2.7). The performance gain due to the high wireless link bandwidth is observable only for very small error rates (e.g., 0%-0.05%).

2.3.2.4 Single New-Reno Flow with Correlated Error(Scenario 4)

The wireless link bandwidth is assumed to be 2 *Mbps*. In this scenario, we choose to investigate only the performance of TCP New-Reno (which is regarded as the de facto standard in the present Internet) under correlated channel error. For a particular average packet error rate, the channel error correlation (as manifested in the lengths of the error bursts) varies as the mobile speed varies [24]. For example, for the assumed wireless channel bandwidth and maximum segment size, with 10% error rate, mobile speed of 1 *km/hr*, 3 *km/hr*, 5 *km/hr*, 10 *km/hr* and 100 *km/hr* correspond to burst-error size of 11.41, 3.85, 2.37, 1.42 and 1.11 segments, respectively. For 1% average error rate, the corresponding burst-error sizes are 3.41, 1.39, 1.14, 1.04 and 1.01 segments, respectively.

The average throughput decreases with increased channel error correlation (Fig. 2.8).

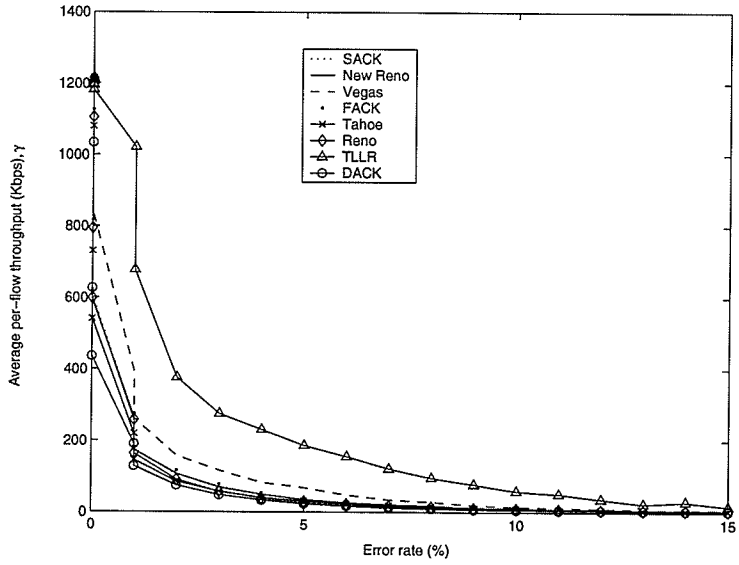

Figure 2.7. *Throughput performances of different TCP variants (scenario 3).*

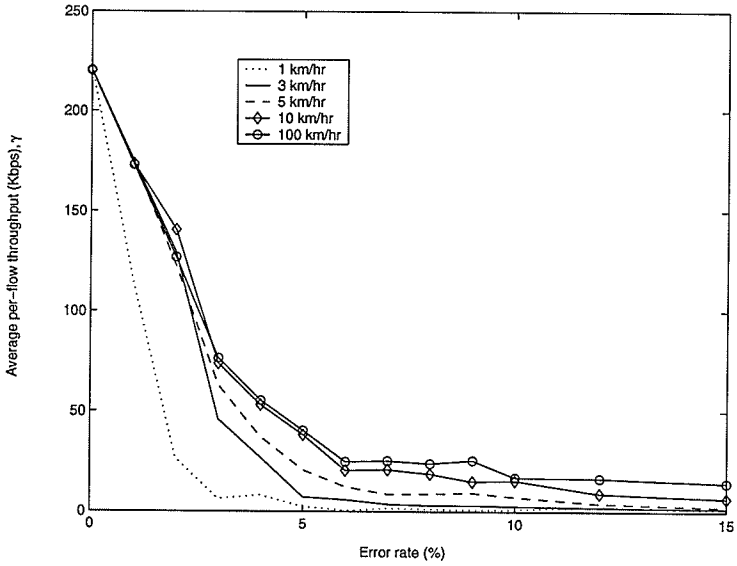

Figure 2.8. *Throughput performance of TCP New-Reno (scenario 4).*

Table 2.3. *Fast retransmit/fast recovery analysis of New-Reno under correlated error loss (for 1000* second *run)*

Mobile speed	TO	FR	MD	SND	FLT
	Error rate = 3%				
1 km/hr	83	11	11	8	14.82
3 km/hr	230	234	231	164	13.93
5 km/hr	259	328	319	172	13.64
10 km/hr	301	399	380	204	13.41
100 km/hr	300	404	384	191	13.45
	Error rate = 5%				
1 km/hr	105	1	1	1	13.00
3 km/hr	189	16	14	20	11.62
5 km/hr	321	75	71	137	11.45
10 km/hr	451	172	161	294	11.59
100 km/hr	479	187	175	251	11.50
	Error rate = 10%				
1 km/hr	76	0	0		
3 km/hr	140	0	0		
5 km/hr	298	0	0		
10 km/hr	532	0	0		
100 km/hr	507	0	0		

In the presence of highly correlated error, the acknowledgements from the mobile TCP sinks do not reach the TCP senders and the TCP error recovery is triggered primarily by timeouts. For example, with 10% error rate, it is observed that there is no *fast retransmit* at all (Table 2.3).

One interesting observation is that, although the number of timeouts decreases/increases with decreasing/increasing mobile speed, the average throughput decreases/increases. This is because, as user speed decreases, the channel error correlation increases, as a consequence of which the value of RTO backoff factor generally becomes high. With an increase in the value of RTO backoff factor, the end-to-end error recovery after a timeout becomes more sluggish, and consequently, the throughput deteriorates.

The amount of energy consumption is observed to increase with increasing channel error correlation.

2.3.2.5 Single Flow with Correlated Error(Scenario 5)

The wireless link bandwidth is assumed to be 2 $Mbps$. We assume mobile speed of 3 km/hr at which the burst-error size is 3.85 and 1.39 for error rate of 10% and 1%, respectively. The performance trends (Fig. 2.9)in this scenario are observed to be radically different from those in the random error case . Throughput performance of TCP Vegas reduces by about 80% (compared to the uniform error case) when the error rate is 2%. SACK TCP now performs better than TCP New-Reno over a range of error rates from 0% to 5%.

Since the system dynamics is now characterized by a large number of timeouts, high ratio of the number of multiple (three or more) packet drops to the number of single packet drops in a single window and loss of a large number of acknowledgements, performance of TCP Tahoe becomes comparable to the performance of SACK TCP. Also, we observe that the performance of TCP New-Reno with TCP-aware link level retransmission deteriorates considerably compared to that in the uniform error case.

TCP Reno is observed to perform the worst in this correlated error scenario. Although the number of fast retransmits is fewer than that in the uniform error case, most of the fast retransmits are due to multiple packet drops (three or more) in a single window. For example, with 1% error rate, there are 402 fast retransmits, among which 375 are due to multiple drops (Table 2.4). Again, 327 of the 375 packet drops

Table 2.4. *Fast retransmit/fast recovery analysis under correlated error (Error rate 1%, 3% and 10% and Run time 1000* second *)*

TCP variant	TO	FR	MD	SND	FLT
	Error rate = 1%				
Tahoe	8	539	538	0	14.98
Reno	431	402	375	226	13.01
New-Reno	7	544	541	1	14.99
SACK	4	553	547	1205	14.97
DACK	12	530	484	260	14.42
FACK	0	548	546	3284	14.93
TLLR	0	0	0		
Vegas	157	445			14.20
	Error rate = 3%				
Tahoe	195	214	210	59	14.42
Reno	346	95	93	25	12.80
New-Rreno	230	234	231	164	13.93
SACK	330	251	246	870	14.02
DACK	254	215	210	103	11.82
FACK	159	349	348	1941	14.51
TLLR	0	0	0	0	0
Vegas	381 175E	164			10.82
	Error rate = 10%				
Tahoe	140	0	0		
Reno	140	0	0		
New-Reno	140	0	0		
SACK	140	0	0		
DACK	145	0	0		
FACK	140	0	0		
TLLR	68	0	0		
Vegas	248 0E	0			

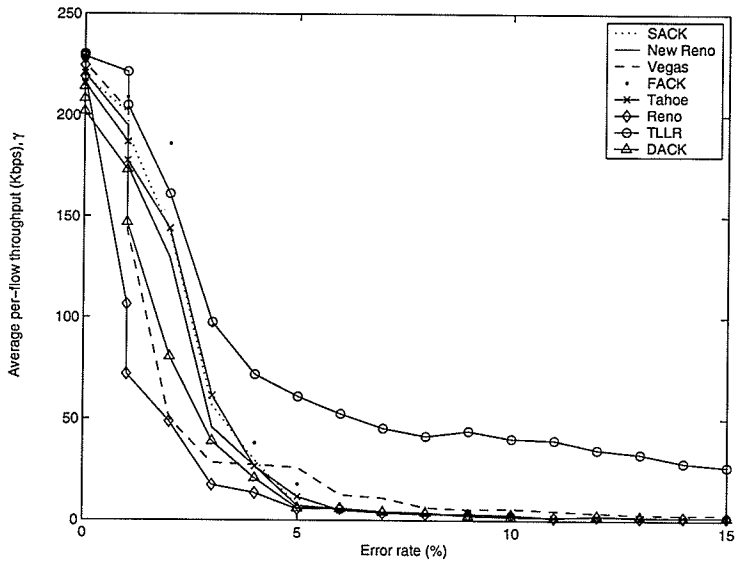

Figure 2.9. *Throughput performances for the different TCP variants (scenario 5).*

are due to three or more packet drops in a single window. Most of these multiple (more than three) drop cases are observed to be followed by timeouts. In fact, when three or more packets are dropped from a window of data, the Reno sender is forced to wait for a timeout most of the time.

TCP Tahoe is observed to perform significantly better than Reno under correlated error scenarios. Since after switching to *slow-start*, Tahoe forgets all outstanding data that were transmitted earlier and increases its congestion window upon receipt of each acknowledgement, it results in multiple transmission attempts for same packets (some of which have presumably been lost already), and consequently, the throughput improves for the multiple drop cases.

Since the number of packet drops per transmission window increases in the correlated error case, SACK TCP performs better than New-Reno. TCP New-Reno requires $n \times RTT$ to recover from n packet drops in a single window whereas SACK recovers much faster. This is mainly due to its better pipe estimation method and usage of a larger retransmission pool. For the same reason FACK TCP performs even better than SACK TCP.

Since a large number of acknowledgements are lost due to correlated channel error, sluggishness in transmitting acknowledgements in the case of TCP New-Reno with

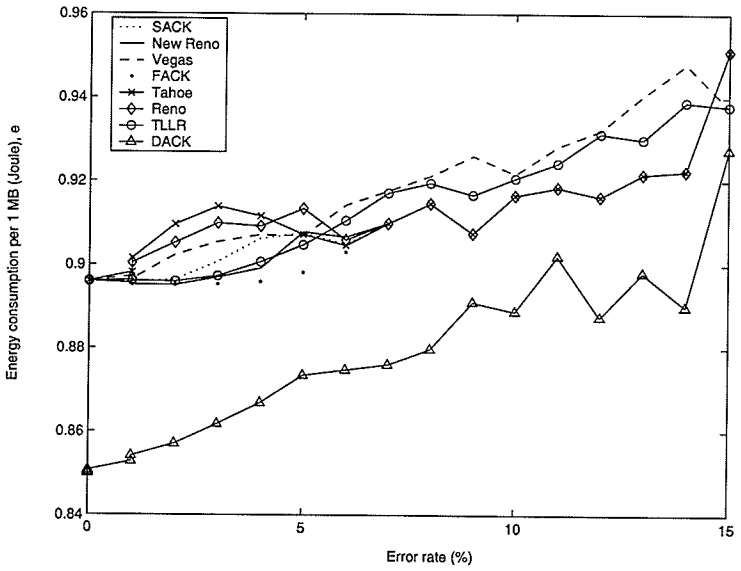

Figure 2.10. *Energy consumption for the different TCP variants (scenario 5).*

DACK option does not significantly impact the throughput performance. For the same reason, sharp response to packet losses in the case of TCP Vegas is not very conducive to improving TCP throughput performance in the correlated error case. Under different channel error rates and channel error correlation, the fixed values of α and β do not work well. Also, TCP Vegas experiences the highest number of timeouts.

For error rate greater than 7%, the energy consumption is observed to be more or less same for all the TCP variants (except for Vegas and New-Reno with TLLR) (Fig. 2.10). This is due to the fact that for a large error rate all of the TCP variants experience almost the same number of timeouts and there may be no *fast retransmit* at all.

2.4 Maximum Achievable TCP Throughput Under Window-Based End-to-End Transmission Control

In this section, we obtain the upper bound on the TCP througput performance under window-based end-to-end transmission control (i.e., throughput of *ideal end-to-end TCP* (IE^2-TCP)) and compare it against the throughput performances of TCP New-Reno, SACK TCP and FACK TCP. For any window-based transmission control, the timeout mechanism is a must. Note that, all of the basic TCP variants use similar timeout mechanism along with exponential RTO backoff. Also, they use the *fast retransmit* mechanism (i.e., immediate retransmission of lost segment detected by *tcp_rexmit_thresh* number of duplicate ACKs). However, the basic end-to-end TCP variants mainly differ in the implementation of the *fast recovery* mechanism. Note that, for the maximum possible throughput performance in a wired-cum-wireless scenario, IE^2-TCP should be aggressive enough in retransmission and in measuring the outstanding data segments. The upper bound on the throughput performance under wireless loss (achievable by the IE^2-TCP) can be obtained using the simulation model described before based on the following assumptions:

- IE^2-TCP uses the timeout mechanism along with exponential RTO backoff and it uses *fast retransmit* mechanism to retransmit a lost segment after reception of *tcp_rexmit_thresh* number of duplicate ACKs.
- IE^2-TCP retransmits a lost segment after receiving partial duplicate ACKs.
- Rather than using a go-back-N type of retransmission, IE^2-TCP uses selective acknowledgement (SACK)-based retransmission.
- IE^2-TCP uses the information in the SACK acknowledgement to estimate outstanding data.
- IE^2-TCP maintains the transmission window size equal to the receiver-advertized window size all the time.

Figs. 2.11-2.12 show some typical results on the long-term average throughput performance of IE^2-TCP when compared to those of SACK TCP, TCP New-Reno and FACK TCP under both random error and correlated error conditions. As is

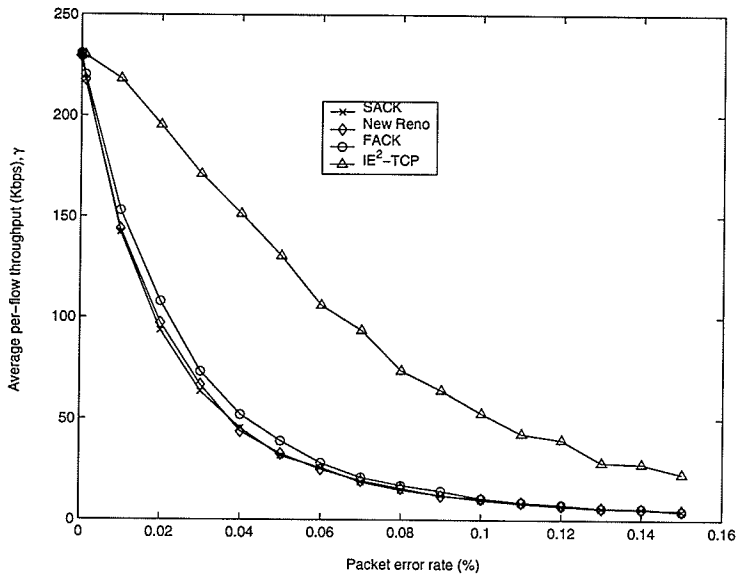

Figure 2.11. *Throughput performances for SACK TCP, TCP New-Reno, FACK TCP and IE^2-TCP under random packet error in the wireless link.*

evident from Fig. 2.11, under random packet error in the wireless link IE^2-TCP can achieve a throughput which is higher by as much as 200% (e.g., for packet error rate of 4%) than that due to SACK TCP and FACK TCP. Improvement in throughput performance can be even higher under correlated error scenarios (e.g., 250% improvement for packet error rate of 4% with average error burst length of 3.85) (Fig. 2.12). Therefore, it can be concluded that the basic TCP variants perform significantly poor compared to the *ideal TCP* (i.e., IE^2-TCP) in a wired-cum-wireless scenario. Since the throughput performance of IE^2-TCP characterizes the envelope of the maximum possible throughput under a TCP-like window-based end-to-end transmission control, the TCP modifications based on end-to-end approaches should use this as the benchmark performance.

2.5 Summary and Outlook

Throughput, fairness and energy performances of the different basic TCP variants have been investigated in a wide-area cellular wireless environment for both uniform and correlated wireless channel errors. The following provides a summary of the key

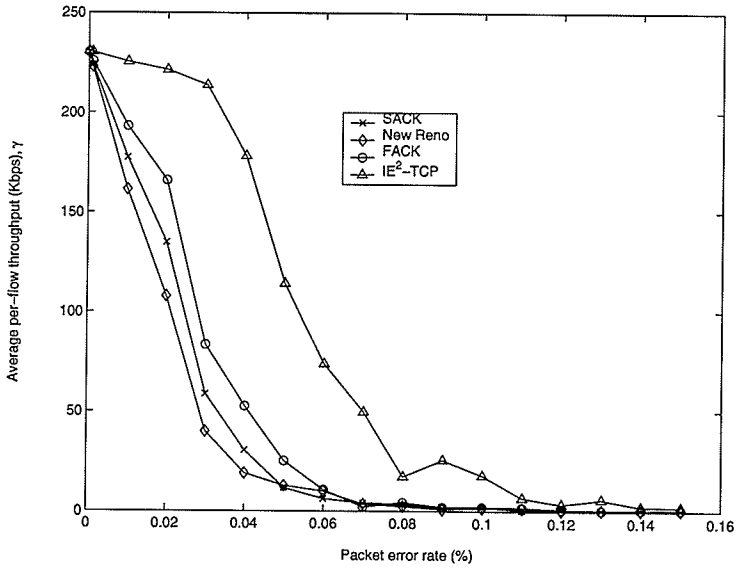

Figure 2.12. *Throughput performances for SACK TCP, TCP New-Reno, FACK TCP and IE^2-TCP under correlated packet error in the wireless link (for mobile velocity of 3 km/hr).*

observations:

- Implications of the fast retransmit and timeout events on the TCP performance largely depend on the wireless channel error characteristics (e.g., error rate and degree of channel error correlation). For example, in the case of TCP Vegas, sharp response of the sender due to its fine-grained expiry mechanism works well under uniform error case while the same strategy results in poor performance under correlated error scenarios. Therefore, although coarse timeout is undesirable under random losses, the ineffectiveness of the 'Vegas expiry' mechanism under correlated error cases suggests that very sharp timeouts may also not be desirable under such error conditions.
- A method which provides better estimation for the number of outstanding segments is always conducive to the TCP throughput performance under both random and correlated error scenarios. Due to this reason, FACK TCP is observed to be consistently better than SACK TCP and TCP New-Reno under both random and correlated error scenarios.

- Since the impact of timeout becomes more dominant, the throughput performances of all of the basic end-to-end TCP variants (which primarily differ in the *fast recovery* mechanism) suffer under correlated error scenarios. Again, end-to-end protocols with link level retransmissions (e.g., snoop protocol) also suffer serious performance degradation in case of correlated channel errors.
- Lower degree of wireless channel error correlation is more conducive to energy saving at the mobile TCP sinks.
- In a wide-area wireless scenario, TCP connections with high bandwidth-delay product may not get their fair share. Splitting a single TCP flow (with a high bandwidth-delay product) to multiple flows (say, x flows) increases the throughput approximately x times for a certain range of low error rates.
- TCP performance in a wired-cum-wireless environment can be improved significantly by using some transport level mechanism to differentiate between the congestion loss and wireless channel loss and then adjusting the window adaptation mechanism accordingly.
- The transmission window size at the TCP sender should be adapted differently depending on the degree of correlation in the wireless link errors. In addition, timer granularity may be adapted dynamically based on error correlation.
- The throughput performance of an end-to-end TCP designed for a wide-area cellular environment based on transport level modifications should be compared against the maximum achievable throughput envelope. This envelope, which can be characterized empirically based on simulation results, defines the maximum throughput achieveable under any TCP-like window-based end-to-end transmission control mechanism.

Chapter 3

TCP Enhancements for Wireless Networks

In this chapter, a brief overview of the different approaches for enhancing TCP performance (based on transport level modifications) along with a qualitative performance comparison are presented.

3.1 TCP Solutions for Wireless Networks

There has been a flurry of recent works ([26]-[53]) on improving TCP performance over wireless networks. A taxonomy of the different proposed mechanisms to improve TCP performance in wireless networks is shown in Fig. 3.1. Proposed modifications can be primarily classified according to the protocol level they operate on.

In addition to the basic TCP variants, link level and transport level approaches to improve TCP performance have been proposed in the literature. Link level approaches mainly try to make the radio link robust against wireless errors by using local retransmission mechanisms. A link layer scheme to improve TCP performance may or may not use transport layer knowledge. Next, tranport level modifications are mainly of two types – end-to-end and non-end-to-end. End-to-end solutions depend on the modification on either at the TCP source or at the TCP sink. Non-end-to-end modifications are mainly done at the base station/access point.

For wide-area cellular wireless networks, end-to-end TCP solutions based on transport-

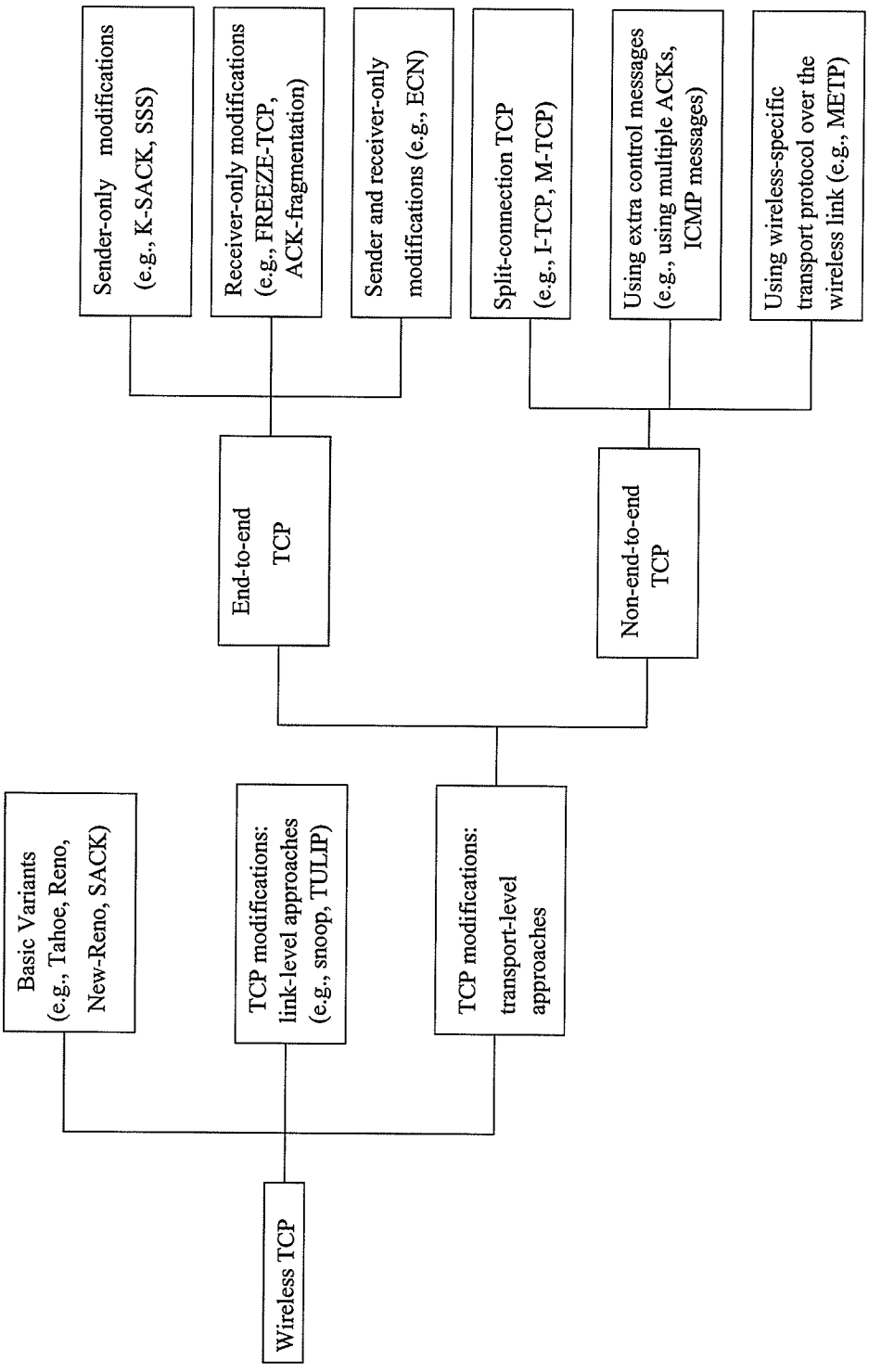

Figure 3.1. *Taxonomy of TCP solutions (based on transport-level approaches) for wireless networks.*

level modifications which use some intermediary (such as a base station) may not maintain true end-to-end semantics (e.g., I-TCP [42]) and the intermediary may become a bottleneck due to the overhead involved in processing traffic associated with each connection and handing over the 'state' of each connection to the new intermediary during handoff.

Link-level TCP solutions such as those which use TCP-aware smarts in the base station (e.g., snoop protocol [28]) have scalability problems and may even degrade the performance of TCP in a wide-area cellular environment when the latency over the wireless link dominates the round trip time [37]. In addition, such a solution requires the base station to maintain significant state and is often tuned to specific flavor of TCP (e.g., snoop protocol does not work well with TCP Vegas).

3.2 Transport Level Modifications to the Basic TCP Variants for Wireless Networks

3.2.1 Sender-Only Modifications

- *K-SACK* [30]: K-SACK is an efficient SACK-TCP variant of TCP New-Reno that requires modification at the TCP sender only. The novelty of this protocol is that it differentiates between wireless losses and congestion losses from anticipated loss pattern and behaves accordingly. During *fast recovery*, in case of wireless loss, it does not halve the congestion window instead it stalls the growth of the window. The *fast recovery* phase is partitioned into two phases: *halt growth* phase and *K-recovery* phase. During K-recovery phase, the sender apprehends congestion only if it anticipates K *lookahead-loss* within a loss window of size *lwnd*, where K and *lwnd* are appropriately chosen parameters for the protocol.

  The parameter *lwnd* can be chosen to be equal to the number of outstanding packets. The parameter *lookahead-loss* estimates the number of losses in the outstanding packets. For each such loss the sender has received either (i) at least *max_dupack* number of duplicate acknowledgements, or (ii) *max_dupsack* number of selective acknowledgements with higher sequence numbers. For

lookahead-loss below K (e.g., $K = 2$), the sender assumes loss due to channel error.

Upon entering into the *fast recovery* phase, if lookahead-loss is less than K, the sender enters into the *halt growth* phase, otherwise to the *K-recovery* phase and transition between these phases occurs depending on the value of the lookahead-loss.

During the *halt growth* phase, congestion window remains frozen and the sender remains in this phase until the lookahead-loss becomes at least K. When the sender enters into the K-recovery phase for the first time, *ssthresh* is set to $max(cwnd/2, 2 \times MSS)$ and *cwnd* is set to *ssthresh*. Upon re-entry, *ssthresh* remians unchanged, but congestion window is set to $max(cwnd/2, 1 \times MSS)$. This may cause the congestion window to change multiple times during the same *fast recovery* period. As a result, upon exiting *fast recovery*, depending on the value of *ssthresh* and *cwnd*, K-SACK TCP may find itself in the *slow start* phase instead of the *congestion avoidance* phase.

K-SACK performs a better *pipe* (i.e., number of packets in the link) estimation by measuring the *pipe* as the number of unacknowledged segments which are not marked *lost* or marked *lost* and retransmitted. This improved pipe estimation results in faster error recovery.

The main advantage of K-SACK is that it detects packet loss due to wireless channel error without explicit notification and it can be used seamlessly in wired or wired-cum-wireless environment without separate tuning. It was observed to perform very well under high *i.i.d.* error condition. Simulation results showed that, at 5% *i.i.d.* error rate the throughput was almost twice of that due to SACK-TCP over New Reno.

Although K-SACK performs well under random packet errors in the wireless channel, the performance of this scheme will suffer under correlated error conditions. This is due to the fact that, a moderate degree of correlated error will increase lookahead-loss to K and thus push K-SACK into K-recovery phase. Therefore, during *fast recovery* the sender may spend significant amount of time in *K-recovery* phase compared to that in the *halt growth* phase.

To combat the situation due to correlated packet loss in the wireless channel,

higher value of K can be chosen. However, it will increase the chance of congestion collapse in the wired Internet. Since in a real Internet scenario, only a small percentage of packet losses (e.g., 15% [31]) is recovered by *fast recovery*, enhancement only in the *fast recovery* mechanism as in K-SACK is not likely to improve the TCP performance significantly compared to the other TCP variants in a wide-area wireless Internet. In additon, under pathological scenarios K-SACK would suffer from oscillation between the *K-recovery* and the *halt growth* phase which would reduce *cwnd* to small values, and consequently, the performance may fall below that of SACK TCP.

- *TCP Westwood (TCPW)* [32]: TCPW uses a sender-side modification of TCP congestion window adaptation algorithm which is based on estimation of the bandwidth used by the TCP connection via monitoring the rate of returning ACKs. The bandwidth estimate (BWE) is used to set the congestion window and and the slow-start threshold after a congestion episode (i.e., after three duplicate acknowledgements or after a timeout), and thereby, TCPW achieves a "faster" error recovery. The basic congestion window dynamics during *slow-start* and *congestion avoidance* are unchanged. However, TCP Westwood does not use any method to determine the cause of packet loss, i.e., whether a packet loss is due to errors in the wireless link or due to congestion in a router. We will provide more details about TCP Westwood on next chapter 4.

  As an end-to-end solution to error and congestion control in mixed wired and wireless networks, TCPW was observed to provide significant throughput gain over TCP Reno and SACK TCP for both random and correlated error conditions in the wireless link. Also, the observed throughput fairness among TCPW flows across different round trip times was better compared to that for TCP Reno flows. This was due to the fact that, under TCPW long connections suffered less reduction in *cwnd* and *ssthresh*.

  As the round trip time increases, the time for the feedback information to reach the TCP sender also increases which reduces the effectiveness of TCPW. Also, TCPW performs poorly when random packet loss rate exceeds a small threshold. During long correlated error burst, TCP timeouts occur which affect TCPW performance severely.

- *TCP Santa Cruz* [33]: TCP Santa Cruz (SC) uses new congestion control and error recovery strategies which are designed to work with path asymmetries, networks with lossy links, limited bandwidth, variable delay and out-of-order packet delivery. Since congestion losses are preceded by an increase in the network bottleneck queue (which may not be true for random losses in the wireless link), TCP-SC monitors the queue developing over a bottleneck link and determines whether congestion is incipient in the network and responds by increasing or decreasing the congestion window accordingly. The goals are to perform timely and efficient early retransmission of any lost packet, eliminate unnecessary retransmissions for correctly received packets when multiple losses occur within a window of data, and provide RTT estimates during periods of congestion and retransmission.

  The congestion control algorithm in TCP-SC is based upon the measurement of *relative delay* that packets experience with respect to each other as packets are transmitted through the network. For each transmitted packet, TCP-SC sender maintains information about the transmission time of the packet, the arrival time of an ACK for that packet and arrival time of the data packet at the receiver (which is reported by the receiver in its ACK). Then for any two data packets i and j, the *relative forward delay* $D^F_{i,j}$ can be calculated as $D^F_{j,i} = R_{j,i} - S_{j,i}$, where $D^F_{j,i}$ represents the additional forward delay experienced by packet j with respect to packet i, $S_{j,i}$ is the time interval between the transmission of the packets, and $R_{j,i}$ is the interarrival time of data packets at the receiver. Based upon the values of $D^F_{j,i}$ over time, the change in the states of the queues (and specially the bottleneck queue) are determined. For example, if the sum of relative delays over an interval is 0, it implies that no additional congestion or queueing is present in the network at the end of the interval with respect to the beginning.

  If n is the operating point of the congestion control algorithm (i.e., n is the desired number of packets in the bottleneck queue which is assumed to be one packet more than the BDP of the network), it attempts to maintain the total number of packets queued at the bottleneck link from the beginning of the connection until t_i (N_{t_i}) to n, where $N_{t_i} = N_{t_{i-1}} + M_{W_{i-1}}$, with $M_{W_{i-1}}$

being the additional amount of queueing introduced over the previous window W_{i-1}, and $N_{t_1} = M_{W_0}$. Now $M_{W_{i-1}}$ can be calculated based on the relative delay measurements as follows: $M_{W_{i-1}} = \frac{\sum D_k^F}{\bar{t}_{pkt}}$ where $\bar{t}_{pkt}$ is the average packet service time and k is the number of packet pairs within window W_{i-1}. Note that, the average packet service time can be calculated based on the timestamps returned by the receiver.

TCP-SC adjusts the congestion window once for each interval equal to the amount of time it takes to transmit one window of data. Over this interval, $M_{W_{i-1}}$ is calculated and at the end of the interval it is added to $N_{t_{i-1}}$. If $N_{t_i} < n - \delta$ (δ is some fraction of a packet), the congestion window is increased linearly, while if $N_{t_i} > n + \delta$, the congestion window is decreased linearly during the next interval, otherwise the congestion window is maintained at its current size. In this way, since adjustment of the congestion window does not depend on the arrival of ACKs, the congestion control algorithm becomes robust to ACK loss.

TCP-SC sender receives precise information on each packet correctly received and it uses a tighter estimate of the RTT per packet. TCP-SC recovers losses quickly without necessarily waiting for three duplicate acknowledgements from the receiver. Let packet i initially transmitted at time t_i is lost and marked as a hole in the ACK window. Packet i is retransmitted as soon as an ACK arrives for any packet transmitted at time t_x such that $t_x > t_i$, and $t_{current} - t_i > RTT_e$, where $t_{current}$ is the current time and RTT_e is the estimated round trip time of the connection.

TCP-SC was observed to provide high throughput and low end-to-end delay and delay variance over networks with a simple bottleneck link, networks with congestion in the reverse path of the connection, and networks which exhibit path asymmetry. TCP-SC can be implemented as a TCP option by utilizing the extra 40 bytes available in the options field of the TCP header.

TCP-SC does not differentiate between packet loss due to congestion and packet loss due to wireless link error. In fact, effectiveness of the proposed relative delay-based approach under correlated wireless channel losses is unclear.

- *Selective Slow Start (SSS)* [34]: Under this scheme, TCP attempts to distinguish

between packet loss due to congestion and packet loss due to handoff based on the pattern of losses. For this, after a timeout the TCP sender counts the number of lost segments in the last S attempts including the current one and compares it with a pre-calculated *LIMIT*. If the number of lost segments is higher than *LIMIT*, the sender assumes that these losses are due to network congestion and initiates *slow start*, otherwise it continues to transmit at its current rate using the same timer values assuming that the losses are due to handoff. The value of the parameter *LIMIT* is set in a way that it can account for all segment losses during a handoff. For example, *LIMIT* can be set as $LIMIT = \frac{K \times C \times T_h}{Seg}$, where C is the senders transmission rate (in *bps*), T_h is the average handoff duration (in *seconds*), Seg is the average segment size and K is a safety factor.

Although the *SSS* scheme was shown to perform better than standard TCP in wireless ATM environment for high handoff rate (e.g., $0.2/second$), the main limitation of this scheme is that for rate adaptation at the TCP sender it does not take wireless link conditions into account. Also, using a high value of *LIMIT* may cause congestion collapse in the wired Internet.

3.2.2 Receiver-Only Modifications

- *Freeze-TCP* [35]: This is a proactive mechanism in which the receiver notifies the sender of any impending 'blackout' situation (e.g., due to wireless channel fading or handoff) by 'zero window advertisement (ZWA)' and prevents the sender from entering into congestion avoidance phase. Upon receiving ZWA, the sender enters into the 'zero window probes (ZWP)' mode and freezes corresponding timers. While in ZWP mode, the sender transmits zero window probes and the interval between successive probes grows exponentially until it reaches 1 minute, where it remains constant. When the 'blackout' period is over, the receiver sends TR-ACKs (Triplicate Reconnection ACKs) for the last data segment successfully received to enable *fast retransmit* at the TCP sender. Ideally, the 'warning period' prior to disconnection (i.e., how much in advance should the receiver start ZWA) should be long enough to ensure that exactly one ZWA gets across the sender. If the warning period is longer than this, the

sender will be forced into ZWP mode prematurely resulting in idle time prior to disconnection. If the warning period is too small, the receiver might not have enough time to send out a ZWA which will cause the sender's congestion window to drop.

One drawback of this approach is that, the receiver needs to predict the impending disconnections. For this, some cross-layer information exchanges may be necessary.

- *ACK-Fragmentation Protocol* [36]: If the mobile host, after sending three duplicate acknowledgements, receives a new retransmitted packet from the sender, it sends N equally spaced cumulative acknowledgements for this new packet within one round-trip time. If for this received packet the sequence number of the first octet is S and last octet is F, the ith cumulative acknowledgement will acknowledge upto octet $S + i(F - S)/N$. The value of N can be computed from the random packet loss probability in the wireless link, the link capacity, round-trip time and buffer size at the bottleneck link [36].

  The effectiveness of this protocol comes from the fact that increased number of acknowledgements within same round-trip time expedites the resumption of normal transmission window at the TCP sender. Moreover, it increases the chance that the sender receives at least one acknowledgement.

3.2.3 Modifications at the Sender and the Receiver

- *Wireless TCP (WTCP)* [37]: WTCP is an end-to-end reliable transport layer protocol for wired-cum-wireless networks which uses *rate-based* transmission control instead of window-based transmission control and the rate adaptation computations are performed by the receiver. Since the receiver performs the rate computations, the effects of delays variations and losses in the ACK path are eliminated. The goal of WTCP is to decrease the transmission rate aggressively in case of congestion (so that the congestion alleviates quickly) and decrease less aggressively in the case of incipient congestion to improve efficiency.

  To distinguish between congestion-related and non-congestion-related packet loss, the WTCP receiver, using a history of packet losses, computes the mean and mean deviation in the number of packet losses when the network is pre-

dicted to be uncongested. If the loss is predicted to be due to congestion, the transmission rate is decreased aggressively by 50%. For non-congestion related packet loss WTCP uses the ratio of the average interpacket delay observed at the receiver to the interpacket delay at the sender (rather than using packet loss and retransmission timeouts) to control transmission rate. The WTCP receiver maintains two state variables *lavg_ratio* and *savg_ratio* for the long-term and short-term running averages of the ratio of the observed sending rate at the receiver to the actual sending rate at the sender. At any time, the receiver can be in one of the three states–*increase* (when $lavg\_ratio > \alpha_+$ and $savg\_ratio > \beta_+$), *decrease* (when $lavg\_ratio > \alpha_-$ and $savg\_ratio > \beta_-$), *maintain.*

WTCP uses SACK and no retransmission timer for loss recovery. The sender tunes the desired rate for ACK transmission by the receiver so that it receives at least one ACK in a threshold period of time and can react to the new transmission rate.

In summary, WTCP is significantly different from TCP and tuning of the different parameters would be required to optimize the performance of WTCP.

- *Explicit Congestion Notification (ECN) TCP* [38]: This proposal is based on the TCP with explicit congestion notification [39] mechanism where the routers should inform the TCP sender of incipient congestion so that the TCP senders can lower the transmission rate. For example, if RED (Random Early Detection) mechanism is used, a router signals incipient congestion to TCP by setting the Congestion Experienced (CE) bit in the IP header of the data packet when the average queue size lies between two thresholds min_{th} and max_{th}. In response, the receiver sets the 'ECN-Echo (ECE)' bit in the ACKs and when the sender receives an ACK packet with ECE bit set, it invokes the congestion control mechanism. In case of packet loss in the wireless channel, the receiver sends ACK packets without CE bit set and on receiving three duplicate ACKs without CE bit set the sender deduces that the packet loss is due to errors in the wireless channel. In this case, the sender does not invoke the *congestion avoidance* algorithm.

  In case of long outage in the wireless channel, this scheme may not work well. Again, this can be used only for ECN-capable TCP.

- *Eifel Algorithm* [40]: A large number of timeouts and fast retransmissions that occur at a TCP sender have been observed to be *spurious*. That is, the ACKs received at the sender after a timeout or fast retransmit are actually the ACKs for the original packets and not for the retransmitted packets. Spurious fast retransmits result when the packet reordering in the Internet results in a reordering length more than the duplicate ACK threshold. Due to spurious timeouts and spurious retransmissions, the TCP sender unnecessarily reduces the transmission rate and enters into a go-back-N type of retransmission mode. When the sender enters into the retransmission mode, it causes the receiver to send duplicate ACKs which may result in *fast retransmit* after a timeout resulting in further reduction in transmission rate. The *Eifel algorithm* was proposed to eliminate the problems caused by *spurious timeouts* and *spurious fast retransmits* in TCP.

  The Eifel algorithm uses the TCP timestamp option. Each packet transmitted by the sender is timestamped and the values of *ssthresh* and *cwnd* are also stored. The receiver returns the timestamp value of the packet in the corresponding ACK. After a timeout or a fast retransmit, the values of *ssthresh* and *cwnd* are updated according to the usual algorithms and the packet is retransmitted. After receiving the ACK, if the sender finds that it is for the original packet, it restores the values of *ssthresh* and *cwnd*.

  This algorithm does not take packet loss due to wireless channel error into account, and therefore, does not solve the problems of TCP in a wired-cum-wireless network.

3.2.4 Modifications at Base Station: Non-End-to-End Transport Level Approaches

- *Indirect-TCP (I-TCP)* [42]: I-TCP splits a transport layer connection between the MH and the FH into two separate connections – one over the wireless link between the MH and its mobile support router (MSR) at the base station and the other between the MSR and the FH over the fixed network. After receiving the data packets destined to the MH, the MSR sends ACKs to the FH and forwards the packets to the MH using a separate transport protocol designed

for better performance over wireless links. Such a transport protocol for the wireless link can support notification of events such as disconnections to the link-aware and mobility-aware applications. In this case, the mobile hosts can run simple wireless access protocol to communicate with the MSR and the MSR manages the communication overheads for communicating with the FHs. For example, when an MH wants to communicate with a FH, it sends a request to its current MSR, and the MSR then establishes a TCP connection with the FH. Whenever an MH moves to another cell, the entire connection needs to be moved to the new MSR which should be completely transparent to the fixed network.

Performance of I-TCP was observed to be better than regular TCP in wide-area wireless networks. However, end-to-end TCP semantics are not maintained in this protocol and applications running on the mobile host have to be relinked with the I-TCP library. Also, I-TCP is not suitable for cases where the wireless link is not the last part of a connection path, because in such a case a particular connection may need to be split several times resulting in performance degradation.

- *Using ICMP Control Messages* [44]: A scheme for improving TCP performance in wireless networks using ICMP control messages was proposed in [44]. In this scheme, the BS transmits an ICMP-DEFER message to the FH when it fails in its first attempt to transmit the packet to the MH over the wireless link. On receiving the ICMP-DEFER message for a particular packet, the sender resets the corresponding timer. When all local retransmissions fail, the BS notifies the sender using ICMP-RETRANSMIT message so that the sender can retransmit the packet.
- *Using Multiple Acknowledgements* [45]: The scheme proposed in [45] maintains end-to-end semantics and uses two types of ACKs, namely, *partial ACK* (ACK_p) and *complete ACK* (ACK_c) to distinguish losses due to congestion from losses due to wireless link error. The sender uses the regular TCP mechanism with *slow-start*, *congestion avoidance*, *fast retransmit* and *fast recovery*. The complete acknowledgement ACK_c is sent by the receiver and the partial acknowledgement ACK_p is sent by the BS. ACK_p with sequence number N_a informs the

sender that packet(s) with sequence numbers up to $N_a - 1$ have been received by the BS, and the BS is facing problems in forwarding the packets with sequence numbers from ACK_c up to $N_a - 1$ (i.e., it has not received the ACK from the wireless host even after waiting for the maximum possible delay time between the transmission of a packet from BS and reception of the ACK from the mobile host). Then the sender marks the corresponding packets and updates RTO to give the BS more time to perform retransmission. If timeout occurs, and the packets are still marked, the sender will backoff the timer only without invoking any congestion control method. On receipt of ACK_c, the sender works similar to a normal TCP sender.

When the BS receives an in-sequence packet, or an out of sequence packet which is not in its buffer, the packet is buffered and transmitted to the mobile host. The BS starts a timer with the timeout value equal to the maximum time which can elapse between the reception of a packet at the BS and its acknowledgement from the mobile host. If the BS receives an out of sequence packet which is present in the buffer, it sends an ACK_p to the sender. If the local timer at the BS expires before receiving an acknowledgement from the mobile host, the BS retransmits the packets and sends an ACK_p to the sender.

- *Mobile-End Transport Protocol (METP)* [46]: METP hides the wireless link from the rest of the Internet in a wired-cum-wireless network by terminating the TCP connection at the BS on behalf of mobile host. TCP/IP between the BS and a mobile host is replaced by a low overhead protocol. All TCP connections are handled by METP at the BS which negotiates with another host in the Internet to open or close a TCP connection, and keeps the connection state and sending and receiving buffers. For data transfer from a mobile host to an Internet host through a TCP connection, the mobile host sends the data to the BS, and METP sends them out as TCP segments to the destination. When a TCP/IP packet destined to the mobile host arrives at the BS, METP sends an ACK and puts it in the receiving buffer from where a separate process transmits it to the mobile host. In case of temporary link failure, since METP at the BS continues to receive data from the fixed host, it sends out ACK with smaller advertised window. The advertized window is retained to its original level when

the wireless link becomes good again.

METP uses link layer retransmissions and acknowledgements for reliable data delivery over the wireless link. If no acknowledgement is received immediately after a data frame transmission, the frame is retransmitted after a random backoff interval. To avoid buffer overflow, flow control is achieved by the METP at the receiver through periodically sending out a feedback packet to inform the sender of the available buffer space.

METP uses a header size of only 12 bytes for packets exchanged between the mobile host and the BS. The header size can be further reduced by using the idea of header compression. In case of handoff the old BS opens a separate TCP connection with the new BS and sends all data and state information.

The merits of this protocol are that by using small header it incurs less overhead for wireless transmission and it avoids activation of the TCP congestion control mechanisms in case of packet loss in the wireless channel. METP was observed to provide better throughput performance than split-connection TCP and split-connection TCP with selective ACK under varying handoff interval and BS buffer size. However, METP does not maintain the strict end-to-end semantics and suffers from the scalability and reliability problems (due to failure of the BS).

- *WTCP* [47]: WTCP uses a modified flow and error control protocol between the BS and the mobile host. After receiving a TCP/IP packet from a fixed host, WTCP at the BS buffers it (if this is the next packet expected from the fixed host or if the packet has a larger sequence number than expected) along with its arrival time. WTCP then transmits the buffered packet to the mobile host and when a packet is transmitted to the mobile host, the BS schedules a new timeout if there is no other timeout pending. Upon receiving the ACK from the mobile host, WTCP frees the corresponding buffer and sends the ACK to the fixed host. WTCP performs local error recovery based on duplicate acknowledgements or timeout. In case of timeout, WTCP reduces the transmission window size to one assuming subsequent bad channel condition. As soon as the BS receives an ACK, the transmission window size is set again to the receiver advertized window size. Also, upon reception of a duplicate ACK, WTCP opens

the transmission window in full.

Another feature of WTCP is, it attempts to hide the time spent by the BS for local error recovery by adding that time to the timestamp value of the corresponding segment so that the TCP's round trip time estimation at the source is not affected. In this way, the TCP source's ability to effectively detect congestion in the wired network is not impacted.

Due to its aggressiveness in window adaptation, WTCP was observed to provide better/equal throughput performance compared to I-TCP and *snoop TCP*. However, a more optimized flow and congestion control scheme along with header compression (as in METP) can be used under similar scenario to provide even better performance.

A qualitative comparison among the different end-to-end TCP modifications is provided in Table 3.1.

3.3 Summary and Outlook

A brief overview of the different proposed TCP mechanisms (based on transport level modifications) along with their advantages and disadvantages have been presented. Among all these mechanisms, our focus is on the approaches that estimate bandwidth and use that estimate to control transmission rate. TCP Westwood falls in this category. We have a detailed analysis of TCP Westwood on next chapter 4. From all the TCP apporaches studied in this chapter, we can conclude that future research in end-to-end TCP design for wired-cum-wireless networks should address

- how to distinguish among packet losses due to wireless channel error and congestion error (e.g., by using some estimation/filtering mechanisms) and comparison among the different (estimation/filtering) mechanisms
- how to exploit the above information for adjusting the transmission rate at the sender in the case of packet loss to design end-to-end TCP based on modifications at the sender only
- fairness, energy-efficiency and TCP-friendliness of an end-to-end TCP customized for wired-cum-wireless networks.

Table 3.1. *Comparison among the transport-level TCP modifications.*

TCP Variant	Maintains end-to-end semantics	Can handle end-to-end encryption	Requires intermediaries' support	Can handle random packet loss	Can handle burst errors efficiently
K-SACK	Yes	Yes	No	Yes	No
Westwood	Yes	Yes	No	Yes	No
Santa Cruz	Yes	Yes	No	Yes	-
SSS	Yes	Yes	No	Yes	No
Freeze-TCP	Yes	Yes	No	Yes	Yes
ACK-fragmentation	Yes	Yes	No	Yes	No
WTCP (rate-based)	Yes	Yes	No	Yes	Yes
ECN	Yes	No	Yes	Yes	No
Eifel	Yes	Yes	No	No	No
I-TCP	No	No	Yes	Yes	No
Using ICMP	No	No	Yes	Yes	Yes
Multiple-ACK	Yes	No	Yes	Yes	Yes
METP	No	No	Yes	Yes	Yes
WTCP (window-based)	Yes	No	Yes	Yes	Yes

Chap 4

Analysi of TCP Westwood

In this chapter we investigate the bandwidth estimation mechanism of *TCP Westwood* [32] and its consequent impact on the throughput and friendliness performances of *TCP Westwood*. We also investigate the interaction of *TCP Westwood* with a RED (Random Early Detection) [56] queue in the network router.

4.1 Introduction

An important way of transport-level TCP modification is to exploit an end-to-end bandwidth estimation. In such a case, the fair share of the bottleneck bandwidth for each flow is estimated at the TCP sender and the transmission rate is adjusted accordingly(e.g., in *TCP Westwood*). This approach to provide enhanced transport over heterogeneous networks requires only slight moifications at the sender side while it maintains the end-to-end semantics and requires no support from the lower layers and/or any network component.

TCP Westwood is a recently proposed end-to-end modification of TCP in which the transmission rate for a flow at the sender is controlled based on the estimated available bandwidth for that flow in the end-to-end path. Here, we analyze the behavior of the bandwidth estimation mechanism in *TCP Westwood* by using simulations based on *ns-2*. Simulation results reveal that the proposed technique results in erroneous bandwidth estimation under different scenarios with multiple concurrent *TCP Westwood* flows.

4.2 TCP Westwood

TCP Westwood sender monitors the rate at which acknowledgements arrive at the sender and from this it estimates the bandwidth available for this connection. Whenever the sender perceives a packet loss (i.e., when a timeout occurs or three duplicate acknowledgements are received), the sender uses the bandwidth estimate to properly set the congestion window (*cwnd*) and the slow-start threshold (*ssthresh*). By adjusting the *cwnd* and the *ssthresh* values based on the estimated available bandwidth, *TCP Westwood* tries to avoid overly conservative reductions in *cwnd* and *ssthresh* (and hence transmission rate) and as well as ensure faster recovery from packet loss.

TCP Westwood uses a discrete time low pass filter with variable gain to estimate the available bandwidth in the end-to-end path as follows [32]:

$$\hat{b}_k = \alpha_k \hat{b}_{k-1} + (1 - \alpha_k)\left(\frac{b_k + b_{k-1}}{2}\right) \tag{4.1}$$

where $\hat{b}_k$ is the filtered estimate of the available bandwidth at time $t = t_k$, $\alpha_k = (2\tau - \Delta_k)/(2\tau + \Delta_k)$, $\Delta_k = t_k - t_{k-1}$ and $1/\tau$ is the cutoff frequency of the filter. Since an acknowledgement ACK_k received at the TCP sender at time t_k implies that upto byte $BYTENO(ACK_k)$ have been received by the TCP receiver, a sample of the estimated bandwidth is:

$$b_k = \frac{(BYTENO(ACK_k) - BYTENO(ACK_{k-1}))}{t_k - t_{k-1}} \tag{4.2}$$

where t_{k-1} is the time when the previous acknowledgement (i.e., ACK_{k-1}) was received.

When the inter-arrival time Δ_k increases, the filter gain α_k decreases, and consequently, the last estimate $\hat{b}_{k-1}$ receives less significance and the past two recent samples have higher significance. The bandwidth estimate (BE) $\hat{b}_k$ is used to set *ssthresh* and *cwnd* during timeout or fast recovery as follows:

```
/* algorithm for course timeout expiration */
if (course timeout expires)
  ssthresh = (BE * RTTmin)/segment_size;
  if (ssthresh <2 ) {
    ssthesh = 2;
```

```
  }
  cwnd = 1;
endif

/* algorithm for n dupacks */
if (n dupacks are received)
  ssthresh = (BE * RTTmin)/segment_size;
  if (cwnd > ssthresh) {
    cwnd = ssthesh
  }
endif
```

Since the estimated bandwidth is considered to be the currently available bandwidth, *TCP Westwood* sets both *ssthresh* and *cwnd* equal to $BE \times RTTmin$ and then the standard New-Reno fast retransmit and fast recovery follow. During timeout, slow-start threshold is set equal to BE and the cwnd is set equal to one. As timeout is an indication of severe congestion and/or high wireless loss, it is reasonable to set congestion window to one. At the same time setting of *ssthresh* equal to BE ensures speedy recovery.

4.3 Accuracy of the Bandwidth Estimation Mechanism in TCP Westwood

Fig. 4.1 shows the network topology used to analyze the behavior of the bandwidth estimation procedure. In Fig. 4.1, the bottleneck link is used by n TCP flows, S_i and R_i represent the ith TCP source and sink, respectively. For each TCP flow, the acknowledgements and the data packets pass through the same path. The bottleneck link capacity is 20 *Mbps* and all the packets are assumed to be of size 1000 bytes. Each TCP flow is assumed to have identical propagation delay. Ten TCP flows (flow1 - flow10) are started initially and another ten flows (flow11 - flow20) are started at 30 seconds. All flows continue to the end of simulation (60 seconds). Each flow has infinite backlogged traffic.

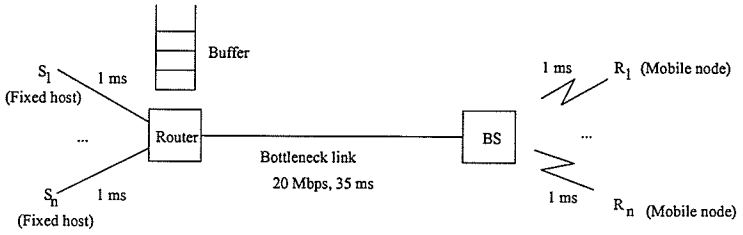

Figure 4.1. *Simulation scenario.*

4.3.1 Scenario 1: Bandwidth Estimation Under No Loss

In this scenario, we assume no loss (either due to congestion or wireless error). To avoid congestion loss, sufficient buffer is used in the bottleneck link.

Figs. 4.2 and 4.3 show typical variations in bandwidth estimate (BE) with time for flow1 and flow15, respectively. Similar variations in bandwidth estimation are observed for other flows (in flow1 - flow10 and flow11 - flow20). As is evident from Fig. 4.2, after initial rise up, the available bandwidth is estimated to be about 11 *Mbps* during the first 30 seconds although actual fare share in this case is only 2 *Mbps* (bottleneck capacity of 20 *Mbps* shared among 10 flows). For second half of the simulation time (where number of flows is doubled), the available bandwidth is estimated to be about 6.5 *Mbps* for all flows (flow1 - flow20) although the fare share during that period is 1 *Mbps*. Therefore, the estimation error is about 450% during the first 30 seconds and about 550% during the last 30 seconds. Note that the estimated bandwidth (and hence the estimation error) is quite stable.

Typical variations in the aggregate of the bandwidth estimate (ABE) for all flows with time is shown in Fig. 4.4. The aggregate bandwidth estimate is observed to be about 120 *Mbps* and 140 *Mbps* during the first half and the second half of the simulation period, respectively although the bottleneck link bandwidth is only 20 *Mbps* for all periods. Therefore, the bandwidth estimation method used in *TCP Westwood* is erroneous. Error in bandwidth estimation increases as the number of flows increases.

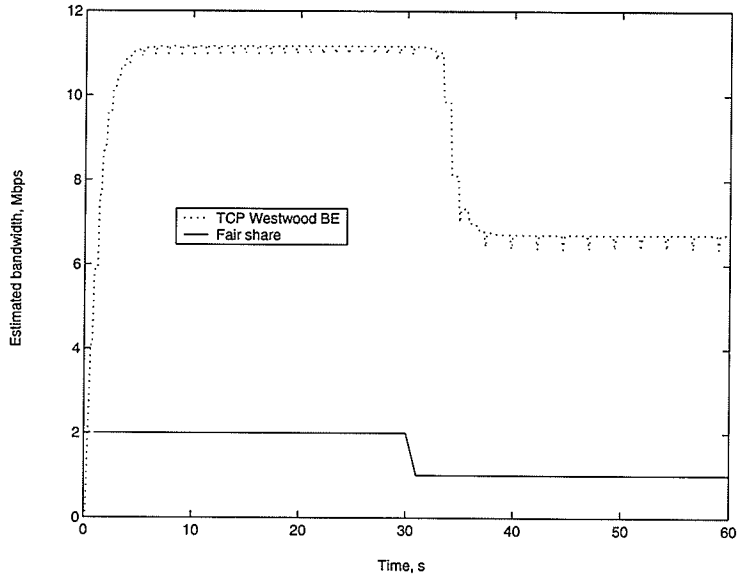

Figure 4.2. *Variation in BE for flow1 (scenario 1).*

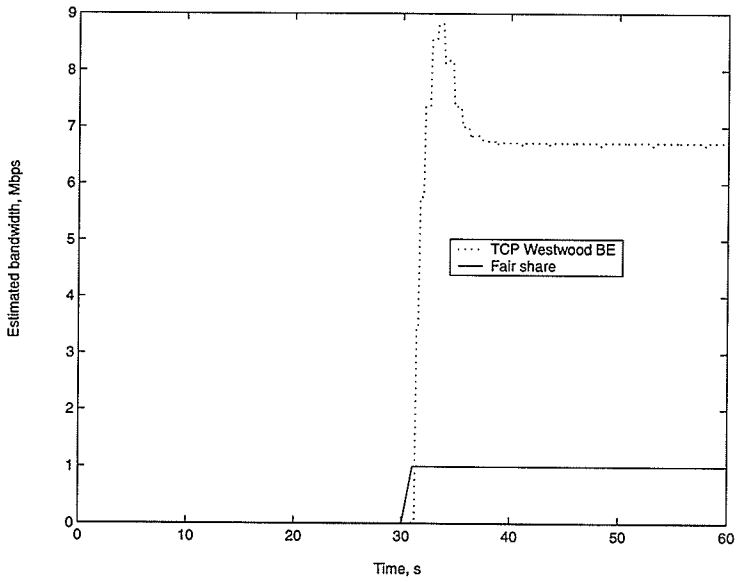

Figure 4.3. *Variation in BE for flow15 (scenario 1).*

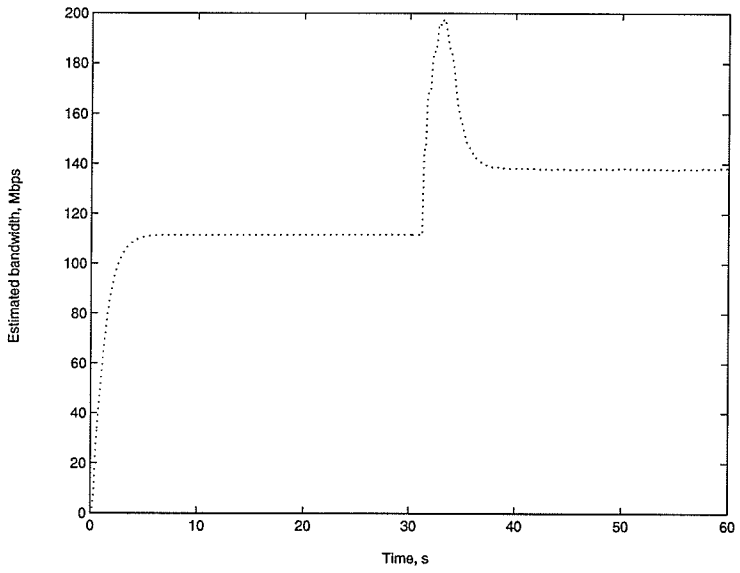

Figure 4.4. *Variation in aggregate of estimated bandwidth for all flows (scenario 1).*

4.3.2 Scenario 2: Bandwidth Estimation Under Wireless Loss

In this scenario, the TCP sinks ($R_1 - R_{20}$) are assumed to be connected to the bottleneck router through wireless a link of bandwidth 22 *Mbps* (e.g., 802.11b WaveLAN). Packet loss is assumed to be only due to errors in the wireless link (i.e., no congestion loss) and packet error rate in the wireless link is assumed to be 1%. Typical variations in the bandwidth estimate for each flow and the aggregate bandwidth for all flows are shown in Figs. 4.5, 4.6, repectively. Unlike scenario 1 the variations in the instantaneous bandwidth estimates for all the flows may not be same in this case. The aggregate bandwidth estimate for all flows is in the presence of wireless loss still very high (i.e., 100 *Mbps* during the first 30 seconds and 150 *Mbps* during the last 30 seconds).

4.3.3 Scenario 3: Bandwidth Estimation Under Congestion Loss

In this scenario, packet loss is assumed to be only due to congestion in the bottleneck router. The amount of buffer at the bottleneck router is assumed to be equal to the bandwidth-delay product of the network (185 packets). Similar to scenario 2,

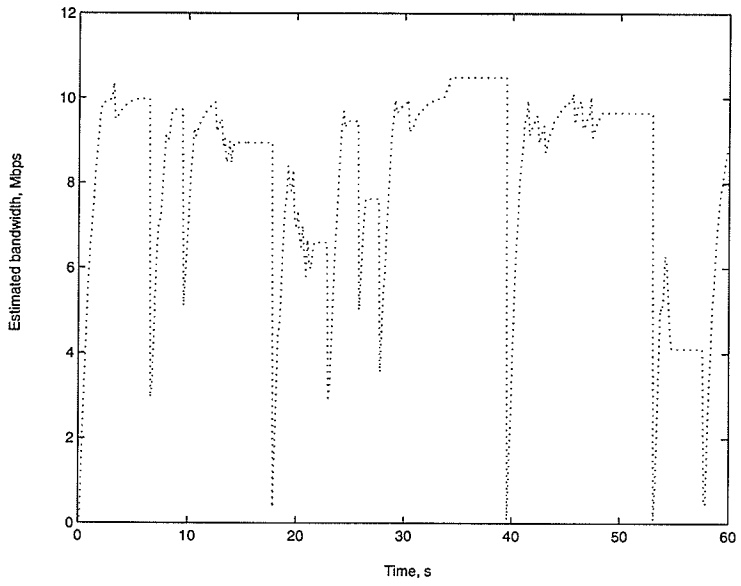

Figure 4.5. *Variation in BE for flow1 (scenario 2).*

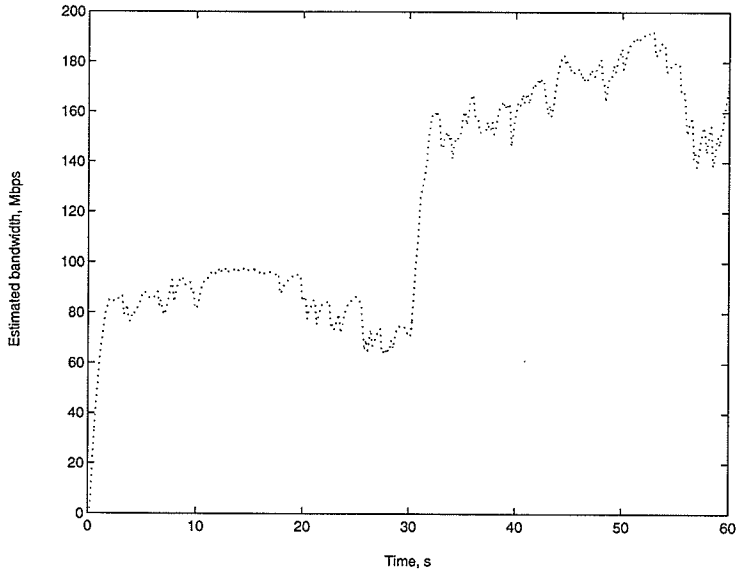

Figure 4.6. *Variations in the aggregate of estimated bandwidth for all flows (scenario 2)*

estimated bandwidth for each flow has been observed to be high compared to the corresponding fair share (Fig. 4.7). The aggregate of the estimated bandwidth for all flows is about four times and seven times the ideal estimated bandwidth during the first 30 seconds and the last 30 seconds of the simulation run, respectively. This erroneous bandwidth estimation may lead to congestion collapse.

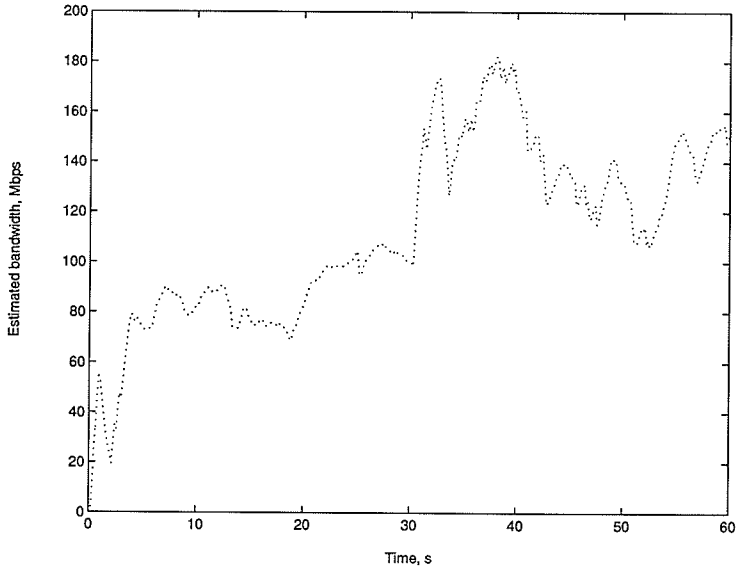

Figure 4.7. *Variation in aggregate of the estimated bandwidth for all flows (scenario 3).*

4.3.4 Scenario 4: Bandwidth Estimation Under Both Congestion Loss and Wireless Loss

In this scenario, the bottleneck buffer is set to 185 packets. Packet loss rate in the wireless link is assumed to be 1%. Fig. 4.8 depicts bandwidth estimation of flow1 as representative of first 10 flows. Although there are occasional degradations in the estimated bandwidth, it remains near 10 *Mbps* most of the time. Variations in the aggregate of the estimated bandwidth are similar to those in scenario 3.

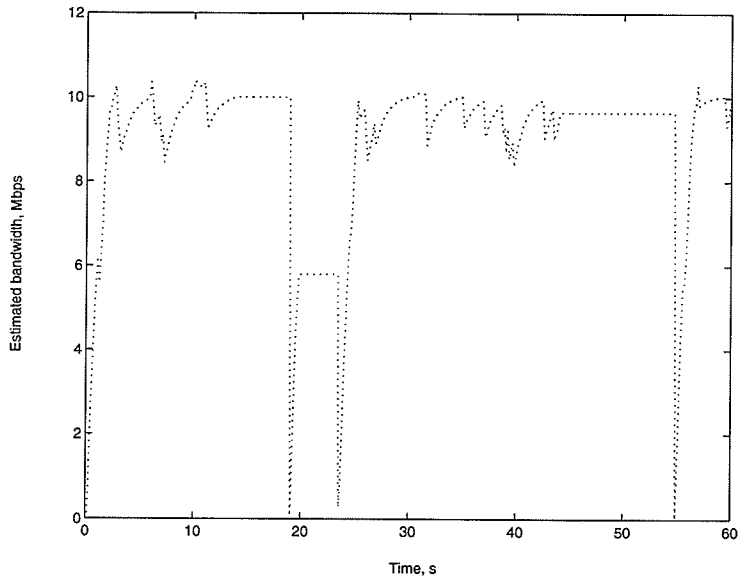

Figure 4.8. *Variation in BE for flow1 (scenario 4).*

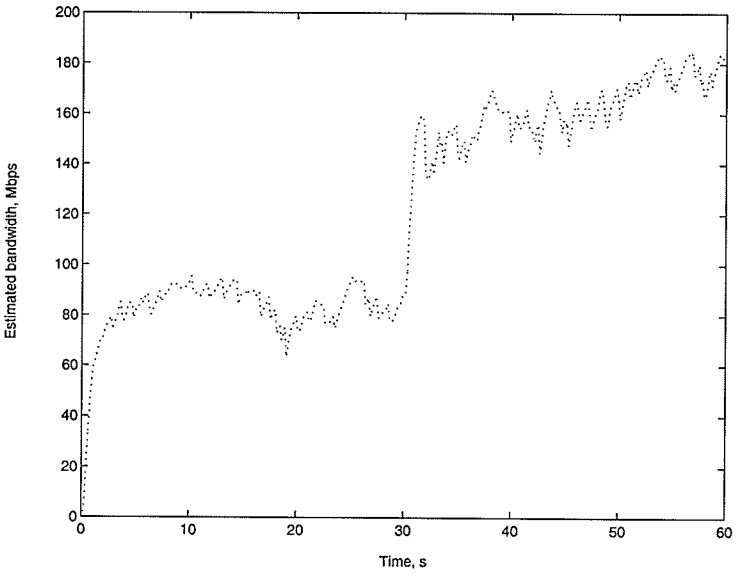

Figure 4.9. *Variation in aggregate of the estimated bandwidth for all flows (scenario 4).*

4.3.5 Summary of Observations

The bandwidth estimation mechanism in *TCP Westwood* always over estimates the available bandwidth for each flow (compared to its fair share). In the presence of errors (congestion and/or wireless) the inaccuracy in the bandwidth estimation increases with increasing number of flows. This erroneous bandwidth estimation causes each TCP Westood flow to transmit at a rate which is much higher compared to its fair share. This may cause TCP Weswood to become unfriendly to other TCP variants and may also lead to congestion collapse (and hence drastic reduction in througput).

4.4 Impact of Aggressive Bandwidth Estimation

To demonstrate the impact of erroneous bandwidth estimation of *TCP Westwood* on its performance, we observe the variations in congestion window for a TCP Westwood sender, queue length in the bottleneck buffer and throughput compared to those for a TCP New-Reno flow.

4.4.1 Variations in Congestion Window

Since *TCP Westwood* sets *ssthresh* and *cwnd* based on current estimated bandwidth (which is much higher than the actual available bandwidth), the congestion window is observed to be very high (above 100 segments in this case). In the ideal case, the congestion window size should be around 18 ($\approx 2\ Mbps \times 74\ ms/8000$) segments. For a New-Reno flow, variation in the congestion window size is closer to that of the ideal case (Fig. 4.10). Therefore, a *TCP Westwood* sender will transmit more packets resulting in much higher level of congestion in a bottleneck router.

4.4.2 Variations in Bottleneck Queue Length

A RED queue is assumed at the bottleneck router for which values of *maxthresh* and *minthresh* are 150 and 100, respectively. After the first 30 seconds of simulation run, 10 *TCP Westwood* flows are added on top of the 10 TCP New-Reno flows. During the first half of the simulation the variation of the queue length is observed to be quite stable and the maximum queue length is observed to be about 120 (Fig. 4.11).

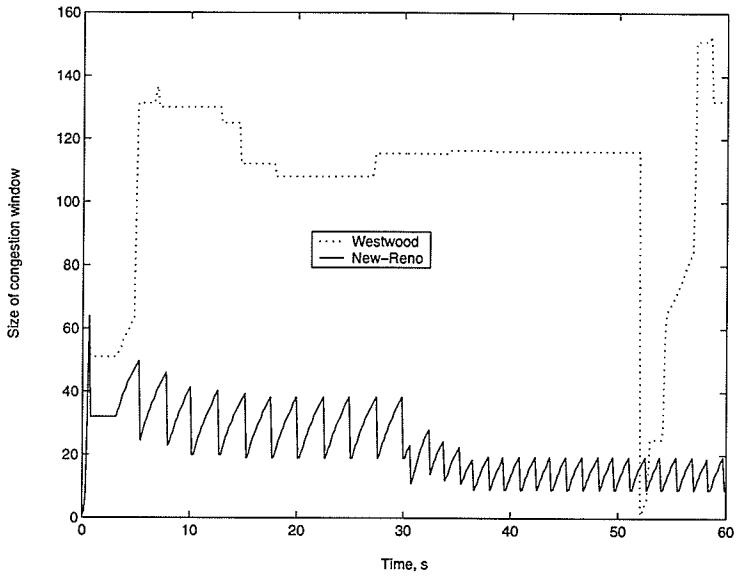

Figure 4.10. *Variations in congestion window size for TCP Westwood and TCP New-Reno (under congestionl loss only).*

However, addition of the *TCP Westwood* flows causes the queue dynamics to become more unstable and queue length is observed to vary between 0 to 185 (the maximum limit). The aggressive bandwidth estimation mechanism of *TCP Westwood* causes the queue length to exceed the limit and hence packet-drop for all the flows.

4.4.3 Variations in Throughput

The throughput performance of *TCP Westwood* is compared to that of TCP New-Reno when they run separately under conditions similar to scenario 3.

The observed throughput for a *TCP Westwood* flow is highly variable over time compared to that for a TCP New-Reno flow (Fig. 4.12). This is due to the fact that a typical *TCP Westood* flow pushes much more traffic in the network than expected and hence causes more packet losses. From the simulation traces we have observed that during the 60 seconds of simulation run the tagged *TCP Westwood* flow experienced 38 timeouts while TCP NewReno experienced only 3 timeouts. The number of packet drops for TCP Westwood has been five times more than that for TCP NewReno. As a result, the throughput for TCP New-Reno is observed to be much higher (250

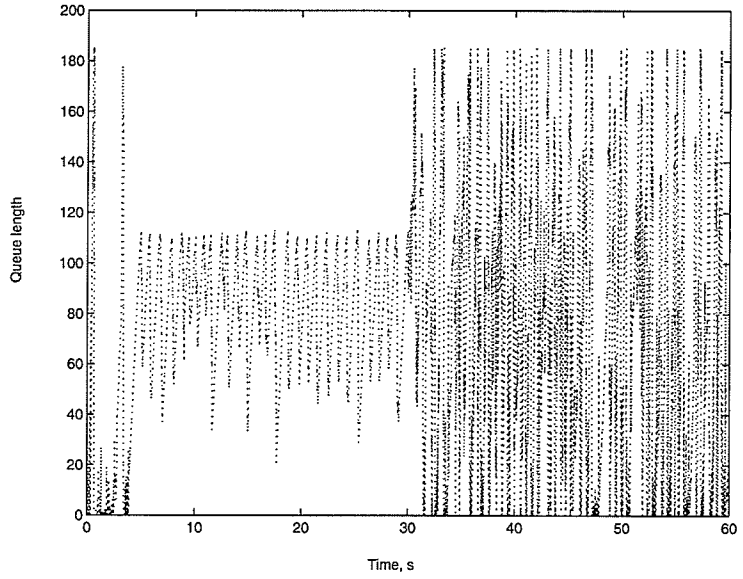

Figure 4.11. *Variations in bottleneck link queue size for TCP Westwood and TCP New-Reno (under congestionl loss only).*

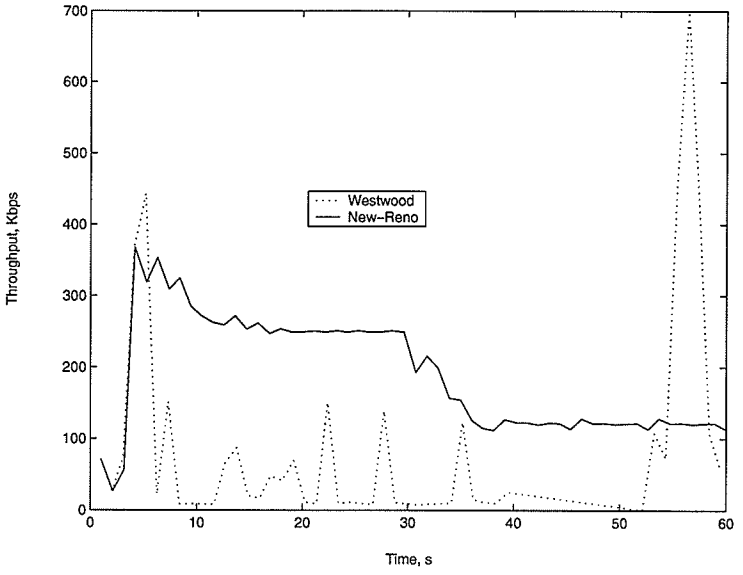

Figure 4.12. *Variations in throughput for TCP Westwood and TCP New-Reno (under congestion loss only).*

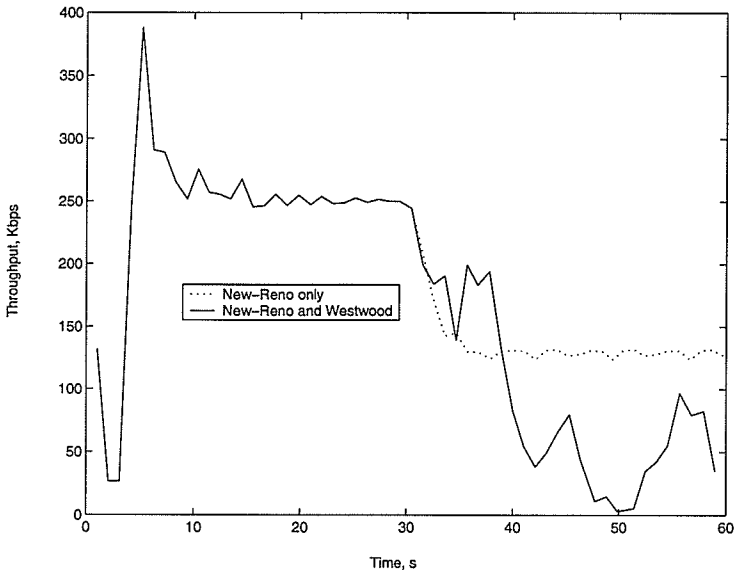

Figure 4.13. *Friendliness of TCP Westwood compared to TCP New-Reno (under congestionl loss only).*

KB/sec - 130 *KB/sec*) compared to that for *TCP Westwood* (70 *KB/sec* - 60 *KB/sec*) (Fig. 4.12).

4.4.4 Friendliness of TCP Westwood to Other TCP Flows

To determine the friendliness behavior of *TCP Westwood* compared to that of TCP New-Reno, we observe the variations in throughput for a tagged flow (flow1 in this case) in two cases–when the tagged flow co-exists with all TCP New-Reno flows and when it co-exists with *TCP Westwood* flows. For these two cases 10 new TCP New-Reno flows and 10 new *TCP Westwood* flows are added, respectively, on top of the 10 TCP New-Reno flows.

Introduction of *TCP Westwood* flows causes more fluctuation in the throughput of the tagged flow (Fig. 4.13). With the introduction of TCP New-Reno flows, the throughput of the tagged flow stabilizes to about 130 *KB/sec* while for the *TCP Westwood* case the average is around 60 *KB/sec*. Therefore, *TCP Westwood* is unfriendly towards the tagged flow and at the same time it wastes network bandwidth (since it does not utilize the bandwidth to increase its own throughput).

4.5 Summary

We have observed that the bandwidth estimation mechanism in *TCP Westwood* always overestimates the available bandwidth for a flow in the end-to-end network path. This causes *TCP Westwood* flows to become severely unfair to the other type of TCP flows and may also cause congestion collapse. Therefore, the results reported in [32] on the performance of TCP Westwood is somewhat fallacious. The cause of Westwood's overestimation and performance degradation has been addressed in next chapter 5. Moreover, we have provided alternative solution for bandwidth estimation that does not overestimate.

Chapter 5

TCP Prairie: A New TCP Variant Based on Adaptive Bandwidth Estimation

This chapter presents a new TCP variant called *TCP Prairie* which is a sender-only TCP modification based on dynamic bandwidth estimation in wide-area wireless networks. The protocol addresses the burstiness pattern of ACK arrival and estimates the bandwidth from the ACK stream. The estimate is then used to control the transmission rate. The protocol provides significant throughput improvement compared to *TCP Westwood* and TCP New-Reno and satisfactory fairness and friendliness performances.

5.1 Introduction

The original bandwidth estimation mechanism in *TCP Westwood* was observed to be erroneous (in chapter 4) which could make *TCP Westwood* unfriendly towards other TCP variants (due to over estimation of the "fair share" of the bottleneck bandwidth for the corresponding TCP flow). To alleviate this problem, enhanced estimation methods (e.g., CRB (Combined Rate and Bandwidth) estimation method [54], ABSE (Adaptive Bandwidth Share Estimation) method [55]) were proposed.

In this chapter, we propose a sender-only TCP modification (*TCP Prairie*) which

is based on a novel dynamic bandwidth estimation mechanism for TCP flows in a wired-cum-wireless network. We explore the efficacy of the proposed bandwidth estimation mechanism in *TCP Prairie* and evaluate the resulting throughput and friendliness performances of *TCP Prairie*. Comparative performance among *TCP Prairie*, *TCP Westwood* and *TCP New-Reno* is thoroughly investigated.

5.2 Motivation and Contribution

TCP Westwood is the representative of the TCP modifications based on dynamic bandwidth estimation for improving TCP performance in wired-cum-wireless networks. We observe that *TCP Westwood* measures available network bandwidth for a TCP flow from the rate of the returning ACKs. It calculates bandwidth sample by simply dividing the number of acknowledged bytes by the corresponding inter-arrival time. When the majority of the network traffic is TCP flows, due to the bursty behavior of the TCP senders and the traffic management mechanisms at the routers, this does not provide an accurate estimate of the available network bandwidth for a flow.

TCP's inherent congestion control mechanism results in a bursty transmission pattern by the TCP sender. During the slow-start phase, due to the immediate increase of congestion window by one, the TCP sender transmits two packets on reception of each ACK. Starting with an initial congestion window, repetition of this process in each round-trip time results in a transmission of a packet burst followed by an off period. Also, during the congestion avoidance phase, since the congestion window increases immediately after receiving ACKs for a full transmission window, the bursty pattern is retained. Again, the first-in-first-out queueing policy at a router results in a bursty service pattern for packets.

The cumulative effect of the above two phenomenon is the bursty ACK arrival process at the TCP sender. In a burst, ACKs are closely spaced and the bandwidth estimation based on the inter-arrival time of these ACKs at the sender does not reflect the real bandwidth available for the TCP sender. Therefore, in cases of multiple flows, the calculated sample bandwidth becomes very high, and consequently, the sender overestimates the available network bandwidth. This results in an aggressive packet

transmission by the TCP sender into the network.

Our proposed new TCP variant called *TCP Prairie* is an augmented version of *TCP New-Reno* as majority of TCP implementations are TCP New-Reno these days [59]. *TCP Prairie* is a sender-only TCP modification and is based on dynamic bandwidth estimation in a wide-area wireless network. We present novel mechanisms for bandwidth sample calculation and bandwidth estimation by monitoring the acknowledgement stream and the RTT values at the sender. Based on the RTT values, *TCP Prairie* determines the degree of congestion in the network.

Armed with the information on the degree of congestion, *TCP Prairie* spreads the acknowledgement burst over the off period (i.e., the interval between the arrival of two successive bursts) and obtains the bandwidth sample value. This gives a more accurate and safe bandwidth sample. The bandwidth estimate is obtained from the bandwidth sample using a discrete time EWMA (Exponentially Weighted Moving Average) filter whose gain varies with the change of available bandwidth. The more the change of bandwidth, the more is the gain of the EWMA filter and hence the more is the weight given to current sample.

The congestion control mechanism of *TCP Prairie* is similar to that of *TCP Westwood* or BA-TCP [57]. Whenever the sender perceives a packet loss (i.e., through three duplicate acknowledgements or a timeout) it uses the estimated bandwidth to properly set the congestion window (*cwnd*) and the slow-start threshold (*ssthresh*). By setting the *cwnd* and *ssthresh* values which are based on an accurate estimate of the available bandwidth, *TCP Prairie* avoids reduction in congestion window and slow-start threshold which is neither excessive nor insufficient. Therefore, *TCP Prairie* achieves both faster recovery and effective congestion avoidance.

TCP Prairie is observed to be a very effective transport protocol in handling both wireless loss and congestion loss in a wide-area wireless network. Experimental studies reveal that *TCP Prairie* offers better *throughput* performance compared to *TCP Westwood* and *TCP New-Reno*. *TCP Prairie* is observed to provide better fairness compared to *TCP Westwood*. Also, compared to *TCP Westwood*, it is observed to be more friendly towards *TCP New-Reno* flows. This is due to the fact that the bandwidth estimation in *TCP Prairie* is much more accurate compared to that in *TCP Westwood*.

5.3 End-to-End Bandwidth Estimation in *TCP Prairie*

The *TCP Prairie* sender monitors the acknowledgement stream and the RTT values corresponding to a TCP flow to estimate the available bandwidth for that flow. The available bandwidth for a flow is the amount of the share of the bottleneck bandwidth that a flow can claim without hurting other flows passing through the same bottleneck link (i.e., fair share of the bottleneck bandwidth). For N TCP flows sharing a bottleneck bandwidth with capacity C, the available bandwidth for a flow is defined in the following way:

- If there is only congestion loss, fair share for each flow is C/N regardless of the TCP mix.
- If all the TCP flows are of type *TCP Prairie*, in case of wireless loss, fair share is C/N.
- If there are n_1 *TCP New-Reno* and n_2 *TCP Prairie* flows (where $n_1 + n_2 = N$), in case of wireless loss, each of the *TCP New-Reno* flows will use a bandwidth share less than C/N. In this case each *TCP Prairie* flow can use a bandwidth share more than C/N without affecting the *TCP New-Reno* flows. Therefore, the fair share for each *TCP Prairie* flow is greater than C/N. Note that, *TCP Prairie* can provide better throughput by using this increased bandwidth share.

The mechanisms for calculating the bandwidth sample and estimating the available bandwidth are described below.

5.3.1 Calculation of Bandwidth Sample

To calculate the bandwidth sample, we assume a bursty environment where the sender transmits packets in a bursty manner and also the routers forward packets corresponding to a flow in a bursty fashion. This results in a bursty ACK arrival process at the TCP sender. Later, we will show that the derived formula also holds for non-bursty environment.

Let us assume that an ACK arriving at the sender at time t_k notifies that d_k bytes corresponding to that flow have been received at the receiver. Intuitively, $\frac{d_{k+1}-d_k}{t_{k+1}-t_k} = \frac{\Delta d_k}{\Delta t_k}$ should be considered as the bandwidth sample for the flow. However,

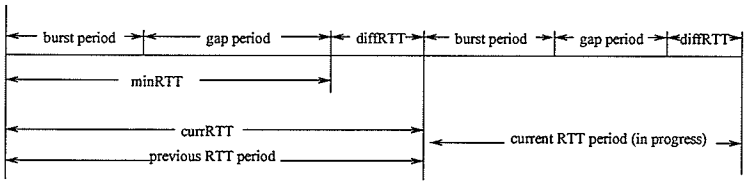

Figure 5.1. *Different parts of an RTT period.*

due to the burstiness in the ACK arrival process, Δt_k is much smaller for most cases. Therefore, $\frac{\Delta d_k}{\Delta t_k}$ may not reflect the fair share of bandwidth for the corresponding TCP flow.

As explained earlier, a TCP sender always transmits packets in a bursty fashion. From transmission perspective, a TCP source sends a burst of packets followed by an *off* period during which there is no transmission. Similarly, from reception point of view, a TCP sender receives a burst of ACKs followed by an *off* period. A burst period of transmission overlaps with the burst period of reception. Therefore, it can be said that the lifetime of a TCP flow is a collection of *burst* periods (during when it receives ACKs and transmits TCP segments and *off* period (during when there is no ACK reception and no transmission).

For simplicity, let us assume that there is a single burst period during each RTT period (actually multiple burst can be converted into a single burst period) and the first segment of the burst period is tagged for RTT. Fig. 5.1 shows the different parts in an RTT period, where *minRTT* is the minimum RTT seen by the flow throughout its lifetime, and *currRTT* is the measured value of the RTT from previous RTT period and the current RTT period is in progress. Since the first segment of the burst is tagged for RTT, the time difference between two consecutive burst is equal to RTT. Here, $(1 + acwnd)$ is the number of segments that are acknowledged as a single burst in the current RTT period, *diffRTT* is the difference between *minRTT* and *currRTT* (i.e., *diffRTT* = *currRTT* - *minRTT*). Now, we divide the *off period* (i.e., the interval between two burst periods) into two parts – *diffRTT* and *gap period.* Therefore, an RTT period is an aggregate of *diffRTT*, *burst period* and *gap period.*

During a *burst period* from t_k to $t_{k+acwnd}$, the TCP sender receives $(1 + acwnd)$ acknowledgements for data ranging from d_k to $d_{k+acwnd}$. We can say that, the service time in the bottleneck router for the data acknowledged during the burst period is

equal to the duration of the burst period. Here, the more the service time at the bottleneck router, the more will be the duration of the burst period. Therefore, the *service bandwidth* for the flow can be calculated as

$$\begin{aligned} BW_s &= \frac{\text{data burst ACKed during burst period}}{\text{burst period}} \\ &= \frac{d_{k+acwnd} - d_k}{t_{k+acwnd} - t_k}. \end{aligned} \tag{5.1}$$

Let us assume that all segments are of identical size and there is neither missing ACK nor delayed ACK. We also assume that all the ACK inter-arrival times are equal. Therefore,

$$\begin{aligned} BW_s &= \frac{d_{k+acwnd} - d_k}{t_{k+acwnd} - t_k} \\ &= \frac{(d_{k+1} - d_k) \times acwnd}{(t_{k+1} - t_k) \times acwnd} \\ &= \frac{\Delta d_k}{\Delta t_k}. \end{aligned} \tag{5.2}$$

This service bandwidth $\frac{\Delta d_k}{\Delta t_k}$ is exactly similar to our intuitive bandwdith sample. It is much higher and almost equal to the highest service rate of the bottleneck link (under the assumption that all traffic through the bottleneck link are due to TCP flows).

The same burst suffers *diffRTT* additional delay before being served at the bottleneck router and the total time experienced by the burst in the bottleneck router is *(burst period + diffRTT)*. Therefore, we define the *experienced bandwidth* of the flow as follows:

$$\begin{aligned} BW_e &= \frac{\text{data burst ACKed during burst period}}{\text{burst period + diffRTT}} \\ &= \frac{d_{k+1} - d_k}{(t_{k+1} - t_k) + diffRTT/acwnd}. \end{aligned} \tag{5.3}$$

This experienced bandwidth BW_e is still much higher than fair share of bandwidth when there is congestion in the network.

Now, we define the parameter α $(0 \leq \alpha \leq 1)$ which determines the degree of congestion. The more the congestion in the network, the more is the delay in the bottleneck router and the more is the value of RTT. We define α to be equal to $min(1,$

$\gamma \times diffRTT/minRTT$), where γ is a scaling factor for the normalized *diffRTT* with respect to *minRTT*. For all of our simulations, we set the value of γ to be equal to 1.5. Note that, the value of α approaches one as the congestion in the network increases.

In case of fully congested situation, the rate of ACK reception per RTT period (which is a measure of throughput) can be regarded as the fair share of bandwidth. In a fully congested situation, fair share cannot be higher than achieved throughput as it will hurt other flows. But for partial congestion or no congestion, the fair share is certainly higher than the achieved throughput. Considering that the portion of the *gap period* contributing to the total time is $\alpha\times$ *gap period*, the approximate fair share of bandwidth for the flow (BW_a) can be defined as follows:

$$\begin{aligned} BW_a &= \frac{\text{data burst ACKed during burst period}}{\text{burst period} + diffRTT + \alpha \times \text{gap period}} \\ &= \frac{d_{k+1} - d_k}{(t_{k+1} - t_k) + diffRTT/acwnd + \alpha \times \text{gap period}/acwnd}. \end{aligned} \tag{5.4}$$

In presence of congestion (i.e., when α is large), BW_a is close to the fair share of the bandwidth for the flow. But in case of wireless loss, the value of α becomes small and BW_a goes far away from the fair share of bandwidth. When wireless loss dominates, congestion does not develop and the value of *diffRTT* and hence α becomes very small.

There are cases when multiple flows pass through the bottleneck link, but sufficient congestion does not develop at the router (due to small receiver window and/or packet loss due to wireless link error). In these cases, *diffRTT* and α become very small. Consequently, BW_a becomes too high and almost similar to the service bandwidth BW_s. To alleviate this problem, we modify (5.4) by introducing a new term $(1-\alpha)\times v$ in the denominator. The parameter v should increase with increasing number of flows through bottleneck link.

Finding the exact value of v is itself a research issue. We set v to be equal to the measured variance of inter-arrival time (which increases with increasing number of flows). From the simulation results we have observed that this setting works quite well. Therefore, the actual fair share of bandwidth is calculated as

$$BW_f = \frac{d_{k+1} - d_k}{(t_{k+1} - t_k) + diffRTT/acwnd + \alpha \times \text{gap period}/acwnd + (1-\alpha) \times v}. \tag{5.5}$$

Now, if we allow delayed ACKs or missing ACKs, then consecutive acknowledgements may acknowledge more than one segment. Defining $\frac{d_{k+1}-d_k}{seg\_size} = c$ (where $c \geq 1$), the formula for the fair share of bandwidth can be written as

$$BW_f = \frac{d_{k+1} - d_k}{(t_{k+1} - t_k) + c \times diffRTT/acwnd + \alpha \times c \times \text{gap period}/acwnd + (1-\alpha) \times c \times v}. \quad (5.6)$$

In case of equi-spaced (or non-bursty) ACKs, ACKs are distributed evenly over the *gap period* and the *gap period* vanishes. For the same reason, all inter-arrival times become almost same and v also vanishes, however, the term $c \times diffRTT/acwnd$ does not vanish which helps us to obtain a conservative sample when the bottleneck router is congested.

By using (5.6), we obtain a reasonably accurate estimate of the available bandwidth for a flow in both bursty and non-bursty situations. Experimental results in Section "Accuracy of Bandwidth Estimation" validate this claim.

5.3.2 Bandwidth Estimation

To estimate the available bandwidth from the bandwidth sample, we use an EWMA filter whose gain varies with the change in the bottleneck link capacity. Note that, a constant gain exponential filter is not suitable in such a scenario since its response time is high when the available network bandwidth changes. Also, we do not consider the filter used in *TCP Westwood* due to the following two reasons:

- Being a low pass filter, it provides more weight to the sample which has higher Δt_k. Consider the case when bottleneck bandwidth increases due to reduction of non-TCP traffic (e.g., CBR traffic). In this case Δt_k decreases and the filter provides lower weight to the sample, though the better strategy would be to provide more weight to this sample to obtain a faster response.
- We have observed through simulations that most larger inter-arrival times of ACKs are accompanied by larger inter-transmission time of the corresponding segments. Providing more weight to samples with larger Δt_k without considering inter-transmission time is questionable.

We propose the following EWMA filter whose gain is adaptive to the network load:

$$\begin{aligned} P_i &= (1 - K_{i-1})P_{i-1} + (\delta + |prevRTT - currRTT|)/\delta \\ R_i &= minRTT/\delta \\ K_i &= P_i/(P_i + R_i) \\ \hat{b_i} &= (1 - K_i)\hat{b_{i-1}} + K_i b_i. \end{aligned} \tag{5.7}$$

In (5.7) K_i is the gain of the filter at step i, δ is the minimum ACK inter-arrival time, $prevRTT$ is the RTT measured before $currRTT$, P_i is the estimate of the change of bottleneck load at step i, R_i is the round-trip pipe size (i.e., the network bandwidth-delay product in packets), b_i is the bandwidth sample at step i, and $\hat{b_i}$ is the filtered estimate of bandwidth at step i.

Note that, in the expression for P_i, $|(prevRTT - curRTT)/\delta|$ represents the change of bottleneck load during an RTT period. Also, note that, K_i increases with increasing P_i. The rationale behind such formulation of K_i is that the more the change in the bottleneck buffer, the more the change in bandwidth should be and hence the more the value of K_i should be.

5.4 TCP Prairie Congestion Control Mechanism

In this Section we describe how the estimated available bandwidth (EB) is used for congestion control at a *TCP Prairie* sender. Firstly, in *TCP Prairie* the congestion window updating mechanisms during slow-start and congestion avoidance phase are the same as those in *TCP New-Reno*. That is, the congestion window increases exponentially and linearly, respectively, during the slow-start and congestion avoidance phases.

As in *TCP Westwood*, the estimated bandwidth (EB) is used to set the slow-start threshold and to tune congestion window after a slowdown event (i.e., after three duplicate ACKs are received or after a timeout occurs). In the case of reception of three duplicate ACKs, *TCP Prairie* sets *cwnd* and *ssthresh* as follows:

```
if (three DUPACKs are received)
  ssthresh = EB x minRTT/seg_size;
```

```
  if(cwnd > ssthresh)
        cwnd = ssthresh;
  endif
endif
```

The standard *fast retransmit/fast recovery* then follows. During congestion avoidance phase TCP sender probes for available bandwidth by continually increasing the congestion window in a linear fashion. If it receives three duplicate ACKs, it implies that TCP has hit the bottleneck capacity or wireless loss has damaged one or more packets. In either case, the estimated bandwidth is regarded as available bandwidth for the flow. Hence, it sets *ssthresh* equal to $EB \times minRTT$. The rationale behind using $minRTT$ is that this setting allows the bottleneck queue to be drained after a congestion episode.

If a packet loss is indicated by the occurence of a timeout, *cwnd* and *ssthresh* are set as follows:

```
if (coarse timeout expires)
  ssthresh = EB x minRTT/seg_size;
  if(ssthresh < 2)
        ssthresh = 2;
  endif
        cwnd = 1;
endif
```

That is, after a timeout *ssthresh* is set equal to EB and *cwnd* is set equal to one. This behaviour is justified because a timeout indicates a more severe congestion loss and/or wireless loss. Therefore, the basic *TCP New-Reno* behavior is retained while a reasonably speedy recovery is ensured by setting *ssthresh* to the value of EB.

5.5 Performance Evaluation

The network topology used for performance evaluation is shown in Fig. 5.2, where the fixed hosts act as TCP sources and the mobile nodes (connected to the base station/access point through wireless links) act as TCP sinks. A single bottleneck

link is shared by the TCP flows and the flows have identical propagation delay and infinitely backlogged sources. The bottleneck router uses a first-come-first-served scheduling policy.

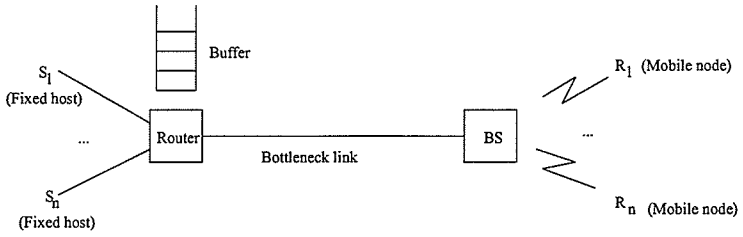

Figure 5.2. *Network topology.*

5.5.1 Accuracy of Bandwidth Estimation

Accurate bandwidth estimation is of prime importance since as it has direct impact on congestion window settings and hence on congestion control. In this Section, we compare the effectiveness of the proposed bandwidth estimation algorithm with that of *TCP Westwood*. All the results are obtained using ns-2 [23]. For this purpose, we consider five different scenarios – one case under UDP (User Datagram Protocol) traffic and four cases in presence of competing TCP flows. The throughput and friendliness performances of *TCP Prairie* and results on the interaction of *TCP Prairie* with RED queue are also presented.

5.5.1.1 Bandwidth Estimation Under UDP Traffic

The configuration simulated here has a bottleneck link capacity of 12 *Mbps*. One TCP flow (*TCP Prairie/TCP Westwood*) shares the bottleneck link with two UDP flows. The transmission rate of each UDP flow is 4 *Mbps*. The first UDP flow is turned on at 20 *second* and the second UDP flow is turned on at 40 *second*. At time 60 *second*, first UDP flow is turned off and at time 80 *second*, the second UDP flow is turned off. The UDP flows then remain silent until the end of the simulation. One *TCP Prairie/TCP Westwood* flow sends data throughout the entire simulation period.

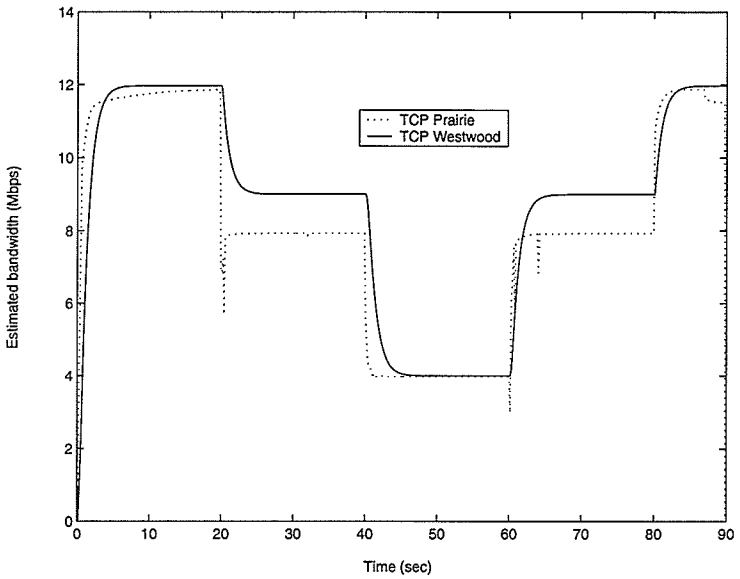

Figure 5.3. *Bandwidth estimation under UDP traffic.*

Fig. 5.3 shows the performance of the bandwidth estimation mechanisms for both *TCP Prairie* and *TCP Westwood*. In this simulation scenario, the available bandwidth for the TCP flow is 12 *Mbps*, 8 *Mbps*, 4 *Mbps*, 8 *Mbps* and 12 *Mbps* during the interval of 0-20 *second*, 20-40 *second*, 40-60 *second*, 60-80 *second* and 80-90 *second*, respectively. As is evident from Fig. 5.3, *TCP Prairie* estimates the remaining bandwidth more accurately than *TCP Westwood*. Moreover, the bandwidth estimation mechanism in *TCP Prairie* has a faster response time to the changes in the available bandwidth.

5.5.1.2 Bandwidth Estimation Under Competing TCP Flows (No Loss Scenario)

The configuration simulated here features a 20 *Mbps* bottleneck link with one way propagation time of 37 *ms*. We assume that the TCP segment size is 1 KB and all the segments are of identical size. 10 TCP flows start at 0 *second* and another 10 flows start at 30 *second*. All flows remain active until the end of the simulation (at 60 *second*). In this scenario, we assume that there is neither wireless loss nor congestion loss. To avoid congestion loss, sufficient buffer is used in the bottleneck link.

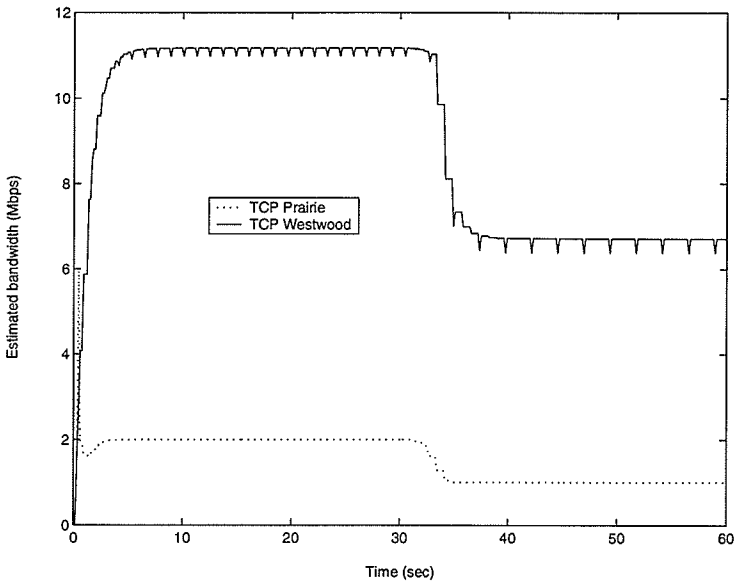

Figure 5.4. *Bandwidth estimation under TCP traffic (no loss scenario).*

We run *TCP Prairie* and *TCP Westwood* under the above stated scenario. Fig. 5.4 shows the estimated bandwidth for flow 1. The fair share of bandwidth is 2 *Mbps* during the first 30 *seconds* and 1 *Mbps* during the last 30 *seconds*. As is evident from Fig. 5.4, *TCP Prairie*'s bandwidth estimation for flow 1 matches exactly with the fair share of the bottleneck bandwidth for this flow. This is due to the fact that *TCP Prairie*'s bandwidth estimation addresses the burstiness in the network properly. The estimated available bandwidth for *TCP Westwood* is far from the fair share and the estimation error is as high as $450\% - 550\%$ in this case.

5.5.1.3 Bandwidth Estimation Under Competing TCP Flows (with 0.1% Wireless Loss)

The simulation configuration here is exactly similar to that is Section 5.5.1.2 except that we introduce wireless loss here. For wireless loss, we consider that TCP sinks are connected to the bottleneck router through wireless link of 22 *Mbps* (e.g., 802.11b WaveLAN). Packet loss is assumed to be only due to errors in wireless link (i.e., there is no congestion loss) and packet error rate in the wireless link is assumed to be 0.1%.

Fig. 5.5 shows the estimated bandwidth of both *TCP Prairie* and *TCP Westwood.*

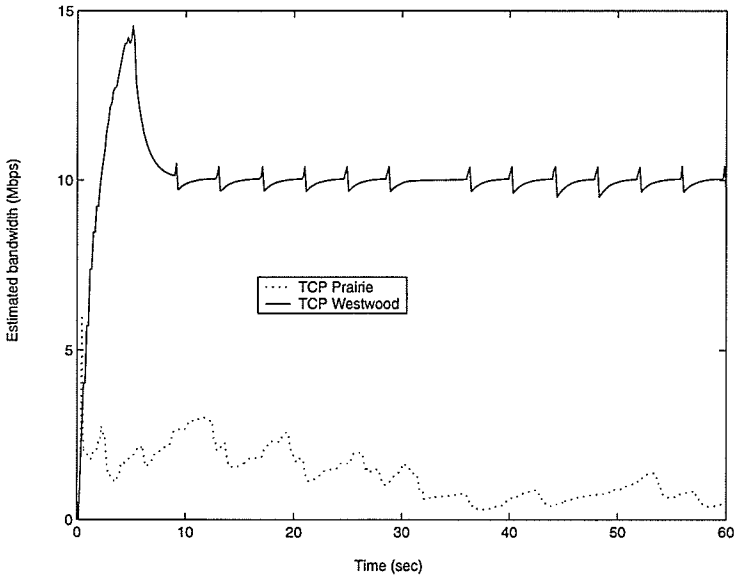

Figure 5.5. *Bandwidth estimation under TCP traffic (with wireless loss).*

As is evident from Fig. 5.5, the bandwidth estimation in *TCP Prairie* is close to the fair share while *TCP Westwood* overestimates the available bandwidth.

5.5.1.4 Bandwidth Estimation Under Competing TCP Flows (with Congestion Loss)

In this scenario, packet losses are assumed to be only due to congestion in the bottleneck router. The simulation configuration in this case is similar to that in no loss scenario case. But bottleneck buffer in this case is set equal to the bandwidth delay product (185 packets) of the network.

Fig. 5.6 shows the results on bandwidth estimation for both *TCP Prairie* and *TCP Westwood*. Similar to the previous results, *TCP Prairie*'s bandwidth estimation is closer to the fair share, but *TCP Westwood* overestimates the available bandwidth. Here, the periodic rise and fall in the estimated bandwidth (by *TCP Prairie*) is due to the variations in the bottleneck buffer occupancy.

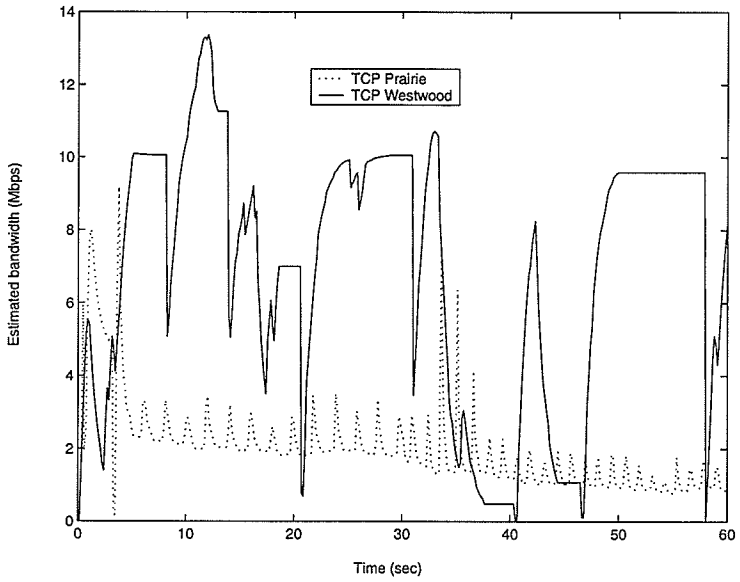

Figure 5.6. *Bandwidth estimation under TCP traffic (with congestion loss).*

5.5.1.5 Bandwidth Estimation Under Competing TCP Flows (with Congestion Loss and Wireless Loss)

The simulation configuration here is the combined cases of congestion loss and wireless loss case. The bottleneck buffer is set to 185 packets and the wireless loss rate is assumed to be 0.1%. Fig. 5.7 shows typical results on bandwidth estimation by *TCP Prairie* and *TCP Westwood* in this scenario.

In summary, *TCP Prairie* addresses the burstiness of ACK arrival process properly, and therefore, it estimates the bandwidth more accurately (compared to *TCP Westwood*) in all cases of loss and traffic types. The bandwidth estimation mechanism in TCP Westwood always overestimates the available bandwidth. The inaccuracy in bandwidth estimation in *TCP Westwood* increases with increasing number of flows. The cause of this over estimation is that TCP Westwood does not address the burstiness in the ACK arrival process.

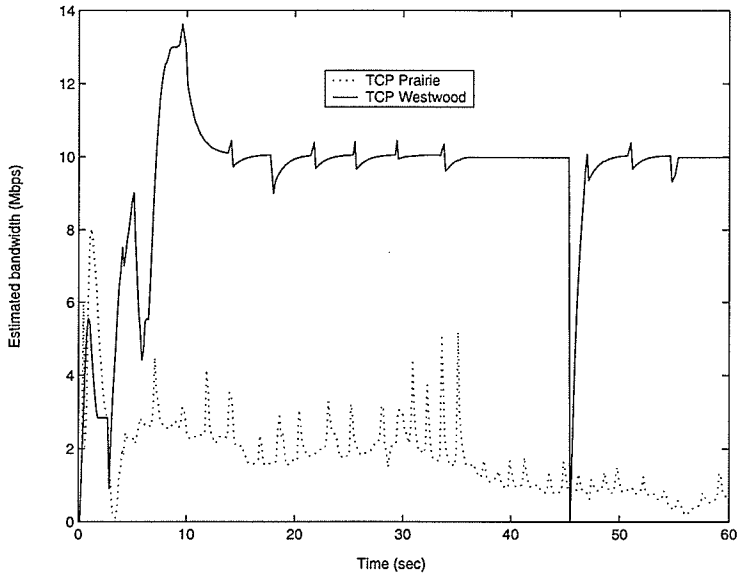

Figure 5.7. *Bandwidth estimation under TCP traffic (with congestion loss and wireless loss).*

5.5.2 Throughput, Fairness and Friendliness Performances of TCP Prairie

In this Section we compare the throughput and the fairness performances of *TCP Prairie*, *TCP Westwood* and *TCP New-Reno*. Also, we compare the friendliness performances of *TCP Prairie* and *TCP Westwood* towards *TCP New-Reno*. All the results are obtained using *ns-2* [23].

5.5.2.1 Fairness and Friendliness

Fairness is a measure for the relative throughput performance of a set of TCP flows of the same type. *Friendliness* is a measure of how the TCP flows corresponding to the different TCP variants affect the performance of each other when they share the same bottleneck link.

The simulation configuration here features a single bottleneck link with capacity 20 *Mbps* and one way propagation delay of 37 *ms*. The buffer size at the bottleneck router is equal to the pipe size (i.e., the bandwidth delay product of the network). We assume that the TCP segment size is 1 *KB* and all the segments are of identical

size. The link has no wireless loss except where otherwise stated.

To investigate the fairness performance, 10 simultaneous TCP flows of the same type are run and the throughput performance for each flow is measured. Based on the throughput for each flow, fairness index is calculated. In this case, the fairness index for *TCP Prairie*, *TCP Westwood* and *TCP New-Reno* are observed to be 0.9968, 0.9687, and 0.9979, respectively. Therefore, *TCP Prairie* is more fair than *TCP Westwood* and its fairness is comparable to that of *TCP New-Reno*.

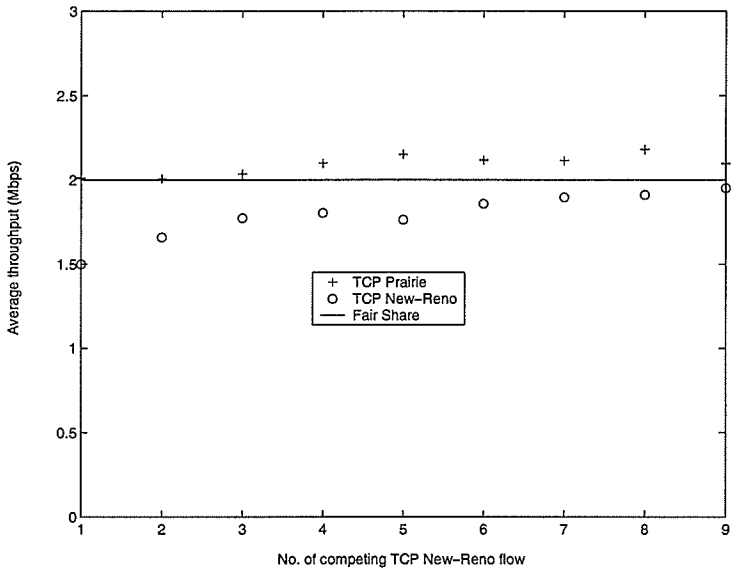

Figure 5.8. *Friendliness of TCP Prairie towards TCP New-Reno.*

Now we compare the friendliness of *TCP Prairie* and *TCP Westwood* towards *TCP New-Reno*. We run simulations with total 10 TCP flows of various schemes sharing a 20 *Mbps* bottleneck link. In Figs. 5.8-5.9, the horizontal axis shows the number of competing *TCP New-Reno* flows, the remaining flows being *TCP Prairie* (in Fig. 5.8) or *TCP Westwood* (in Fig. 5.9). The vertical axis shows the average throughput per flow in *Mbps*. In Fig. 5.9 we observe that *TCP Westwood* achieves more throughput than its fair share of 2 *Mbps*. Therefore, it is unfriendly towards *TCP New-Reno*. The reason is that *TCP Westwood* over estimates the available bandwidth share as was observed earlier.

The results in Fig. 5.8 show that the throughput achieved by a *TCP Prairie* flow is close to its fair share. Here, *TCP Prairie* flow reduces the average throughput of

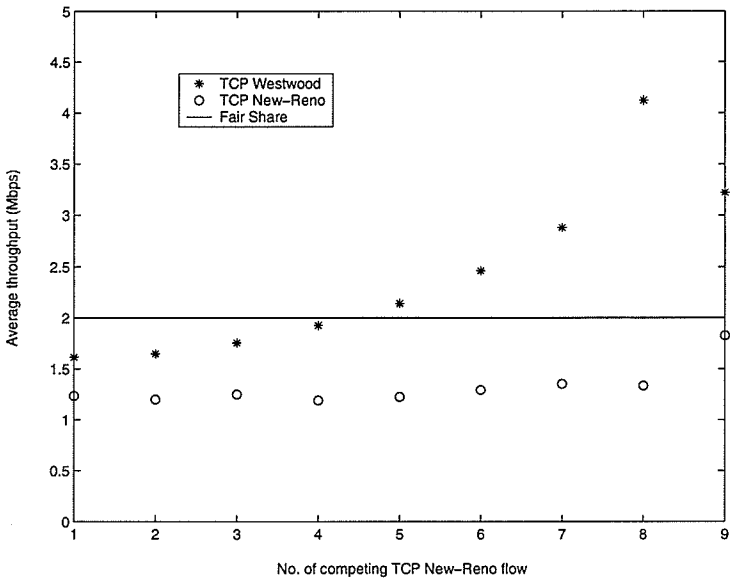

Figure 5.9. *Friendliness of TCP Westwood towards TCP New-Reno.*

TCP New-Reno flow only by minimal amount. For practical purpose, we can claim that TCP Prairie is friendly.

5.5.2.2 Interaction with RED

The purpose of RED queue management is to prevent extreme congestion and "phasing" and to enhance fairness [56]. The simulation topology in this case is similar to that in Section "Fairness and Friendliness". First 10 flows are of type *TCP New-Reno* and they start at 0 *second*. Another 10 flows are of type *TCP Prairie/ TCP Westwood* and they start at 30 *second*. All the flows remain active till the end of the simulation (at 60 *second*). The bottleneck buffer is of type RED and its capacity is equal to the network bandwidth-delay product (which is equal to 185 packets in this case). For the RED queue, the *min_thresh* and *max_thresh* are set to 110 and 150, respectively.

During interaction of RED and *TCP New-Reno* during the first 30 *seconds*, the RED router maintains a more stable queue-length, thus reducing the jitter and queuing delay. This trend is also observed during the last 30 *seconds* with *TCP Prairie* in Fig. 5.10. Therefore, *TCP Prairie* interoperates positively with RED algorithm and *TCP New-Reno*. As is observed in Fig. 5.11, with the introduction of *TCP Westwood*

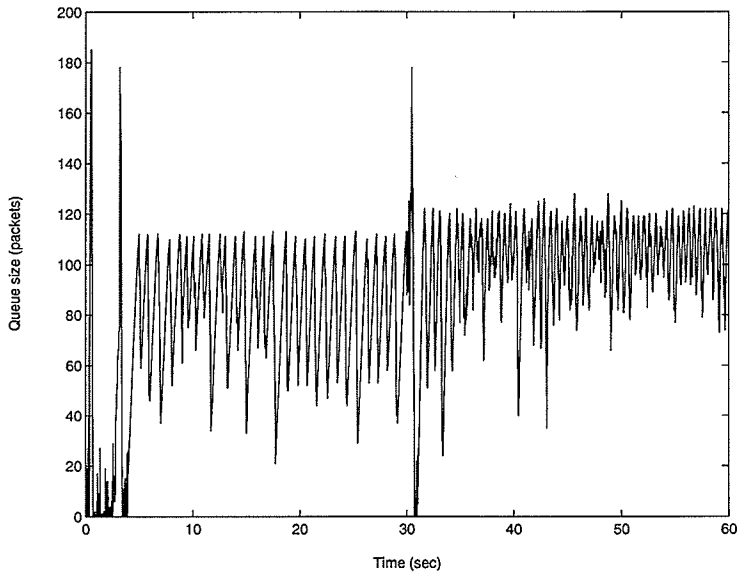

Figure 5.10. *Typical variations in RED queue with TCP New-Reno and TCP Prairie flows.*

(at 30 *second*), the queue-length becomes unstable and it varies quite rapidly. This type of interaction between the RED router and *TCP Westwood* may lead to global network instability [58].

5.5.2.3 Throughput Under Congestion Loss

To evaluate throughput under congestion loss, we consider a 20 *Mbps* bottleneck link with round-trip delay of 80 *ms*. In this case, 10 TCP flows of the same type run over the bottleneck link. The size of the bottleneck buffer is set to 200 packets (which is equal to the bandwidth delay product of the newtork). We run three different simulations - one with *TCP New-Reno*, one with *TCP Prairie* and the one with *TCP Westwood*. The average per flow throughput is observed to be 1.97 *Mbps*, 1.97 *Mbps* and 1.54 *Mbps* for *TCP Prairie*, *TCP New-Reno TCP Westwood*, respectively. *TCP Westwood* suffers performance degradation due to the fact that it suffers more congestion loss because of its over estimation of the available bandwidth. In fact, it is critical to measure bandwidth accurately and use it to improve TCP throughput. *TCP Prairie* offers superior performance because of its more accurate bandwidth

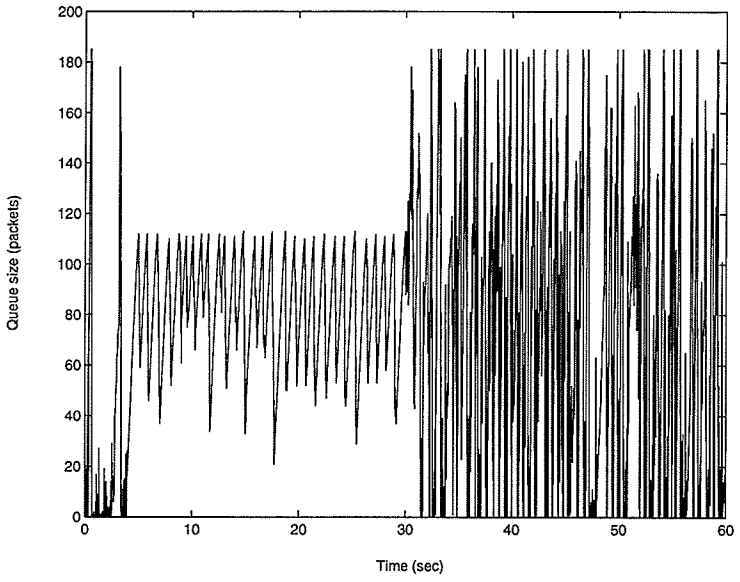

Figure 5.11. *Typical variations in RED queue with TCP New-Reno and TCP Westwood flows.*

estimation.

5.5.2.4 Throughput Under Wireless Loss

We examine a number of different scenarios to compare the performances of *TCP Prairie* and *TCP Westwood* to the performance of *TCP New-Reno* in wired-wireless environment. The bandwidth of the bottleneck link is assumed to be 20 *Mbps*. The propagation time over the wired link is assumed to be 37 *ms*. The wireless portion of the link has capacity 22 *Mbps* with negligible propagation time. The wireless link connects the base station to a destination mobile terminal.

Fig. 5.12 shows the throughput performance of *TCP Prairie*, *TCP Westwood* and *TCP New-Reno* under varying loss rate from 0% to 5%. *TCP Prairie* and *TCP Westwood* offer higher throughput than *TCP New-Reno* under different loss rates. Note that, *TCP Prairie* does better than *TCP Westwood* at 0.1% loss rate. The reason is that *TCP Westwood* suffers some congestion loss at lower error rate due to its over estimation. The largest improvement is obtained for 0.01%-1% loss rate. The performance gain of *TCP Prairie* over *TCP Westwood* is 17% at 0.01% error rate and

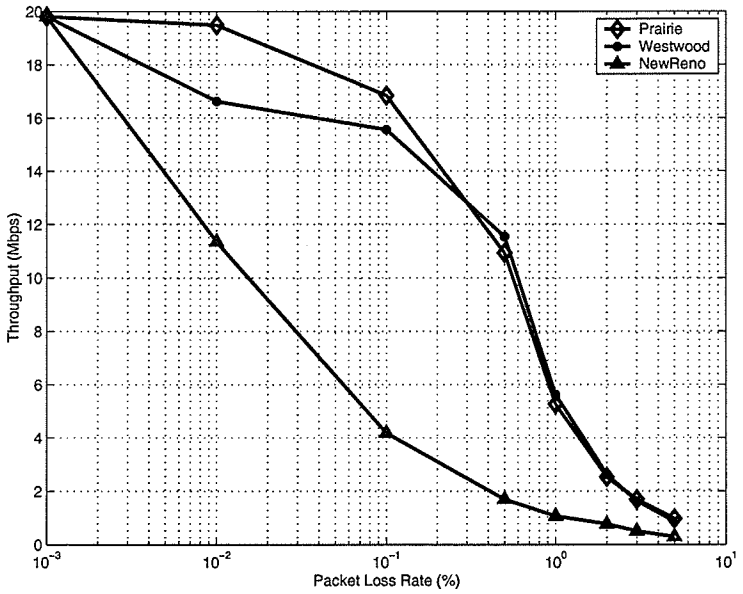

Figure 5.12. *Typical variations in TCP throughput with packet loss rate.*

it is 542% over *TCP New-Reno* at 0.5% error rate.

Simulation results in Fig. 5.13 show that the performance gains for *TCP Prairie* and *TCP Westwood* (compared to *TCP New-Reno*) increase with increasing bottleneck bandwidth. It is assumed that the error rate here is 0.1% and one way propagation time is 37 *ms*. *TCP Prairie* offers better performance than *TCP Westwood*. Therefore, *TCP Prairie* is more effective in utilizing the high capacity links *en route* between the fixed host and the mobile host. Note that, when the link bandwidth is small, all the protocols are equally effective. This is because a small window is adequate in this case and window optimization is not a significant issue.

Fig. 5.14 shows variations in throughput gain with round trip propagation time. We run simulations with one way propagation time in the wired portion of the network varying from 30-180 *ms* and with wireless link loss rate set to 0.1%. As is evident from Fig. 5.14, both *TCP Prairie* and *TCP Westwood* perform significantly better than *TCP New-Reno*.

In short, *TCP Prairie*, while achieving friendliness towards *TCP New-Reno* in presence of congestion loss, does not suffer efficiency degradation due to wireless loss.

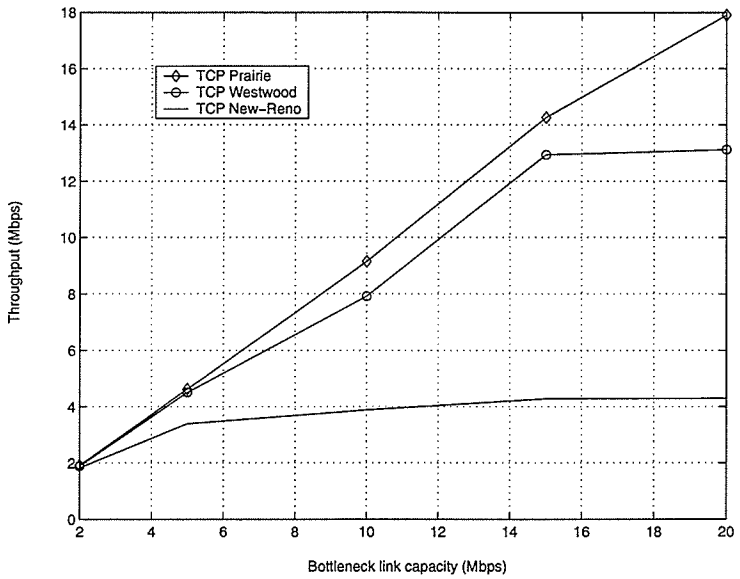

Figure 5.13. *Typical variations in TCP throughput with bottleneck link capacity.*

5.6 Summary

In this chapter, we have proposed a sender-only TCP modification called *TCP Prairie* based on a novel bandwidth estimation mechanism in heterogeneous wired-wireless network. *TCP Prairie* addresses bursty behavior of sender and router to calculate bandwidth sample. Bandwidth samples are further smoothed through a filter whose gain is proportional to the change of bottleneck load. By incorporating these two levels of adaptivity *TCP Prairie* provides an accurate estimation of the flow bandwidth share, which turns out to be a critical factor to obtain efficiency and friendliness towards *TCP New-Reno*. We have tested proposed *TCP Prairie* protocol in single and multiple flow scenarios, with congestion loss and wireless loss, and with and without RED routers. *TCP Prairie* provides higher throughput compared to *TCP Westwood* (which is another sender-only TCP modification based on adaptive bandwidth estimation) while maintaining friendliness towards *TCP New-Reno* flow. Also, compared to *TCP Westwood*, it interacts better with RED routers.

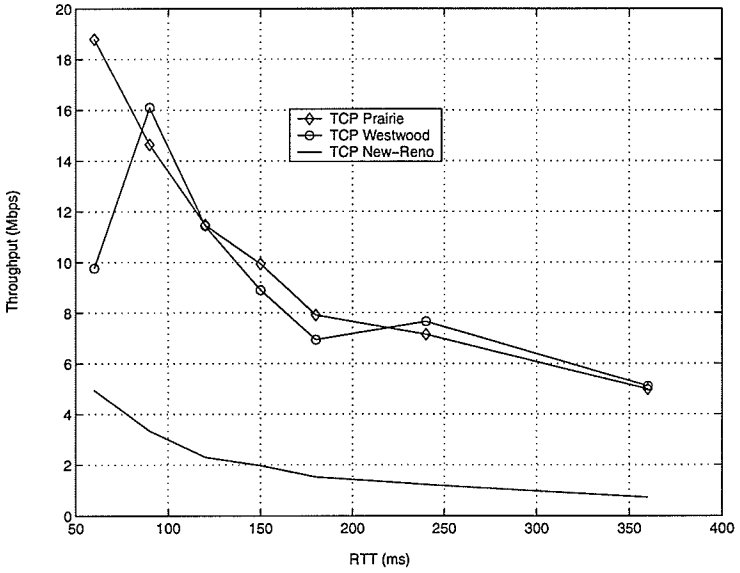

Figure 5.14. *Typical variations in TCP throughput with RTT.*

Chapter 6

Conclusion

In this chapter, we summarize the works presented in this thesis and provide a few directions for possible future research on the problem of improving TCP performance in wired-wireless networks.

6.1 Contributions

The major contributions of this thesis are as follows:

- A comparative study on throughput and fairness of basic TCP variants (e.g., TCP Tahoe, TCP Reno, TCP NewReno, SACK TCP, FACK TCP) in wireless network has been carried out. We have explained throughput behavior in terms of number of timeout, number of total fast recovery, number of multiple drop fast recovery and number of new segments transmitted during fast recovery. For uniform error cases, throughput difference comes mainly due to difference in total number of fast recovery and new segments transmitted during fast recovery. TCP having better outstanding data estimation does more new segment transmission during fast recovery and has been found to do better. For correlated error case, number of timeout and RTO backoff factor has been found to be the dominating cause. As all TCPs deal timeout in the same manner, throughput performances of all TCPs are found almost similar in case of correlated wireless channel error.
- *TCP Westwood* is a TCP variant that employs bandwidth estimation for con-

gestion control. We have detected that its bandwidth estimation is faulty, and therefore, its performance deteriorates in presence of multiple flows. Our analysis has revealed that, in the presence of multiple concurrent flows, a *TCP Westwood* flow overestimates its fair share of available bandwidth (in all the cases of congestion and/or wireless loss or no loss).

As a result, TCP Westwood becomes much aggressive in pumping packets into the network which causes significant amount of packet drops at the routers, and consequently, the throughput deteriorates. Also, *TCP Westwood* becomes unfriendly to other TCP variants and causes unfairness among *TCP Westwood* flows.

- We present a new TCP variant called *TCP Prairie* which is a sender-only TCP modification based on dynamic bandwidth estimation in wide-area wireless networks. The idea is to continuously measure the bandwidth used by a TCP flow via monitoring the rate of returning acknowledgements (ACKs) and the round-trip time values. The estimated bandwidth is then used to set the congestion window (*cwnd*) and the slow-start threshold (*ssthresh*) after a slowdown event (i.e., after three duplicate ACKs or after a timeout).

  The distinguishing feature of *TCP Prairie* (compared to other TCP variants such as *TCP Westwood*) is that it exploits the burstiness pattern of ACK arrivals and estimates the available bandwidth more accurately. For the proposed bandwidth estimation mechanism, the bandwidth sample is calculated by distributing a burst of ACKs over an off period based on degree of congestion and burstiness in the network. The estimation technique is robust against burstiness of ACK arrival and type of loss (e.g., wireless loss, congestion loss).

  Due to a more accurate bandwidth estimation, during congestion control the *TCP Prairie* sender sets the slow-start threshold to a value which is consistent with the available bandwidth for the coorresponding TCP flow. Simulation results obtained using *ns-2* reveal that *TCP Prairie* provides significant throughput performance improvement over *TCP New-Reno* and *TCP Westwood* under congestion and/or wireless loss scenarios. Also, compared to *TCP Westwood*, *TCP Prairie* is observed to be more friendly towards *TCP New-Reno.*

6.2 Directions for Future Research

This work has opened many problems and issues which require futher research. Some of them are listed below.

- All TCP enhancements including *TCP Prairie* and *TCP Westwood* set congestion window to one in the event of timeout. This is to avoid severe congestion in the network. However, if wireless loss is the main cause of timeout, setting the congestion window to one is not a good choice.
  Instead, the congestion window should be set to some value based on the estimated bandwidth or the current congestion window size. Alternatively, a seperate bandwidth estimation should be run for this purpose and that estimation should be much more conservative. This strategy can be applied only when we are sure that the timeout is due to wireless loss. Otherwise, this strategy may cause congestion collapse.
- In some cases of multiple flows with wireless loss, *TCP Prairie* has been observed to overestimate the bandwidth. To defend this situation, we have used the term $(1-\alpha)v$ in the formula for calculating the sample bandwidth. Although setting v to be equal to the variance of inter-arrival works pretty well in most cases, v can be further tuned.
- Performance of the bandwidth estimation mechanism in *TCP Prairie* needs to be investigated in case of short-lived flows.
- Performance of *TCP Prairie* needs to be compared with TCP variants such as *TCP Casablanca* [60] which are based on the idea of distinguishing random wireless losses from congestion losses and make TCP react appropriately to each kind of loss.

Bibliography

[1] R. Ayala, K. Basu and S. Elliot, "Internet technology based infrastructure for mobile multimedia services," in *Proc. IEEE WCNC'99*, pp. 109-113.

[2] L.-F. Chang, V. Varma and R. Cheng, "Architecture alternatives for PCS-to-Internet protocol interworking," in *Proc. IEEE WCNC'99*, pp. 766-770.

[3] C. Wietfeld and U. Gremmelmaier, "Seamless IP-based service integration across fixed/mobile and corporate/public networks," in *Proc. IEEE VTC'99 (Spring)*, pp. 1930-1934.

[4] T. F. La Porta, L. Salgarelli and G. T. Foster, "Mobile IP and wide area wireless data," in *Proc. IEEE WCNC'99*, pp. 1528-1532.

[5] I. Mahadevan and K. M. Sivalingam, "Quality of service architectures for wireless networks: Intserv and diffserv models," in *Proc. International Workshop on Mobile Computing, ISPAN'99*, Perth, Australia, June 1999.

[6] I. Mahadevan and K. M. Sivalingam, "Quality of service in wireless networks using enhanced differentiated services approach," in *Proc. IEEE ICCCN'99*, Boston, October 1999.

[7] S. Iren and P. D. Amer, "The transport layer: Tutorial and survey," *ACM Computer Surveys*, vol. 31, no. 4, pp. 360-405, Dec. 1999.

[8] J. Postel,"Transmission control protocol," *Internet RFC 793*, 1981.

[9] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Computer Communication Review*," vol. 18, no. 4, pp. 314-329, Aug. 1988.

[10] V. Jacobson, "Berkeley TCP evolution from 4.3-Tahoe to 4.3 Reno," in *Proc. of the 18th Internet Engineering Task Force*, University of British Columbia, Vancouver, BC, Aug. 1990.

[11] M. Allman, V. Paxson, and W. R. Stevens, "TCP congestion control," RFC 2581, Apr. 1999.

[12] W.C.Y. Lee (1993). *Mobile Communication Design Fundamentals.* 2nd Edition, John Wiley and Sons.

[13] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov, "TCP over second (2.5G) and third generation (3G) wireless networks," *Internet draft*, May 2002, URL: http://www.ietf.org/internet-drafts/draft-ietf-pilc-2.5g3g-08.

[14] S. Floyd, and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking*, May 1993, URL: http://www-nrg.ee.lbl.gov/floyd

[15] R. Cáceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 5, pp. 850-857, June 1995.

[16] A. Kumar, "Comparative performance analysis of versions of TCP in a local network with a lossy link," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 485-498, Aug. 1998.

[17] T. V. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss," *IEEE/ACM Transactions on Networking*, vol. 5, no. 3, pp. 336-350, June 1997.

[18] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and Sack TCP," *ACM Computer Communication Review*, Jul. 1996.

[19] S. Floyd and T. Henderson, "The New-Reno modification to TCP's fast recovery algorithm," *RFC 2582*, Apr. 1999. options," *RFC 2018*, Apr. 1996.

[20] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, "TCP selective acknowledgement options," *RFC 2018*, Apr. 1996.

[21] M. Mathis and J. Mahdavi, "Forward acknowledgement: Refining TCP congestion control," in *Proc. ACM SIGCOMM'96*, pp. 281-291, 1996.

[22] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communications*, 1995.

[23] S. McCanne and S. Floyd, "NS (Network Simulator)," 1995. URL http:/www.isi.edu/nsnam/ns.

[24] A. Chockalingam, M. Zorzi, L. B. Milstein, and P. Venkataram, "Performance of a wireless access protocol on correlated Rayleigh fading channels with capture," *IEEE Transactions on Communications*, vol. 46, pp. 644-655, May 1998.

[25] D. Chiu and R. Jain, "Analysis of increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN Systems*, vol. 17, pp. 1-14, June 1989.

[26] H. Balakrishnan, S. Seshan and R. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 756-769, Dec. 1997.

[27] H. Balakrishnan, S. Seshan and R. H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," *ACM/Baltzer Wireless Networks*, vol. 1, no. 4, pp. 469-481, Dec. 1995.

[28] H. Balakrishnan, S. Seshan, E. Amir and R. H. Katz, "Improving TCP/IP performance over wireless networks," in *Proc. ACM MOBICOM'95.*

[29] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," in *Proc. IEEE INFOCOM'96*, 1996.

[30] A. Chrungoo, V. Gupta, H. Saran and R. Shorey, "TCP K-SACK: A simple protocol to improve performance over lossy links," in *Proc. IEEE GLOBECOM'01*, San Antonio, Texas, USA, Nov. 2001.

[31] D. Lin and H. Hung, "TCP fast recovery strategies: Analysis and improvements," in *Proc. IEEE INFOCOM'98*, Apr. 1998.

[32] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: End-to-end congestion control for wired/wireless networks," *Wireless Networks* 8, 467-479, 2002.

[33] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP congestion control over Internets with heterogeneous transmission media," in *Proc. IEEE Int. Conference on Network Protocols (ICNP'99)*, Toronto, Canada, Oct. 31-Nov. 3, 1999.

[34] U. Varshney, "Selective slow start: A simple algorithm for improving TCP performance in wireless ATM environment," in *Proc. IEEE MILCOM'97*, pp. 465-469.

[35] T. Goff, J. Moronski, D. S. Phatak and V. Gupta, "Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments," in *Proc. IEEE INFOCOM'00.*

[36] D. N. Banerjee, "Improving wireless-wireline TCP interaction," submitted to *IEEE/ACM Transactions on Networking.*

[37] P. Sinha, N. Venkitaraman, T. Nandagopal, R. Sivakumar and V. Bharghavan, "WTCP: A reliable transport protocol for wireless wide-area networks," in *Proc. ACM MOBICOM'99*, Seattle, Washington, Aug. 1999.

[38] R. Ramani and A. Karandikar, "Explicit congestion notification (ECN) in TCP over wireless networks," in *Proc. IEEE Int. Conference on Personal Wireless Communications (ICPWC'00)*, pp. 495-499.

[39] S. Floyd, "TCP and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, no. 5, pp. 10-23, Oct. 1994.

[40] R. Ludwig and R. H. Katz, "The Eifel algorithm: Making TCP robust againt spurious retransmissions," *ACM Computer Communication Review*, vol. 30, no. 1, Jan. 2000.

[41] N. Vaidya, "Overview of work in mobile-computing," http://www.cs.tamu.edu/faculty/vaidya/slides.ps.

[42] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *Proc.*

15th IEEE Int. Conf. Distributed Computing Systems (ICDCS), pp. 136-143, May 1995.

[43] H. Balakrishnan and R. Katz, "Explicit loss notification and wireless web performance," in *Proc. IEEE GLOBECOM'98, Internet Mini Conference*, Sydney, Australia.

[44] S. Goel and D. Sanghi, "Improving TCP performance over wireless links," in *Proc. of IEEE Region Ten Conference on Global Connectivity in Energy, Computer Communication and Control (TENCON'98)*, Dec. 1998.

[45] S. Biaz and N. Vaidya, "TCP over wireless networks using multiple acknowledgements," Texas A&M University, Technical Report 97-001, Jan. 1997.

[46] K.-Y. Wang and S. K. Tripathi, "Mobile-end transport protocol: An alternative to TCP/IP over wireless links," in *Proc. IEEE INFOCOM'98.*

[47] K. Ratnam and I. Matta, "WTCP: An efficient mechanism for improving TCP performance over wireless links," in *Proc. Third IEEE Symposium on Computers and Communications (ISCC'98)*, Athens, Greece, June 1998.

[48] M. C. Chan and R. Ramjee, "TCP/IP performance over 3G wireless links with rate and delay variation," in *Proc. ACM MOBICOM'02*, Sept. 2002.

[49] A. DeSimone, M. C. Chuah and O. C. Yue, "Throughput performance of transport-layer protocol over wireless LANs," in *Proc. IEEE GLOBECOM'93*, Dec. 1993.

[50] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP Performance over wireless networks at the link layer," *ACM Mobile Networks and Applications*, Special Issue on Mobile Data Networks: Advanced Technologies and Services, vol. 5, no. 1, 2000, pp. 57-71.

[51] N. Vaidya and M. Mehtha, "Delayed duplicate acknowledgements: A TCP-unaware approach to improve performance of TCP over wireless links," Texas A&M University, Technical Report 99-003, Feb. 1999.

[52] C.-F. Chiasserini and M. Meo, "Improving TCP over wireless through adaptive link layer setting," in *Proc. IEEE GLOBECOM'01*, San Antonio, TX, Nov. 2001.

[53] J. W. K. Wong and V. C. M. Leung, "Improving end-to-end performance of TCP using link-layer retransmissions over mobile internetworks," in *Proc. IEEE ICC'99*, pp. 324-328.

[54] R. Wang, M. Valla, M. Y. Sanadidi, B. Ng and M. Gerla, "Efficiency/friendliness tradeoffs in TCP Westwood," in *Proc. IEEE Symposium on Computers and Communications*, Taormina, Italy, July 2002.

[55] R. Wang, M. Valla, M. Y. Sanadidi and M. Gerla, "Adaptive bandwidth share

estimation in TCP Westwood," in *Proc. IEEE Globecom 2002*, Taipei, Taiwan, R.O.C., Nov. 17-21, 2002.

[56] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, Aug. 1993, pp. 397-413.

[57] M. Gerla, W. Weng and R. Cigno, "Bandwidth feedback control of TCP and real-time sources in the Internet," in *Proc. Globecom'2000*, San Francisco, CA, Nov. 2000.

[58] R. J. La, "Instability of a tandem network and its propagation under RED," in *Proc. IEEE ICC'03*, Anchorage, Alaska, June 2003.

[59] J. Padhye and S. Floyd, "On inferring TCP behaviour," in *Proc. ACM SIGCOMM'01*, Aug. 27-31, 2001, San Diego, California, USA.

[60] S. Biaz and N. Vaidya, "De-randomizing congestion losses to improve TCP performance over wired-wireless networks," Technical Report CSSE03-10, Nov. 2003. URL: http://www..crhc.uiuc.edu/wireless/groupPubs.html