Improving the Performance of TCP over Wireless Links

by

SAMIR GOEL

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

May, 1997

Improving the Performance of TCP over Wireless Links

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

Samir Goel

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

May 1997

CSE-1997-M-GOE-IMP

CERTIFICATE

This is to certify that the work contained in the thesis entitled Improving the Performance of TCP over Wireless links by Samir Goel has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

[illegible]

for Dr. Dheeraj Sanghi,
Assistant Professor,
Department of Computer Science & Engineerin
Indian Institute of Technology, Kanpur.

Abstract

In this thesis, we address the problem of degradation in performance of TCP over wireless links. We propose enhancements to TCP so that it responds to certain control signals from the network. These control signals allow TCP to differentiate packet losses unrelated to congestion from those related to congestion. This enables TCP to react appropriately to different packet losses resulting in improved performance. We developed a network simulator for evaluating the performance of our scheme in a wireless computing environment. Simulation results show that our scheme is successful in substantially improving the performance of TCP over wireless links. The overhead of our scheme is also very low.

Acknowledgments

I am grateful to my thesis supervisor, Dr. Dheeraj Sanghi for his constant guidance and support throughout this thesis work. He provided a stimulating environment and gave me ample freedom to do my own thesis. His intuition and enthusiasm for networking has been very inspiring. I thank Dr. Rajat Moona for always being there to help me and for quickly reviewing the thesis report despite his busy schedule. His breadth of knowledge has been very inspiring. I also thank Dr. Pravin Bhagwat, IBM, for providing valuable inputs to the thesis work.

I thank all my friends, and especially Milind, RK, Warsi, Shah, Shyam, and Sunil, for making my stay at IIT Kanpur a memorable experience. I thank Utpal, Murthy, Senapati, and Subramanya for many fun filled and interesting conversations. I feel indebted to Vihari for his generous help in my all kinds of problems.

I thank my wonderful family for always being supportive and encouraging, and for always giving top priority to my education.

Finally, I would like to thank CSE lab staff for keeping the servers up and running.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

This thesis proposes a scheme for improving the performance of TCP [1] over wireless links.

A network simulator for simulating wireless computing environment is also implemented. It serves as a useful tool for studying the behavior of TCP in this environment.

1.1 Motivation

Recent years have witnessed rapid-progress in the area of wireless computing. This has largely been driven by growing popularity of laptop computers coupled with advances in wireless technology, enabling laptop computers to be equipped with wireless interfaces. The current trend strongly suggests that wireless links will be an integral part of future networks.

Wireless links have fundamentally different characteristics than wired links. This is because the surrounding environment interacts with the signal, blocking signal paths and introducing noise and echoes. As a result, wireless links are characterized by low bandwidth and high error rate. Moreover, their error characteristics are bursty and time varying. This is in contrast to wired links which have high bandwidth and very low error rate.

The error characteristics of wireless links result in significant degradation in performance when hosts on wireless LAN make use of reliable transport

[1] Throughout this report, we use TCP to refer to 4.3 BSD Reno implementation of TCP

protocols such as TCP, that were originally developed for wired networks.

TCP implementation has evolved considerably since its first release. Over the years, algorithms have been added to adapt to the changing characteristics of wired networks. New congestion control policies have been introduced keeping in mind the increased reliability (significantly lower bit error rates) of today's networks. These congestion control algorithms adapt well to the changes in end-to-end delays over the lifetime of a connection. Since, packet losses are invariably due to congestion in the network, these algorithms reacts to a packet loss by reducing the sending rate and exponentially backing off the retransmission timer. These congestion control algorithms have proven beneficial in improving the performance of networks.

However, performance of TCP degrades significantly over networks with wireless links. This degradation can be attributed to the error characteristics of wireless links. It results in frequent packet corruption and hence, frequent packet losses over a wireless link. These losses are (mis)interpreted by TCP as signs of congestion in the network and its congestion control algorithm reacts by reducing its sending rate and exponentially backing off its retransmission timer. This unnecessary reduction in sending rate leads to degraded performance.

The performance degradation of TCP poses an important challenge to the researchers. TCP is the most widely used transport layer protocol on the Internet today. TCP traffic constitutes 80% of all the wide-area network traffic [CDJM91]. A large set of applications use TCP as their transport layer protocol. If these applications are to operate with similar effectiveness over wireless links, without requiring any modification, performance problem of TCP needs to be addressed.

The performance problem of TCP over wireless links has been studied by researchers and several solutions [NED94, BB95, YB94, BSAK95] have been proposed. Nanda *et al.* [NED94] have suggested to solve the problem by introducing reliability at the link layer for wireless link. Badri *et al.* [BB95] and Yavatkar *et al.* [YB94] advocate splitting end-to-end TCP connection into two separate TCP connections - one over the wired network and the other over the wireless link. Balakrishnan *et al.* [BSAK95] have suggested augmenting the routing code at the base station with a module that keeps a cache of unacknowledged TCP packets on a connection. The module monitors every TCP packet (data and acknowledgment) that passes through the connection and performs intelligent processing, utilizing the knowledge of TCP. These proposals are discussed in greater detail in Chapter 3. Although, each of these solutions do achieve improvement in TCP's performance, as we will see, these solutions either have a significant overhead associated with them or they are not able to fully achieve the goal of performance improvement.

In this thesis, we describe a scheme to improve the performance of TCP over wireless link and present simulation results. Our scheme is based on the observation that without the co-operation of source TCP, one cannot achieve the goal of substantial performance improvement with minimal overhead. In our scheme, we propose enhancements to TCP so that it responds to certain control signals from the network. Simulation results shows substantial improvement in performance of TCP based on our scheme.

As part of this thesis work, we have also developed a network simulator for simulating wireless computing environment. This has served as a useful tool in testing our ideas. All the simulation results presented in this report were obtained using this simulator.

1.2 Organization of the thesis

The rest of the thesis is organized as follows. In **Chapter 2**, we present a brief overview of TCP and discuss TCP congestion control algorithms.

In **Chapter 3**, we briefly review the work that has been carried out in this field and point out drawbacks of each of the approach proposed so far.

In **Chapter 4**, we discuss the features and design of our network simulator.

In **Chapter 5**, we discuss our scheme. We describe the approach that we have taken and propose a scheme based on this approach. In this chapter, we also present the simulation results.

Finally, we conclude the thesis in **Chapter 6** and provide way to extend this study.

Chapter 2

TCP Overview

This chapter provides a brief overview of TCP. The goal of this chapter is to describe the reaction of TCP to a packet loss. Since TCP links packet loss to congestion in the network, a packet loss brings TCP's congestion control algorithms in action. This chapter briefly explains TCP's congestion control algorithms : slow-start, congestion avoidance, fast retransmit, and fast recovery.

2.1 TCP basics

TCP (Transmission control protocol) is a reliable, connection-oriented, byte-stream transport protocol. It receives data from the application, breaks it into segments and passes them onto IP layer for transmission. IP layer provides a best-effort service. Packets may get lost, duplicated, or re-ordered while traveling through the network. Therefore, TCP is designed to provide reliability by using checksum methods, sequence number, and acknowledgments. Since data as well as acknowledgment may get lost in the network, TCP maintains a timer. If an acknowledgment is not received before the timer expires, it retransmits the segment. Additional complexity in TCP is required as packets may get out of order while traveling through the network. TCP at the receiving end re-sequences packets before passing them, in correct order, to the application.

TCP is a byte stream protocol. Using a TCP connection, two applications exchange a stream of bytes. TCP does not introduce any record boundaries in the stream. So, if an application on one end writes 100 bytes, followed by 200 bytes, the application at the other end cannot identify the size of individual writes. It can read this 300 bytes of data using any combination of read sizes (three reads of 100 bytes each, single read of 300 bytes, etc.). If we consider a stream of bytes flowing in one direction between two applications, TCP numbers each byte with a sequence number. At the receiving side, TCP sends acknowledgment when it receives data from the sender. TCP supports cumulative acknowledgment policy. TCP acknowledgment number indicates that all data bytes up through but not including the mentioned sequence number, has been received successfully. Thus, an acknowledgment number indicates the sequence number of next in-sequence data byte it expects to receive. An out-of-order segment results in generation of a duplicate acknowledgment.

TCP provides end-to-end flow control. The purpose is to prevent a fast sender from swamping a slow receiver. It requires the receiver to advertise its window size to the other end. This window size (referred to as receiver's advertised window size) indicates the number of bytes of data, starting with the one specified in the acknowledgment number field, that the receiver is willing to accept. The sending side uses this information to send only as much data as the receiver has buffers for.

TCP also supports congestion control. It keeps an estimate of bandwidth available on the network path and accordingly, adjusts its sending rate and the amount of data in-transit, based on this estimate.

2.2 Slow-start and Congestion Avoidance

At any point during data transfer, the sender TCP is in one of the two modes - slow start and congestion avoidance. In slow start, sender increases its sending rate more aggressively. The idea is to quickly reach the equilibrium point for the connection. A connection is said to be running at equilibrium if it is utilizing the available network resources optimally. After reaching the equilibrium point, the connection shifts to congestion avoidance mode. In this mode, it slowly increases its sending rate to probe the maximum network capacity on the network path.

In order to implement slow start and congestion avoidance algorithm, TCP uses three variables - receiver's advertised window size (*snd_wnd*), congestion window (*cwnd*), and slow start threshold size (*ssthresh*). The value of *ssthresh* defines the equilibrium point for the connection. The relative values of *cwnd* and *ssthresh* determines the mode. If *cwnd* is less than or equal to *ssthresh*, connection is in slow start mode, otherwise, it is in congestion avoidance mode.

In slow start mode, *cwnd* is set to maximum segment size (MSS) and is incremented by 1 MSS for every acknowledgment received. Thus, in slow start mode, *cwnd* size increases exponentially. In congestion avoidance mode, *cwnd* is incremented by 1/*cwnd* for every acknowledgment received. This results in additive increase in *cwnd* size.

2.3 TCP's reaction to a packet loss

TCP interprets following two events as an indication of a packet loss in the network :

- Occurrence of a timeout.
- Receipt of three duplicate acknowledgments.

Since packet loss caused by corruption is rare (less than 1% of all the packet losses [Jac88]), TCP interprets a packet loss as an sign of congestion in the network and invokes its congestion control algorithms. The specific action taken is dependent on the event that signaled the packet loss. In the following subsections, we briefly look at the reaction of TCP in both cases.

2.3.1 Occurrence of a timeout

Occurrence of a timeout is interpreted by TCP as a sign of severe congestion in the network. It reacts by setting *ssthresh* to one half of the current window size (minimum of *cwnd* and *snd_wnd*), invoking slow start algorithm, and retransmitting the lost packet. Thus, *cwnd* is set to 1 MSS and is incremented by 1 MSS for every acknowledgment received. When *cwnd* becomes greater than *ssthresh*, TCP switches to congestion avoidance mode.

TCP's reaction is illustrated in figure 2.1 and 2.2. Figure 2.1 shows a plot of sequence number and acknowledgment number against time. Figure 2.2 shows the plot of *cwnd* and *ssthresh* against time.

During simulation, the values of MSS and *snd_wnd* were kept at 1 KB and 4 KB, respectively.

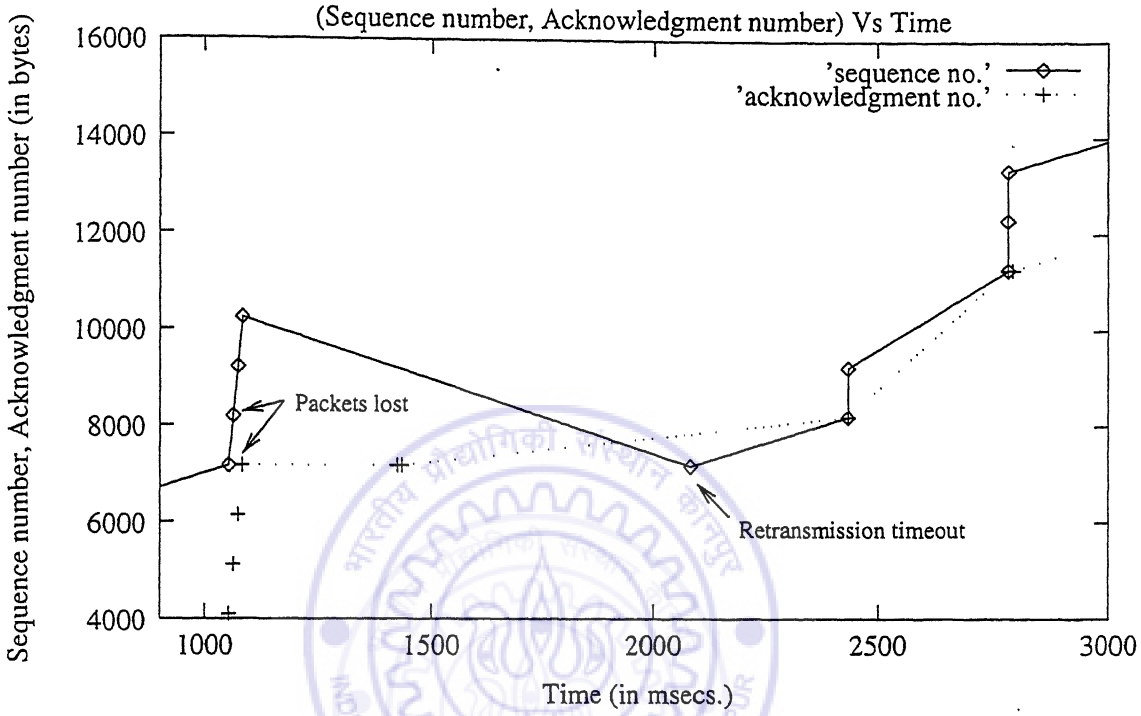

Figure 2.1: Plot of sequence number and acknowledgment number versus time

As shown in figure 2.1, two segments are lost in the network. The remaining two segments successfully reach the destination. Since these segments are not the expected ones, destination queue these segments and generates a duplicate acknowledgment corresponding to each of these. These duplicate acknowledgments indicate the next sequence number expected at the destination.

When the retransmission timer expires, source invokes slow start algorithm and retransmits the lost segment. As per the slow start algorithm, *ssthresh* is set to half of current window size and *cwnd* is set to 1 MSS (figure 2.2). The retransmitted segment reaches the destination successfully. When the source receives the acknowledgment for this segment, it increases its *cwnd* by 1 MSS and transmit two segments. The first of these two segments fills the hole at the destination. The destination passes this segment and the two previously queued segments to the application layer and gener-

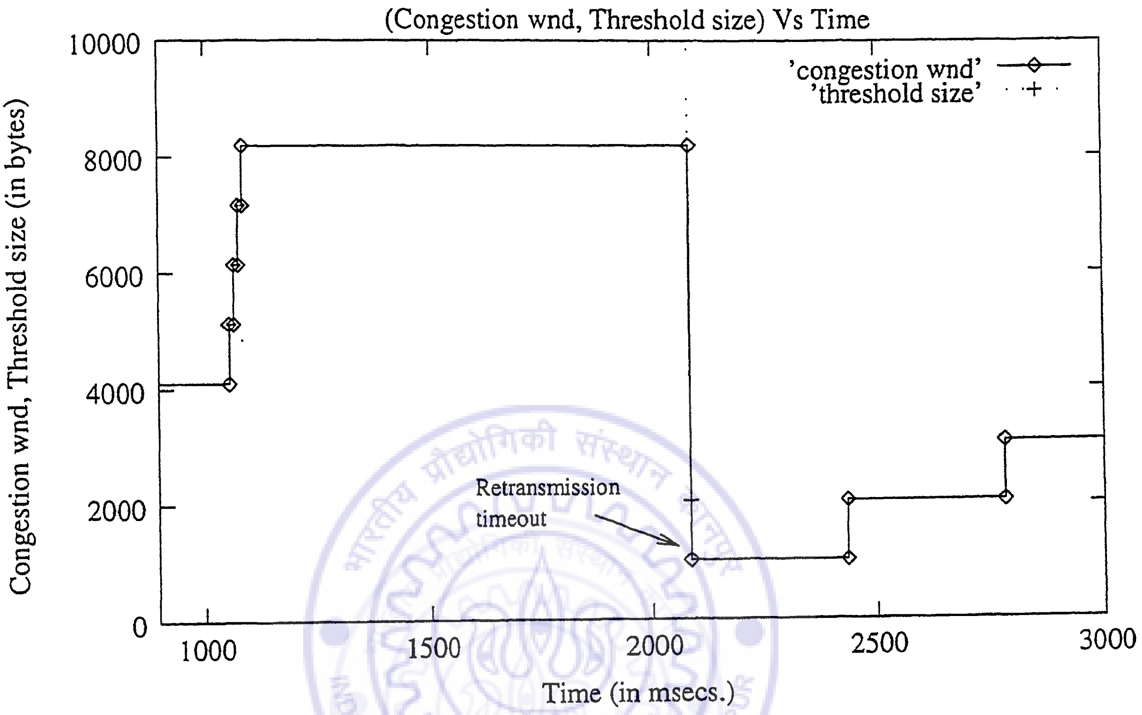

Figure 2.2: Plot of *cwnd* and *ssthresh* versus time

ates an acknowledgment, acknowledging all the segments received so far.

Note that the second of the two consecutively sent segments, was already received at the destination. It results in generation of a duplicate acknowledgment.

2.3.2 Receipt of three duplicate acknowledgments

When an out-of-order segment is received, the receiver TCP generates a duplicate acknowledgment. The purpose of this duplicate acknowledgment is to let the other end know that a segment was received out-of-order and to convey the next in-order sequence number expected. Since, a duplicate acknowledgment is caused either by a lost segment or by re-ordering of segments in the network, TCP waits for a small number of duplicate acknowledgments before making any inference. It is assumed that re-ordering of segments will not result in more than two duplicate acknowledgments. If three or more duplicate acknowledgments are received in a row, it is a strong indication that a segment was lost in the network. TCP interprets this as a sign of mild congestion in the network. On receipt of the third consecutive duplicate acknowledgment, it retransmits the lost segment (segment with sequence number equal to the one indicated in the acknowledgment number field of the duplicate acknowledgments) without waiting for the retransmission timer to expire. This is *fast retransmit algorithm.* Since an acknowledgment (duplicate or non-duplicate) can only be generated when the other end has received a segment, duplicate acknowledgments indicate that data is still flowing between the two ends. TCP therefore performs congestion avoidance instead of slow start, to avoid reducing the flow abruptly. This is *fast recovery algorithm.*

The fast retransmit and fast recovery algorithms are implemented as follows :

When TCP receives the third duplicate acknowledgment, it sets *ssthresh* to half of current window size (minimum of *cwnd* and *snd_wnd*). It retransmits the missing segment and sets the *cwnd* to *ssthresh* + 3*MSS. For each subsequent duplicate acknowledgment that it receives, it increments *cwnd* by 1 MSS and transmits a new packet if allowed by current window size. When a new acknowledgment arrives, acknowledging some data, TCP comes out of fast recovery algorithm. It sets *cwnd* to *ssthresh* and follows congestion avoidance algorithm.

Figure 2.3 and 2.4 illustrate the reaction of TCP to receipt of more than three duplicate acknowledgments. Figure 2.3 shows a plot of sequence number and acknowledgment number against time. Figure 2.4 shows a plot of *cwnd* and *ssthresh* against time.

During simulation, the values of MSS and *snd_wnd* were kept at 1 KB and 32 KB, respectively.

As shown in figure 2.3, a segment is lost on the network. All the subsequent segments successfully reach the destination and results in generation of duplicate acknowledgments. The third duplicate acknowledgment causes fast retransmit algorithm to be invoked. *ssthresh* is set to half of current window size, *cwnd* is set to *ssthresh* + 3*MSS (figure 2.4), and the lost segment is retransmitted.

For every duplicate acknowledgment received, *cwnd* is incremented by 1 MSS and, if allowed by current window size, a new segment is transmitted. As shown in figure 2.3, the duplicate acknowledgments cause three new segments to be transmitted.

The retransmitted segment fills the hole at the destination. Destination

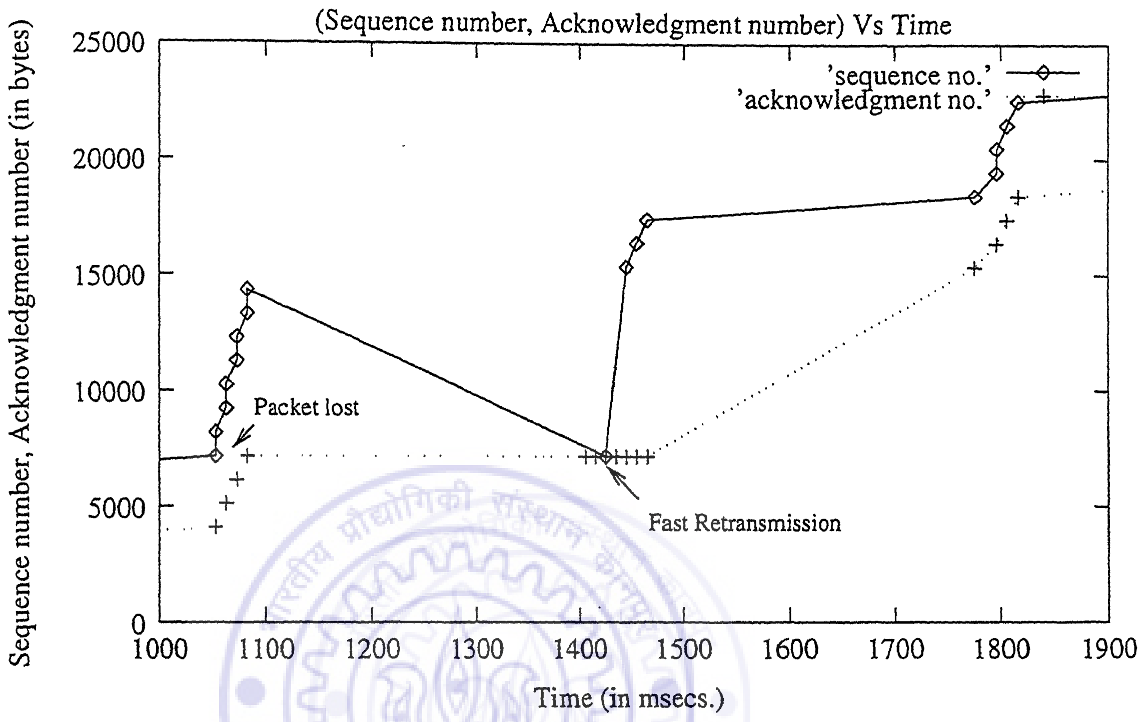

Figure 2.3: Plot of sequence number and acknowledgment number versus time

passes this and all the previously queued segments to the application. It generates an acknowledgment, acknowledging all the segments that have been received so far. This acknowledgment causes the source to come out of fast recovery algorithm. Source sets *cwnd* equal to *ssthresh* and proceeds as per the normal congestion avoidance algorithm.

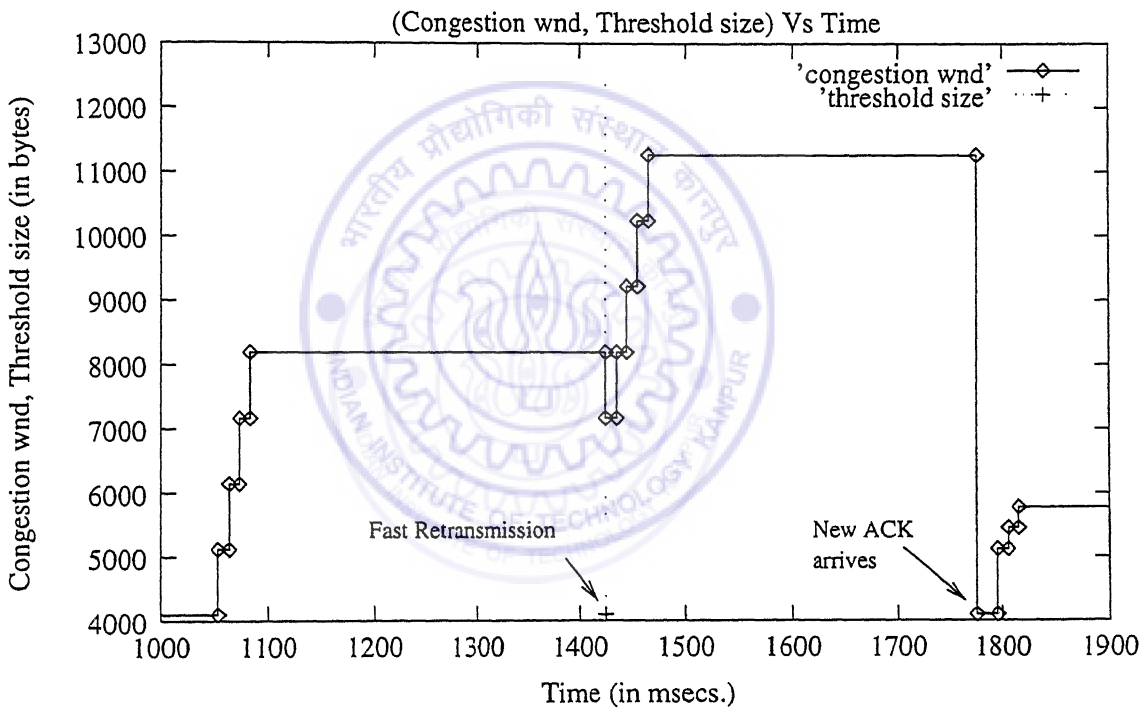

Figure 2.4: Plot of *cwnd* and *ssthresh* versus time

Chapter 3

Related Work

In this chapter, we summarize the solutions that have been proposed earlier to improve the performance of TCP over wireless links. We also point out the drawbacks of each of these solutions. For a detailed comparison, refer to [BPSK96].

3.1 Link-level retransmission [NED94, BBKT96]

This approach attempts to solve the problem by introducing reliability at the link layer for wireless link. The link layer uses a finite selective repeat protocol to recover from errors over the wireless link but it does not intend to completely eliminate all detectable errors. This approach attempts to make a wireless link appear as better quality link to the higher layers, at the cost of reduced effective bandwidth. The improved link quality results in fewer number of packet losses due to corruption on the wireless link and therefore, results in improved performance. This is primarily because detecting a packet loss at the (source) transport layer and retransmitting the packet is significantly costlier than sending packet at a lower rate, especially when packet loss rate is high.

This approach however has certain disadvantages :

- It may not be able to completely shield the source transport layer from all losses on the wireless link.

- Studies have shown that link-layer retransmissions may interfere with TCP's end-to-end retransmissions [DCY93]. So, we might have a scenario wherein a packet that was locally retransmitted some number of times and finally delivered across the wireless link, is retransmitted by the source TCP. This wastes efforts of the link layer, as it has to again put efforts in getting this re-transmitted packet through. Since, link layer as a resource, is shared by all TCP connections using the wireless link, wasted link layer efforts affect the performance of all TCP connections.

3.2 Split connection approach [BB95, YB94]

In this approach, the end-to-end TCP connection is split into two connections - one over the wired network and the other over the wireless link, with base station serving as the common point for two connections (A base station is a router that interfaces wireless link to the wired network). The advantage of this approach is that it isolates the source transport layer from the erratic behavior of the wireless link. It also allows quick recovery from errors over a wireless link.

This approach however, has the following disadvantages :

- The semantics of TCP connection are not preserved. The source TCP may receive an acknowledgment for a packet which hasn't reached its destination.

- Choice of using regular TCP over wireless link in [BB95] results in performance problems. Since TCP is not well-tuned for wireless links, TCP sender of the wireless link often times out causing the original sender to stall. This problem has been addressed in [YB94] wherein use of specialized transport protocol over wireless link is proposed.

- Every packet incurs the overhead of going through the TCP protocol processing twice at the base station. As the number of wireless links in the path increases, this overhead also increases.

- This approach requires the base station to maintain significant amount of state for every TCP connection passing through it.

3.3 Snoop protocol [BSAK95]

In this approach, a module called snoop agent is added to the routing code at the base station. This agent monitors packets flowing on a TCP connection and does intelligent processing, utilizing the knowledge of TCP. It maintains a cache of un-acknowledged TCP packets on a per-connection basis and performs local re-transmission when it detects a packet loss (either by arrival of a duplicate acknowledgment or by a local timeout). It intercepts duplicate acknowledgments, triggered by loss of a packet on a wireless link, and does not allow them to reach the source. This prevents the source from unnecessarily fast retransmitting resulting in improved performance.

However, this approach has the following disadvantages :

- Like the earlier link layer approach, this approach may not be able to completely shield the sender from losses on a wireless link.
- This approach assumes that a wireless link is the last hop in the network path. Though currently wireless links are most commonly used as a last hop link for providing portable computers access to the wired infrastructure, in the near future wireless links are expected to be widely used for connecting two islands of networks as well. Hence, any long term solution must cater for more general environment.
- It requires a base station to maintain significant TCP state information and a cache of unacknowledged packets for every TCP connection passing through it.
- Snoop agent performs some additional processing for every TCP data packet or acknowledgment packet that passes through. Since a base station is a common point for many TCP connections, the cumulative effect of this per-packet processing overhead may become significant and may make base station a bottleneck.
- The proposed solution is not symmetric. A base station has different set of policies for connection from fixed host to mobile host than for connection from mobile host to fixed host. This further increases the complexity of a base station.

Chapter 4

Simulator

Simulation techniques are widely used to study the performance of schemes in the area of computer networks. As part of this thesis work, we developed an event driven network simulator. In this chapter, we present the salient features of the simulator and briefly discuss its design.

4.1 Motivation

Many TCP simulators such as Real [Kes88], Netsim [Hey90], NS [MF95] are available in public domain. However, they have the following disadvantages :

- None of these simulators supports the concept of a wireless link.
- They have a large size and for the most part, they are undocumented.
- Bugs have also been reported in some of the simulators [Gup95].

Modifying undocumented code to introduce the concept of a wireless link was itself a big task and then further modifications were required to simulate and test the proposed schemes. The problem was compounded by the fact that we planned to iteratively evolve the scheme taking inputs from the simulation results. We perceived that the effort involved in effecting these modifications is more than the effort involved in developing our own simulator. Hence, we decided to develop our own simulator, more suited for the application at hand.

4.2 Features

The simulator was specifically aimed at simulating wireless computing environment, however, it supports more general environment involving wired and/or wireless links.

Simulator has the following features :

1. It is an event driven simulator.

2. **TCP**
   It simulates 4.3 BSD Reno implementation of TCP. The simulator only simulates the ESTABLISHED state of a TCP connection. The other states associated with connection establishment phase and connection tear-down phase, are not simulated. More specifically, slow-start, congestion avoidance, fast retransmit, and fast recovery algorithms of TCP, active during ESTABLISHED state, are simulated.

3. **Network Topology**
   Simulator supports arbitrary network topology. It reads the description of the network topology from a file. The description specifies the characteristics of various node and links involved in the simulation.

4. **Nodes supported**
   Various nodes supported by the Simulator are classified on the basis of their functionality into following 3 categories :

   - Source
   - Router
   - Sink

   Within each category, the nodes have been further classified on the basis of the kind of link layer protocol they implement over the wireless link : reliable[1] or un-reliable.

   - Source Node
     Nodes acting as the source of TCP segments are categorized as source nodes. Every source node in the network transmit TCP

[1] As we will see later in this chapter, the link layer protocol that we refer to as reliable is only semi-reliable.

segments to one and only one sink node. A node of this category may also be referred to as 'TCP sender'. Following nodes fall in this category :

  - `wired_source_node`
  - `wireless_source_node`

  `wireless_source_node` implements a reliable link layer protocol over the attached wireless link whereas `wired_source_node` implements an un-reliable link layer protocol.

- Router Node

  Nodes routing IP packets to their destination are categorized as router nodes. Following nodes fall in this category :

  - `normal_router_node`
  - `base_station`

  `base_station` implements a reliable link layer protocol over the wireless link. It also generates control information, if required. It is therefore, an important entity in our scheme. `normal_router_node` implements an un-reliable link layer protocol over the wireless link.

- Sink Node

  Nodes acting as destination of TCP segments are categorized as sink nodes. A sink node may accept TCP segments from one and only one source node. A node in this category may also be referred to as 'TCP receiver'. These nodes receive TCP segments and generate appropriate TCP acknowledgments. Following nodes fall in this category :

  - `wired_sink_node`
  - `wireless_sink_node`

  `wireless_sink_node` implements a reliable link layer protocol over the wireless link whereas `wired_sink_node` implements an unreliable link layer protocol.

5. **Links supported**

   Following two kinds of links are supported :

   - Wired link

Error characteristics of a wired link is modeled by uniform distribution. The packet loss probability over the link is specified by the user.

- Wireless link

  Error characteristics of a wireless link is modeled by 2-state Markov model [BBKT96]. At any given instant, we model the channel as being in one of the two possible states, GOOD or BAD. We assume that packet loss probability in GOOD state is 0 (channel is perfectly reliable), and that in BAD state is 1 (channel is perfectly lossy). We assume that the duration of stay in both the states is exponentially distributed, with different average values. These average values are specified by the user.

  Simulator supports following two models of wireless link :

  - Full duplex :
    In this model, a wireless link is assumed to be a point-to-point link between two nodes in the network, each transmitting at a different frequency. Also, it is assumed that the wireless link is not shared by any other entity in the network.
  - Shared link :
    In this model a wireless link is shared by all the wireless hosts and the base station, present in a single cell. Also, the error characteristics of a channel between the base station and a particular wireless host is independent of the error characteristics of channel between base station and any other wireless host in the same cell [BBKT96].
    Simulator supports IEEE 802.11 MAC protocol over shared wireless link model.

6. **Routing**
   Simulator supports static routing between the nodes. The routes between different pairs of nodes are calculated using Dijkstra's shortest path algorithm, with hop count as the cost metric. This algorithm is invoked by the IP layer of a node, when it receives an IP packet for which it doesn't have an entry in the routing table. This happens when this packet is the first one on a TCP connection.

7. **Trace Generation**
   Simulator generates trace of important parameters. It also generates a trace of important events taking place during simulation.

   The trace of each parameter is controlled by a control flag and may be turned off, if required.

   The trace of important events is controlled by another set of flags. These flags may be used to selectively allow/disallow some events from appearing in the trace. This is necessary as otherwise, the amount of data in the trace may be overwhelming.

8. **Scheduling**
   When more than one wireless host is present in a single cell, usually, there are multiple TCP connections sharing the wireless link. In such a scenario, base station maintains a separate queue (per-connection queue) for each wireless host. These per-connection queues are served in round-robin fashion.

   We opted for this model as it has been shown in [BBKT96] that FIFO scheduling of packets in conjunction with reliable link layer protocol leads to degraded overall throughput and unfair allocation of wireless bandwidth to wireless hosts, due to head-of-line blocking effect.

9. **Miscellaneous**
   Simulator can *use* a trace file to determine the time at which the wireless link should switch from GOOD state to BAD state and vice-versa, during the course of a simulation. This feature is especially useful when the desired behavior of a wireless link cannot be produced by a mathematical model.

   Simulator allows a user to specify the time at which a source should start transmitting packets.

4.3 Simulator Design

The simulator has been built in an incremental fashion. In the beginning, we started with the bare minimum requirements. Thereafter, we kept on adding features as per our requirements. As a result, at certain places, design is not "clean". All through the design and development effort, we have given preference to ease of understanding over efficiency of implementation so that the simulator may serve as a useful tool in future and may be modified as per requirements. C++ language was chosen for implementation so that repeated modifications to the design, resulting from added requirements, do not make it "messy".

4.3.1 Simulator Functional Units

In this section, we describe the design of basic functional units of the simulator. These functional units combine to implement various features supported by the simulator. Each of these functional units have been implemented as a class in C++.

1. Event Scheduler (`class Event_Queue`)

   Event scheduler is the most important component of the simulator. It keeps a linked list of events waiting to happen. The linked list is ordered by the time of occurrence of events.

   The scheduler allows scheduling an event at a particular instant of time, and canceling a previously scheduled event.

   An event node has the following fields :

   - `eve_type` (Event type)
   - `eve_time` (Event time)
   - `src_node_id` (Source node id)
   - `target_node_id` (Target node id)
   - `seq_no` (Sequence number)
   - `ack_no` (Acknowledgment number)

   Source node id contains the id of the node that has queued the event. Target node id contains the id of the node that will take action on the occurrence of this event. Together with sequence number and acknowledgment number, they are used to identify the TCP segment to be processed.

   Every node in the simulator implements its own event handler. When an event occurs, event scheduler passes the event to event handler of the target node.

2. TCP Source (`class TCP_SRC`)

   TCP Source is responsible for injecting TCP segments into the network and for processing TCP acknowledgments, as per TCP algorithms.

   Apart from the variables maintaining state of a TCP connection, `class TCP_SRC` maintains two queues :

   - `send_wnd` (Send window) :
     This is a queue of TCP segments that have not been acknowledged so far by the other end.

     When a TCP segment is sent, it is also queued in `send_wnd`. This queued segment is used in case the segment has to be retransmitted. When a TCP acknowledgment is received, all the segments that are acknowledged are removed from the `send_wnd`.

   - `tcp_in_queue` (TCP input queue) :
     All incoming TCP acknowledgments are queued in this queue before processing.

3. IP (`class IP`)

   IP is responsible for routing IP packets to their respective destination.

   `class IP` is used when we want to simulate an un-reliable model of link layer. Link layer in this model is a forwarding agent. It accepts IP packet from IP layer, encapsulates it in a frame and passes it on to physical layer for transmission. At the other end, link layer receives the frame from the physical layer, and passes the encapsulated IP packet to the IP layer. In our simulator, instead of modeling the link layer explicitly, we have chosen to assume its existence. We do this so that we simulate only to the detail required for our purpose. This results in some additional entries in the routing table which are traditionally not present.

   `class IP` has following three components :

   - `ip_in_queue` (IP input queue) :
     All incoming IP packets are queued here before being routed. The routing module takes a packet from this queue and using the routing table, routes it to appropriate next hop router.

- routing_table :
  A routing table has the following entries :
  - dest_id (Destination node id)
  - router_id (next hop router id)
  - link_ptr (link pointer)
  - link_busy (link busy flag)
  - transmission_completion_time (completion time of current transmission)

  link_ptr points to the characteristics of the link between the current node and the next hop router node (router_id). This is used to calculate transmission time of packet and to determine the status of the link.

  When a packet transmission is in progress, link_busy flag is turned ON and transmission_completion_time contains the time at which the transmission will complete.
- network_if_queue (Network interface queue) :
  Packet that is being transmitted but which has not yet reached the next hop router, is kept in this queue.

4. IP (modified) (class MOD_IP)

   class MOD_IP builds over the functionality of class IP. It implements reliable link layer model for wireless link. Link layer model for wired link remains the same (i.e. un-reliable).

   class MOD_IP adds one field, a link layer pointer, to the routing table. This is the only change in the components of the class IP. Link layer pointer (dll_ptr) is set when the connecting link (link_ptr) is wireless. Otherwise, it is NULL. dll_ptr is used for passing packets to the link layer.

5. Link layer

   Link layer is responsible for recovering from some of the errors over the wireless link through link layer retransmissions (also referred to as local retransmissions, as an analogy to end-to-end retransmissions of TCP). If N successive retransmission attempts for a frame have failed (for some value of N), link layer discards the frame and attempts to deliver next frame in the queue. In this sense, link layer is only "semi-reliable". It employs a stop and wait protocol analogous to (but not exactly similar to) one-bit sliding window protocol.

Simulator supports the following two models of link layer :

- The first model (implemented by `class MOD_DLL_WN`) supports the features mentioned above. This model is used as part of wireless sink node.
- The second model (implemented by `class MOD_DLL_BS`) supports following additional functionality :
  - It generates control information, if required.
  - It maintains per-connection queue for each of the TCP connection passing through (more precisely, for each of the wireless host present in the same cell with which communication is going on). This feature is of use to base station which normally communicates with more than one wireless host in its cell.

  This model is used as part of base station and wireless source node.

Link layer has the following components :

- `network_if_queue` (Network interface queue) :
  In `class MOD_DLL_WN`, this is simply a queue of frames. All IP packets routed and passed by the IP layer are encapsulated and stored in this queue.
  In `class MOD_DLL_BS`, this is a queue of per-connection states. A per-connection state contains the following information :
  - `per_connection_queue` :
    It is a queue of all IP packets routed and passed by the IP layer (similar to `network_if_queue` in `class MOD_DLL_WN`).
  - `no_of_attempts` (Number of attempts) :
    It maintains the number of attempts that have been previously made on transmitting the frame at the head of the `per_connection_queue`.

  - `next_frame_seq_no_expected` :
    It is used to determine whether frame received on this connection is duplicate or not.
  - `dst_node_id` (Destination node id) :
    It maintains id of the node at the other end of the connection. It is used to identify a connection.

- `dll_send_wnd` (Link layer's send window) :
  It contains frames that have been transmitted but have not been acknowledged by the link layer at the other end.
- `dll_in_queue` (Link layer's input queue) :
  Frames that have been received from the other end are queued in this queue, before processing.
- `nw_card_queue` (Network card's queue) :
  Frame that is being transmitted but which has not yet reached the other end, is kept in this queue.

6. TCP Sink (class `TCP_SINK`)

   TCP Sink is responsible for accepting TCP segments and generating appropriate TCP acknowledgments.

   It has the following components :

   - `tcp_in_queue` (TCP input queue) :
     All incoming TCP segments are placed in this queue before processing.
   - `recv_wnd` (Receive window) :
     This simulates TCP's reassembly queue. All out-of-order TCP segments are placed in this queue. This queue is kept sorted in the order of increasing (TCP) sequence number.
   - `seq_no_expected` (Sequence number expected) :
     It contains the sequence number of the segment that is expected next at the sink.

Chapter 5

Proposed Scheme

In this chapter we discuss our proposed scheme and present simulation results. At the end of this chapter, we list the advantages of our scheme.

5.1 Approach

All proposals discussed in Chapter 3, make the following assumptions :

- existing TCP implementations running on hosts *connected to wired networks* cannot be changed,
- changes can made to TCP (or any other software) running on hosts *connected to wireless networks*, and
- changes can be made to those hosts *which provide interface between wired and wireless networks.*

These assumptions are based on the fact that one cannot modify existing TCP implementations overnight, and a new TCP implementation that is incompatible with an existing one, is unacceptable. But these assumptions lead to very restrictive solutions, which are either too complex, or do not achieve very good performance. This is because they impose rigid requirements on the scheme. The only requirement should be that the new implementation be compatible with the older implementations. With this assumption, new implementations may be tuned to provide good performance over wireless networks. They also work correctly with the older implementations. With the passage of time, hosts running older version of TCP may be upgraded in an incremental fashion so that all hosts will be running the newer version eventually.

In this thesis, we propose to augment TCP to respond to control signals from wireless gateway nodes. The simulation results show that the scheme provides substantial performance improvement with minimal cost overhead.

5.2 Basic idea

The performance of TCP degrades because it (mis)interprets losses over wireless links as signs of congestion in the network. If TCP can distinguish a packet loss due to congestion from that due to noise on the channel, then algorithms can be developed to deal with the losses in an efficient fashion. In this thesis, we make the assumption that losses in the wired network are congestion-related, while those in the wireless network are noise-related. These are the same assumptions made by almost every researcher studying this problem, and follow directly from the error characteristics of these networks. They can be confirmed by studying losses on these networks [Jac88, YB94].

The problem now reduces to informing TCP whether the loss was in the wireless network or in the wired network, and to developing good heuristics to deal with the two situations. We can do this by modifying the *base station* (the node that has both wireless and wired interfaces).

One way to decrease the confusion between congestion losses and noise-related losses is to reduce noise-related losses. Several researchers [NED94, BBKT96] have suggested using link level retransmissions to achieve this. We agree with this suggestion because it isolates TCP to some extent, from the erratic behavior of the wireless link, and reduces the need for TCP retransmissions which are costly. But using only link level retransmissions may interfere with TCP's end-to-end retransmissions [DCY93]. Our proposal reduces this conflict by the exchange of additional control information.

5.3 General Framework

There are four components in any scheme based on the ideas discussed above. They are:

- What control information should be generated.
- The protocol to convey this information.
- When (and how frequently) should this information be generated.
- TCP heuristics to handle the information it receives.

Since, every control information represents an overhead in the network, it is crucial to minimize the frequency and total number of such control packets generated.

In the next section, we describe a scheme based on this framework and present simulation results. In subsequent sections, we refine and extend this scheme.

5.4 Simple ICMP scheme

5.4.1 Description of the scheme

In our scheme we are assuming that the wireless link is the last hop in the network path. In real life, most often the wireless link is indeed the last hop in the Internet. The most common scenario is a laptop connected to the Internet through a radio link.

For the purpose of discussion we assume that packets flow from a host on a wired network to a wireless host (host attached via a wireless link to the wired network).

We explain this scheme as a set of decisions we took corresponding to the four components of the general framework mentioned in the previous section.

▷ In TCP/IP network, ICMP messages have traditionally been used to carry control information. We propose to augment this protocol to carry control information as an additional type of message.

▷ Control information should allow TCP to identify the segment that was lost due to noise. For this, it should contain sufficient information to identify both the segment and the TCP connection to which it belongs. An ICMP error message contains just this information. As per the specification of ICMP [Pos81a], an ICMP error message contains an initial portion (the IP header and first 8 bytes of the data portion) of the IP packet that triggered it. For a TCP segment, these 8 bytes following the IP header will be from the TCP header. This portion of TCP header contains the source and destination port numbers and the sequence number. Thus, we achieve our objective of conveying segment sequence number and connection information by making the base station generate an ICMP error message.

We introduce a new type of ICMP error message for this purpose. We refer to this type of ICMP message as ICMP-DEFER message.

▷ The policy for generation of control information should satisfy the following three requirements :

1. An ICMP message should allow TCP to distinguish between a packet loss on a wireless link and a packet loss on the wired network.

2. An ICMP message should help TCP in avoiding conflict between local link layer retransmissions and end-to-end retransmissions.

3. Since, every ICMP message represents an overhead in the network, the total number of such ICMP messages generated should be minimized. This is a constraint on the policy.

The above three requirements may be mutually conflicting. For example, in order to satisfy the second requirement, the base station must generate an ICMP message fast enough to prevent TCP level retransmissions while link layer transmissions are going on. The first requirement dictates that base station should generate control information only when it has exhausted all its local retransmission attempts, so that TCP is sure that loss was on the wireless link. The third requirement means minimum number of control packets should be generated, but again too few packets may mean that TCP does not have complete information. Therefore, it is important to find a balance between these three requirements.

In our scheme, the base station generates an ICMP-DEFER message when the first local attempt at transmitting the packet has been unsuccessful. This policy ensures that within one round trip time, TCP will receive either an acknowledgment for the packet or an ICMP message. This minimizes the possibility of conflict between local link layer retransmissions and end-to-end retransmissions.

▷ An ICMP-DEFER message conveys the following information to the source TCP :

- The segment indicated in the message has reached the base station.
- The first attempt at transmitting the segment at the base station was unsuccessful. Since losses on wireless links occur in bursts, it is likely that all subsequent local retransmission attempts at the base station will also be unsuccessful. Hence, if TCP does not receive an acknowledgment, then, most probably, this segment was lost on the wireless link and the loss is unrelated to congestion.

TCP performs following actions on the receipt of an ICMP-DEFER message :

- If retransmit timer is set for the segment indicated in the ICMP message, it postpones its expiry by the current estimate of retransmission timeout (RTO) value. This is done to avoid any conflict between the local retransmissions at the base station and end-to-end retransmissions. Since, errors on a wireless link generally occur in burst, there is a high chance that a few subsequent local retransmission attempts at the base station will also fail. Hence, acknowledgment for this segment may get delayed. Therefore, it becomes important to postpone the retransmit timer.

  We assume that one RTO time is sufficient for the base station to exhaust all local retransmission attempts for a packet. This assumption is valid because wireless links typically have bandwidth of 1 Mbps, making transmission time over wireless links very small as compared to the value of RTO (minimum value is 1 second [WS96]).
- It stores the information that an ICMP-DEFER message was received for this segment.

If a segment needs to be retransmitted (either because of retransmit timeout or because of receipt of three duplicate acknowledgments), TCP checks if it has received an ICMP-DEFER message for this segment. If it hasn't received an ICMP-DEFER message, it follows normal TCP algorithm. If it has received ICMP-DEFER message, it performs one of the following actions depending on the reason for retransmitting the segment (retransmission timeout or fast retransmission) :

- Retransmission Timeout
  It simply retransmits the segment without changing the congestion window, *cwnd*, or slow start threshold size, *ssthresh*. It also discards the information that ICMP-DEFER message was received for the earlier transmission of this segment.

- Fast Retransmission
  It retransmits the segment without changing the *ssthresh* and discards the information that ICMP-DEFER message was received for the earlier transmission of this segment. It then follows the normal fast recovery algorithm viz. setting the *cwnd* to *ssthresh* + 3 * MSS, and incrementing *cwnd* by one MSS for every subsequent duplicate acknowledgment received and transmitting a new segment if allowed by the current window size (minimum of *cwnd* and receiver's advertised window size).

  When it comes out of fast recovery, it resets *cwnd* back to its value before fast retransmission and fast recovery.

5.4.2 Simulation results

Simulation Setup

During our simulations, we have used the following network topology.

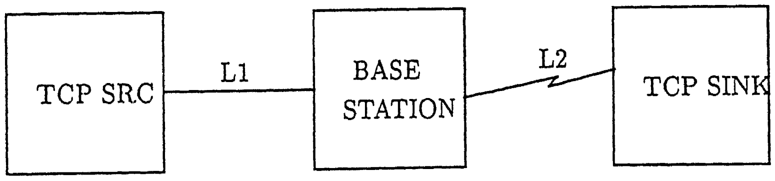

Figure 5.1: Network configuration used in Simulations

The wired link in the topology is representative of the entire network path over the wired network. The propagation delay of this link represents the sum of propagation delay, queuing delay and the processing time of each node in the entire path over the wired network. We assume that the wired link is perfectly reliable. We further assume that the base station does not drop packets because of insufficient buffers.

We simulated wireless link by a 2-state Markov model with the stay period in each state characterized by exponential distribution [BBKT96]. The propagation delay of the wireless link is representative of the processing time at the nodes as well.

Parameters

The following is the list of parameters that are of interest in the simulation.

- Wired link :
  - Bandwidth
  - Propagation delay
  - Loss probability
- Wireless link :
  - Bandwidth
  - Propagation delay
  - Error characteristics [BBKT96]
    - GOOD state
      - mean value of exponential distribution
      - loss probability
    - BAD state
      - mean value of exponential distribution
      - loss probability
- Packet size
- Size of file transfer
- Receiver's advertised window size

We vary receiver's advertised window size, propagation delay of the wired link, and error characteristics of the wireless link in our simulations. The remaining parameters are kept fixed.

Varying the receiver's advertised window size allow us to vary the maximum amount of data that may be in-transit.

Varying the propagation delay of the wired link allow us to simulate various kinds of networks (LAN, WAN, etc.). A WAN can be simulated by keeping the propagation delay value large. A small value simulates a LAN.

Varying the mean value of exponential distribution for GOOD and BAD states allow us to simulate wireless link of different error characteristics.

The Results

Table 5.1 shows the values of various parameters that we used in the simulation.

The propagation delay is set to 492 milli-seconds to highlight the effects of the clash between local link layer retransmissions and end-to-end retransmissions.

The minimum value of RTO is 1 second. We try to maintain the actual round trip time (RTT) (twice the propagation delay of wired link and the propagation delay of the wireless link) very close to the minimum RTO (1 second) by keeping propagation delay at 492 milli-seconds. This makes TCP less tolerant to fluctuations in RTT because of local link layer retransmissions at the base station.

We are interested in the time taken for completion of the transfer. This gives us the TCP-level throughput.

The simulation results are summarized in the bar graph shown in figure 5.2. It shows the time taken for completion of the simulation run for various schemes for different values of receiver's advertised window size. The schemes being compared are : Unmodified TCP for the case when wireless link is perfectly reliable (*No loss case*), Unmodified TCP using normal wireless link assumptions (*Unmodified TCP*), and TCP enhanced based on our scheme (*Simple ICMP scheme*). The graph also shows the percentage performance improvement of *Simple ICMP scheme* over the *Unmodified TCP*.

In the figure, `mean_bad_state` refers to the mean value of exponential distribution for BAD state for the model of wireless link.

From the graph we can see that there is degradation in performance of *Unmodified TCP* and *Simple ICMP scheme* over the *No loss case*. Further,

Parameters	Value
Wired link	
- Bandwidth	8 Mbps
- Propagation delay	492 milli-seconds
- Loss probability	0 (perfectly reliable)
Wireless link	
- Bandwidth	1 Mbps
- Propagation delay	1 milli-seconds
- Error characteristics [BBKT96] -	
* GOOD state	
- mean value of exponential distribution	5 seconds
- loss probability	0 (perfectly reliable)
* BAD state	
- mean value of exponential distribution	0.1 second
- loss probability	1
Packet size	1 KB
Size of file transfer	10 MB
Receiver's advertised window size	16 to 64 KB

Table 5.1: Table showing parameter values used in the simulation (figure 5.2)

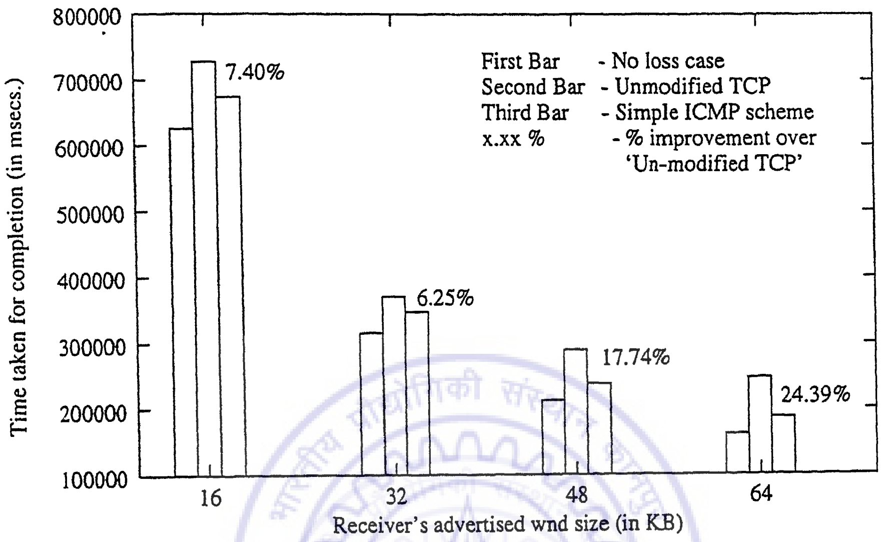

Figure 5.2: Propagation delay of wired link kept at 492 milli-seconds and `mean_bad_state` kept at 0.1 second

we see that the degradation in performance of *Unmodified TCP* increases rapidly with the increase in receiver's advertised window size. This is because it treats losses on the wireless link as signs of congestion and reduces its *cwnd* and *ssthresh*. The effect of this reduction becomes more prominent at larger value of receiver's advertised window size. We see that the performance of *Simple ICMP scheme* does not suffer much degradation. The degradation results from having to retransmit the segments lost on the wireless link. As the figure shows, the percentage improvement of *Simple ICMP scheme* over *Unmodified TCP* reaches 24.39 %.

5.4.3 Discussion

In *Simple ICMP scheme*, TCP reacts to a packet loss on a wireless link by retransmitting the packet while maintaining the same sending rate (i.e. it does not reduce its *cwnd* and *ssthresh*). This results in substantial performance improvement. But, it also introduces a problem. For every packet lost on the wireles link, TCP now recovers it by suffering either a fast retransmit or a retransmission timeout. Since, fast retransmit and fast recovery algorithms were designed to recover from single packet-loss, in case when multiple packets in a transmission window are lost on the wireless link, TCP recovers them by suffering retransmission timeouts as many times as the number of losses. This is in contrast to unmodified TCP which recovers from multiple consecutive packet losses in a window by suffering at most one retransmission timeout.

Illustration of the problem in Simple ICMP scheme

As an illustration of the problem, consider the graph shown in figure 5.3. The graph shows the reaction of TCP, based on *Simple ICMP scheme*, to multiple consecutive packet losses in a window. For this example, the receiver's advertised window size was kept at 4 KB and propagation delay of wired link was kept at 170 milli-seconds (the remaining parameters' values being the same as mentioned in table 5.1).

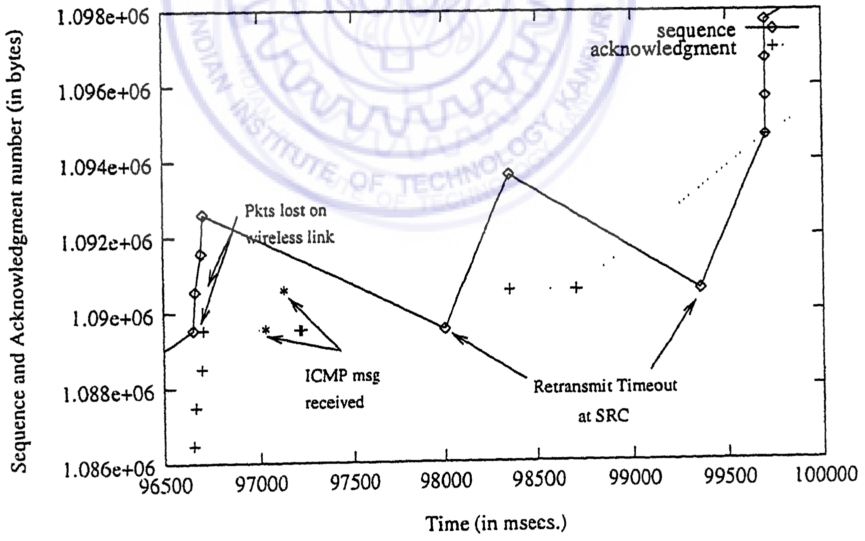

Figure 5.3: Graph illustrating problem with ICMP scheme

As indicated in the figure, two consecutive packets are lost on the wireless link. The two "*" marks on the graph indicate the time at which ICMP-DEFER messages were received by TCP, corresponding to these packets.

The remaining two packets successfully reach the destination and results in two closely spaced acknowledgments (at 97211 and 97221 milli-seconds). At 98003 milli-seconds, TCP suffers a retransmission timeout and retransmits the first lost packet. Since, it had received an ICMP-DEFER message for this packet, it simply resends the packet without reducing its *cwnd* and *ssthresh*. This packet successfully reaches the destination. The acknowledgment for this packet is received by source TCP at 98354 milli-seconds. This acknowledgment allows source TCP to transmit a new packet. When the destination receives this new packet, it generates a duplicate acknowledgment indicating the next sequence number expected. Since no more packets are traveling in the network, no more acknowledgments are received. TCP finally times out at 99354 milli-seconds and retransmits the second packet lost on the wireless link. Since, it had received an ICMP-DEFER message for this packet as well, it refrains from reducing its *cwnd* and *ssthresh*.

Thus, in this case, TCP recovered from two packet losses by suffering two retransmit timeouts. It took 3054 milli-seconds (from 96651 to 99705 milli-seconds) to recover from these two losses.

The graph shown in figure 5.4, illustrates the reaction of unmodified TCP under similar situation. As indicated in the figure, two consecutive packets are lost on the wireless link. The remaining two packets manage to reach the destination and results in two closely spaced acknowledgments (at 97211 and 97221 milli-seconds). At 97703 milli-seconds, TCP suffers a retransmission timeout and retransmits the first lost packet. TCP reduces its *cwnd* and *ssthresh* as per the standard algorithm (figure 5.5). It also sets the next packet to be transmitted to the one immediately following this packet. When the destination receives this packet, it generates an acknowledgment indicating the next packet expected (next packet expected is the second packet lost on the wireless link). This acknowledgment reaches the source TCP at 98054 milli-seconds. On receiving this acknowledgment, source TCP increases its *cwnd* by 1 MSS and sends out two packets. The first of these two packets fills the hole at destination. Destination sends an acknowledgment, acknowledging all the packets sent till now. This acknowledgment is received by the source TCP at 98405 milli-seconds.

Note that in this case, TCP recovered from 2 packet losses by suffering a single retransmit timeout. It took 1754 milli-seconds (between 96651 milli-seconds and 98405 milli-seconds) to recover from the two packet losses. This is almost half of the time required to recover in *Simple ICMP scheme*.

In Simple ICMP scheme, this large delay in recovering from multiple

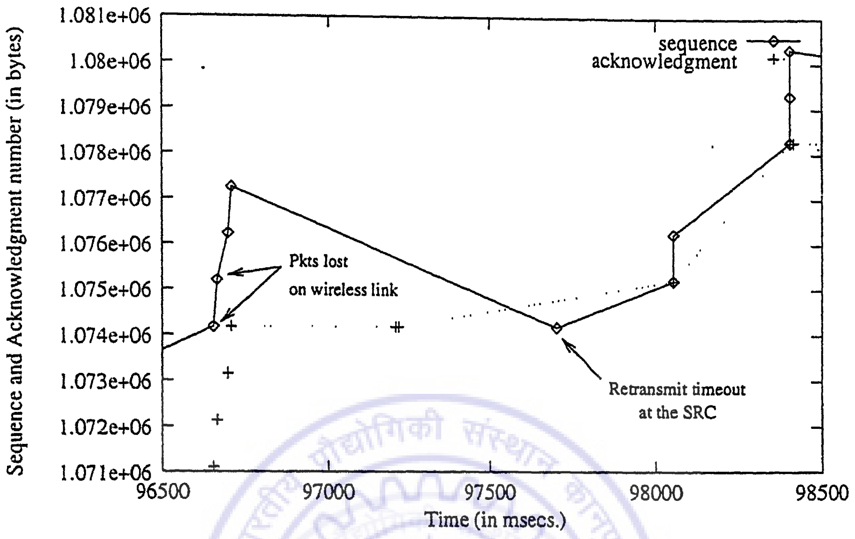

Figure 5.4: Graph illustrating the response of (unmodified) TCP to multiple packet losses

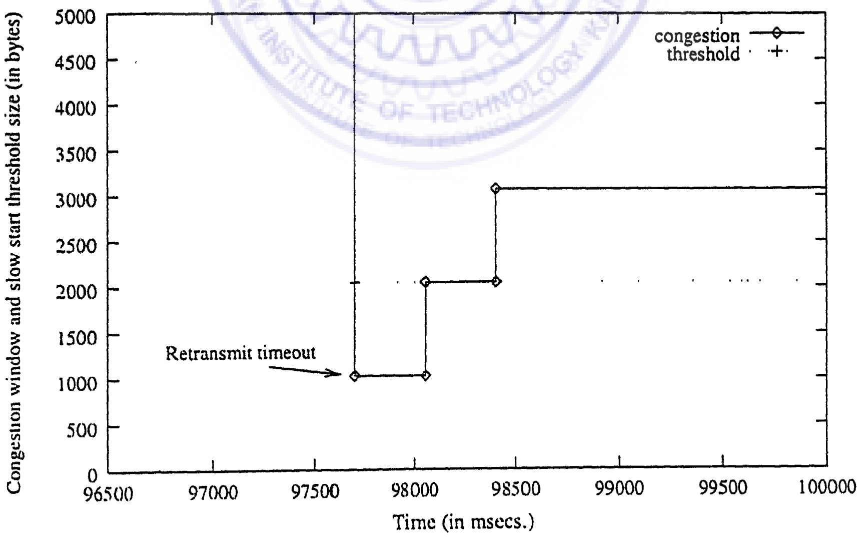

Figure 5.5: Graph showing plot of *cwnd* and *ssthresh* versus time

packet losses, detracts the advantage gained by not reducing the cwnd and ssthresh. Note that this situation is not specific to a particular value of receiver's advertised window size. With larger window size as well, similar pattern was observed (though, the frequency of such patterns does depend on the value of the receiver's advertised window size). With a larger window size, it is possible that a packet loss is recovered by fast retransmit algorithm. But, since fast retransmit and fast recovery algorithms were designed to recover from a single packet loss, if we have multiple packet losses in the same window, we end up with a similar scenario. Figure 5.6 illustrates one such situation with receiver's advertised window size of 96 KB.

The performance improvement gained by *Simple ICMP scheme* then depends on how many times such multiple packet losses occur in a transfer. Simulation with receiver's advertised window size of 4 KB showed that the use of ICMP messages, in fact, resulted in degraded performance! While *Unmodified TCP* took 929824 milli-seconds to complete the transfer, *Simple ICMP scheme* took 945579 milli-seconds to complete the transfer. A closer look reveals that, during the course of transfer, *Unmodified TCP* suffered 74 packet losses on the wireless link. It recovered them with 26 fast retransmits and 19 retransmit timeouts. On the other hand, *Simple ICMP scheme* suffered 67 packet losses on the wireless link during the transfer. It recovered them with 23 Fast Retransmits and a staggering 42 **retransmit timeouts**.

5.5 Refined ICMP scheme

In this section, we identify the problem with simple ICMP scheme and propose modifications to the scheme for better performance.

5.5.1 Problem with Simple ICMP scheme

The main problem with *Simple ICMP scheme* is that the loss detection latency is very high. With the information contained in an ICMP-DEFER message, source TCP can at most avoid a conflict between local and end-to-end retransmissions. When retransmitting the segment, it can refrain from reducing *cwnd* and *ssthresh*. But, it has to still depend on a retransmission timeout or receipt of three or more duplicate acknowledgments to determine that a segment has been lost. This is because of our policy of generating

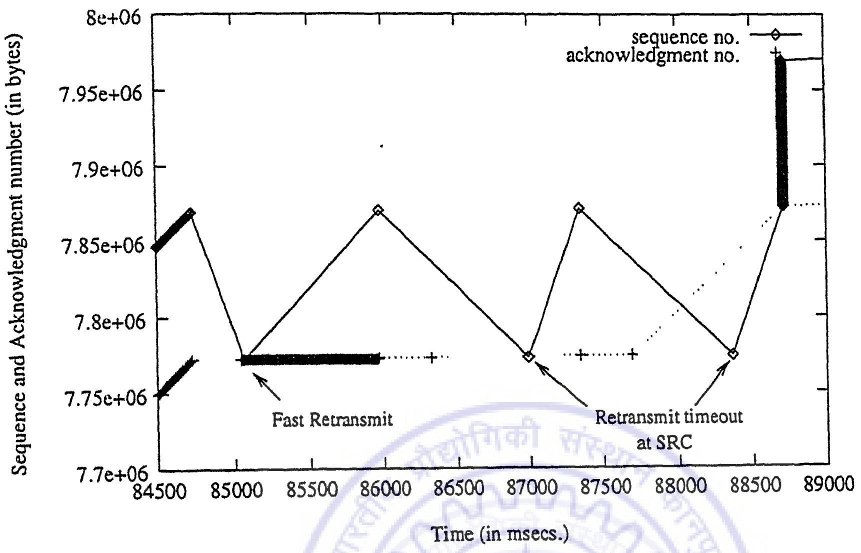

Figure 5.6: Graph illustrating problem with ICMP scheme (receiver's advertised window kept at 96 KB)

ICMP-DEFER message immediately after the first unsuccessful local transmission attempt at the base station.

5.5.2 Modifications to the original scheme

In order to overcome the above mentioned problem, we modify our policy that deals with when the base station should generate an ICMP message and the policy that deals with how the source TCP should respond to control information from the network.

We make the base station generate an ICMP error message when all the local retransmission attempts at the base station have been exhausted. We refer to this type of ICMP message as ICMP-RETRANSMIT message.

The TCP's response to control information is summarized in the following steps :

1. When the TCP receives an ICMP-DEFER message, it checks if the retransmission timer is set for the segment indicated. If so, it postpones the expiry of retransmission timer by the current estimate of RTO. This step is same as the one we followed in *Simple ICMP scheme.* But in this scheme, the TCP *does not store* the information that an ICMP-DEFER message was received.

2. When TCP receives an ICMP-RETRANSMIT message, it retransmits the segment indicated. It also stores the information that the ICMP-RETRANSMIT message was received for this segment.

3. An ICMP-RETRANSMIT message indicates that one segment was lost from the chain of segments sent by the source TCP. When the destination receives subsequent packets, it generates duplicate acknowledgments. When the source TCP receives the first of such duplicate acknowledgments, it switches to fast recovery algorithm. It discards the information that it received an ICMP-RETRANSMIT message for this segment. For every duplicate acknowledgment received, it increments the *cwnd* by one MSS and sends a new segment if allowed by current window size (minimum of *cwnd* and receiver's advertised window size). When it finally receives a new acknowledgment, it comes out of the fast recovery algorithm and resets *cwnd* to the value prior to its entering the fast recovery phase.

4. When source TCP suffers a retransmit timeout, it follows normal TCP algorithms.

5. When source TCP receives duplicate acknowledgments for a segment for which it has not received ICMP-RETRANSMIT message earlier, it follows normal TCP algorithms.

Note that the modified reaction to ICMP-DEFER message enhances the scheme to work on networks with a wireless link anywhere in the network path (and not necessarily the last hop). This is because, an ICMP-DEFER message only indicates that a segment has reached the base station. By storing this information (in our *Simple ICMP scheme*), we made the implicit assumption that if the source TCP does not receive an acknowledgment for this segment, then this segment was definitely lost on the wireless link and that this loss is unrelated to congestion. This assumption will be true only when wireless link is the last hop in the network path. Thus, by not storing the information that an ICMP-DEFER message was received, we avoid making such an assumption.

5.5.3 Simulation results

The following graphs summarize the simulation results for different values of propagation delay of the wired link and for different error characteristics of the wireless link (the remaining parameters' value being the same as mentioned in table 5.1).

Each of the bar charts below, show the time taken for completion against the receiver's advertised window size for the following schemes : *No Loss Case*, *Unmodified TCP*, *Simple ICMP scheme*, and TCP based on the refined ICMP scheme (*Refined ICMP scheme*).

The simulation results shown in figure 5.7 were generated using the parameters' value given in table 5.2. The figure also shows the percentage improvement in performance of *Simple ICMP scheme* and *Refined ICMP scheme* over *Unmodified TCP* case.

We see that the performance of *Refined ICMP scheme* is better than that of *Simple ICMP scheme* for all values of receiver's advertised window size. Its performance is also very close to that of the ideal *No loss case.* As the figure shows, the percentage improvement of *Refined ICMP scheme* over *Unmodified TCP* reaches 28.96 %.

Parameters	Value
Propagation delay of wired link	492 milli-seconds
Error characteristics of wireless link	
- GOOD state	
- mean value of exponential distribution	5 seconds
- loss probability	0
- BAD state	
- mean value of exponential distribution	0.1 second
- loss probability	1

Table 5.2: Table showing parameter values used in the simulation (figure 5.7)

The remaining simulation results are summarized in figures 5.8 to 5.16. Table 5.3 gives the range of parameters' value used in simulations. The figures also show the *percentage degradation* in performance of each scheme with respect to the ideal *No Loss Case*. This performance parameter tells us how far is the performance of each of these schemes, from the ideal *No Loss Case*. It also conveys an estimate of how the schemes fair with respect to each other. We also mention the percentage improvement of the proposed schemes over the *Unmodified TCP* case, in a table along with every bar chart.

We see from the simulation results (figure 5.8 to 5.16) that the performance of *Refined ICMP scheme* is better than that of *Simple ICMP scheme* and *Unmodified TCP*, for all values of receiver's advertised window size. The performance of *Refined ICMP scheme* is also close to the ideal *No loss case* (the percentage degradation being in the range from 2.45% to 24.88%). The higher percentage degradation in performance of *Refined ICMP scheme* occur when the value of `mean_bad_state` is large. This is because at higher values of `mean_bad_state` the burst periods are larger, resulting in more packet losses. The degradation results from having to retransmit these lost packets. However, it is to be noted that, in each of the case, the value of percentage degradation is very much better than the corresponding value obtained for *Unmodified TCP* and *Simple ICMP scheme*. As the figures show, the percentage improvement in performance of *Refined ICMP scheme* over *Unmodified TCP* case reaches 29.77 %.

Note that when `mean_bad_state` value is 0.1 second, the percentage im-

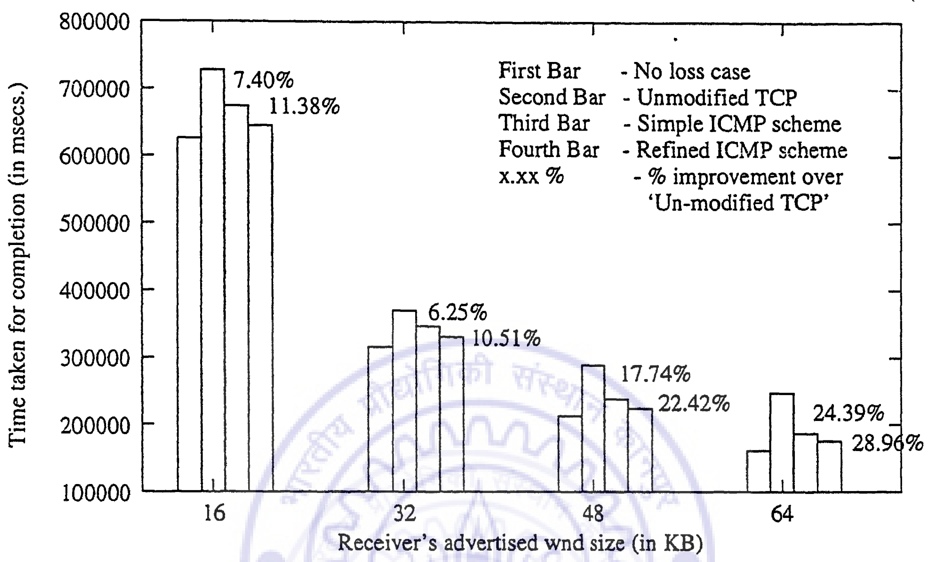

Figure 5.7: Propagation delay of wired link kept at 492 milli-seconds and `mean_bad_state` kept at 0.1 second

Parameters	Value
Propagation delay of wired link	50 to 100 milli-seconds
Error characteristics of wireless link	
- GOOD state	
- mean value of exponential distribution	5 seconds
- loss probability	0
- BAD state	
- mean value of exponential distribution	0.1 to 0.5 second
- loss probability	1

Table 5.3: Table showing parameter values used in the simulation (figure 5.8 to 5.16)

provement of *Refined ICMP scheme* over *Unmodified TCP* case is not much. But, a look at the corresponding value of percentage degradation shows that the performance of *Refined ICMP scheme* in such cases, is already very close to the ideal *No loss case.* For eg., table 5.4 shows that the percentage improvement of *Refined ICMP scheme* is in the range 3.78 % to 8.04 %. The corresponding value of preformance degradation (figure 5.8) is very low and is in the range 2.45 % to 6.35 %.

The performance of *Simple ICMP scheme* is, many a times, worse than the performance of *Unmodified TCP*. This happens especially at higher values of `mean_bad_state` and when receiver's advertised window size is 4 KB. At these values, *Simple ICMP scheme* recovers maximum number of losses through retransmit timeout, which involves considerable waiting time.

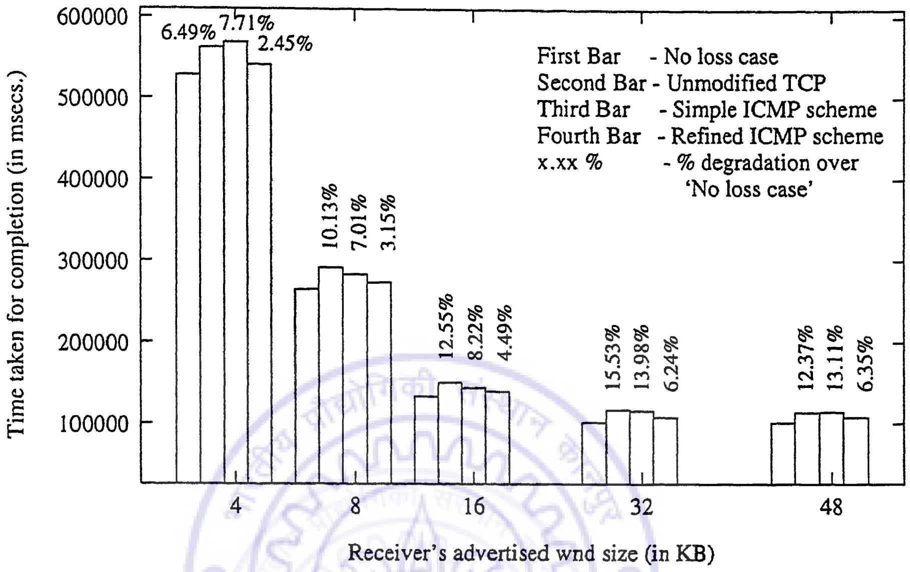

Figure 5.8: Propagation delay of wired link kept at 100 milli-seconds and `mean_bad_state` kept at 0.1 second

Receiver's advt. window size	% Improvement over *Un-modified TCP* case	
	Simple ICMP scheme	*Refined ICMP scheme*
4	-1.14 %	3.78 %
8	2.83 %	6.34 %
16	3.84 %	7.16 %
32	1.34 %	8.04 %
48	-0.65 %	5.35 %

Table 5.4: Table showing percentage improvement of the proposed schemes over *Unmodified TCP* case (figure 5.8)

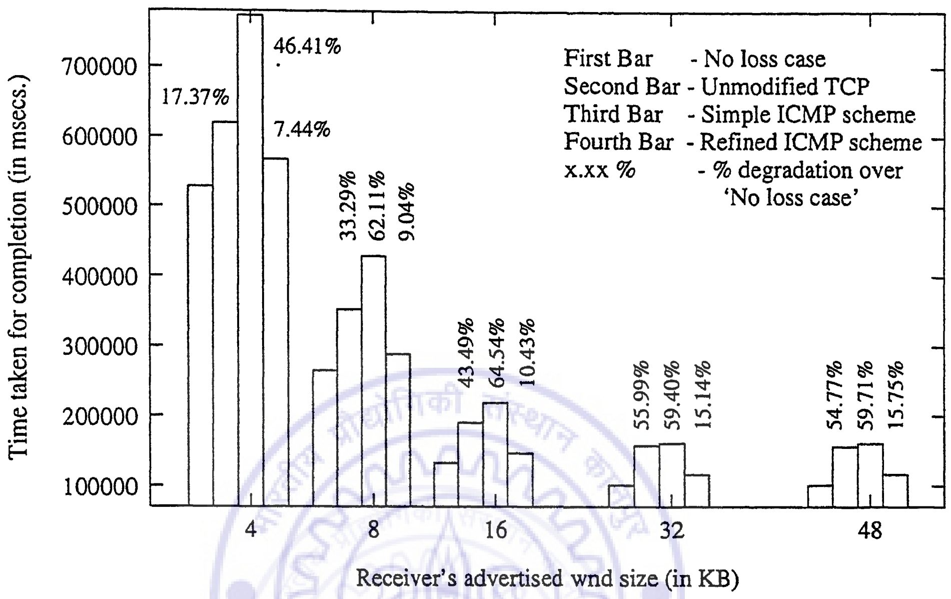

Figure 5.9: Propagation delay of wired link kept at 100 milli-seconds and `mean_bad_state` kept at 0.3 second

Receiver's advt. window size	% Improvement over *Un-modified TCP* case	
	Simple ICMP scheme	*Refined ICMP scheme*
4	-24.73 %	8.46 %
8	-21.62 %	18.18 %
16	-14.66 %	23.04 %
32	-2.18 %	26.18 %
48	-3.19 %	25.2 %

Table 5.5: Table showing percentage improvement of the proposed schemes over *Unmodified TCP* case (figure 5.9)

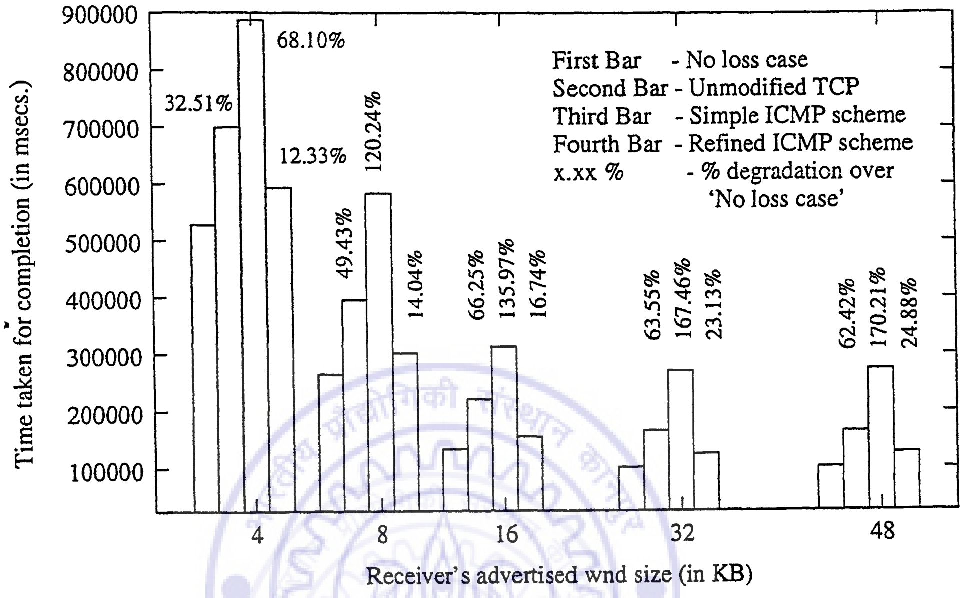

Figure 5.10: Propagation delay of wired link kept at 100 milli-seconds and mean_bad_state kept at 0.5 second

Receiver's advt. window size	% Improvement over *Un-modified TCP* case	
	Simple ICMP scheme	*Refined ICMP scheme*
4	-26.86 %	15.22 %
8	-47.53 %	23.68 %
16	-41.93 %	29.77 %
32	-63.52 %	24.71 %
48	-66.36 %	23.11 %

Table 5.6: Table showing percentage improvement of the proposed schemes over *Unmodified TCP* case (figure 5.10)

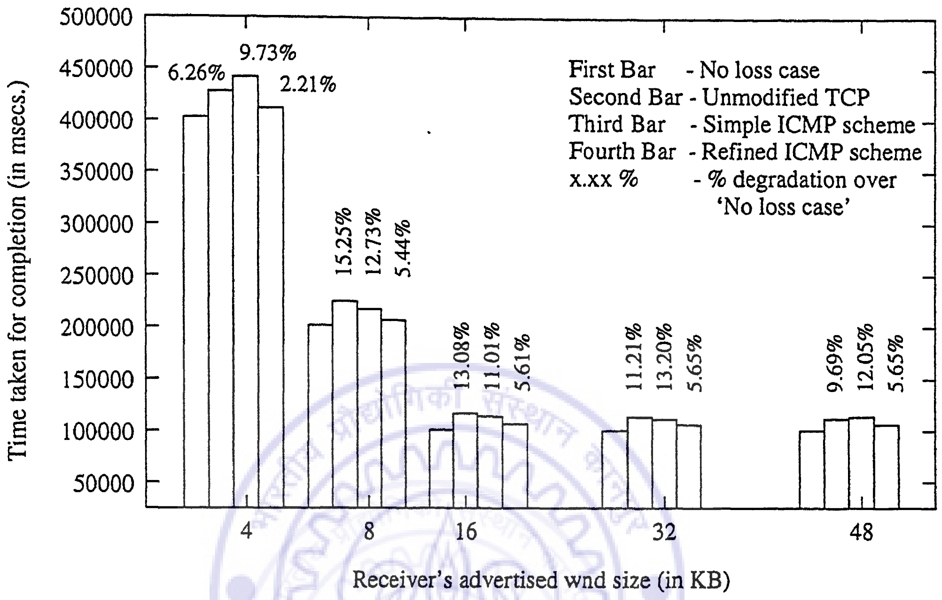

Figure 5.11: Propagation delay of wired link kept at 75 milli-seconds and mean_bad_state kept at 0.1 second

Receiver's advt. window size	% Improvement over *Un-modified TCP* case	
	Simple ICMP scheme	*Refined ICMP scheme*
4	-3.26 %	3.81 %
8	3.34 %	8.00 %
16	2.18 %	8.50 %
32	1.83 %	6.61 %
48	-1.17 %	5.06 %

Table 5.7: Table showing percentage improvement of the proposed schemes over *Unmodified TCP* case (figure 5.11)

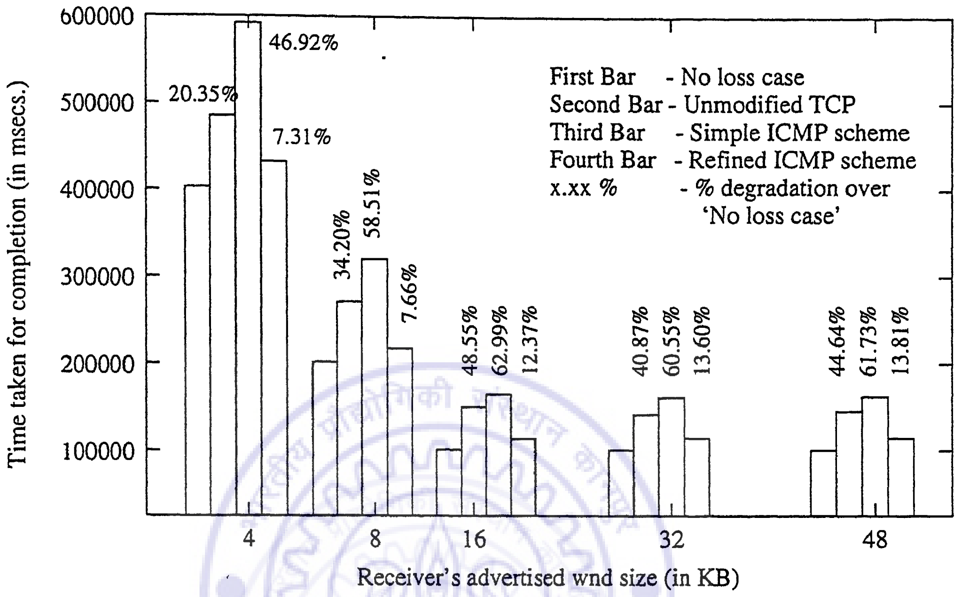

Figure 5.12: Propagation delay of wired link kept at 75 milli-seconds and `mean_bad_state` kept at 0.3 second

Receiver's advt. window size	% Improvement over *Un-modified TCP* case	
	Simple ICMP scheme	*Refined ICMP scheme*
4	-22.07 %	10.82 %
8	-18.10 %	19.77 %
16	-9.71 %	24.35 %
32	-13.97 %	19.35 %
48	-11.81 %	21.31 %

Table 5.8: Table showing percentage improvement of the proposed schemes over *Unmodified TCP* case (figure 5.12)

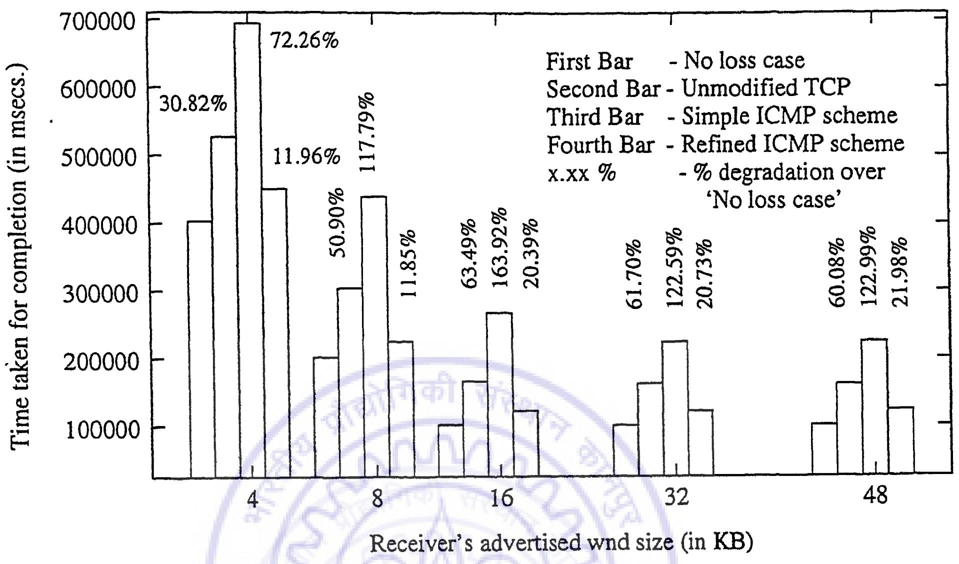

Figure 5.13: Propagation delay of wired link kept at 75 milli-seconds and `mean_bad_state` kept at 0.5 second

Receiver's advt. window size	% Improvement over *Un-modified TCP* case	
	Simple ICMP scheme	*Refined ICMP scheme*
4	-31.67 %	14.41 %
8	-44.32 %	25.87 %
16	-61.42 %	26.36 %
32	-37.65 %	25.33 %
48	-39.29 %	23.79 %

Table 5.9: Table showing percentage improvement of the proposed schemes over *Unmodified TCP* case (figure 5.13)

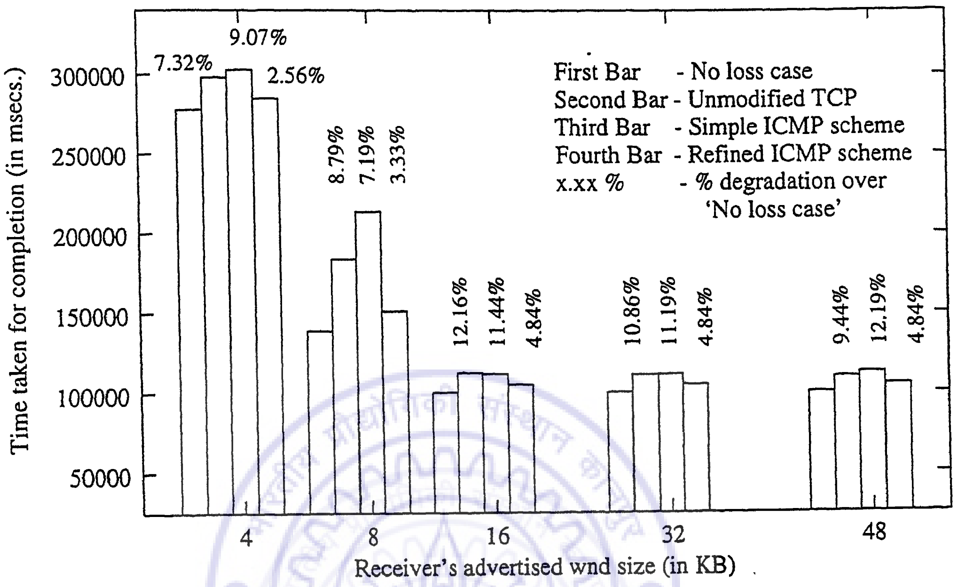

Figure 5.14: Propagation delay of wired link kept at 50 milli-seconds and `mean_bad_state` kept at 0.1 second

Receiver's advt. window size	% Improvement over *Un-modified TCP* case	
	Simple ICMP scheme	*Refined ICMP scheme*
4	-1.63 %	4.43 %
8	1.46 %	5.01 %
16	0.64 %	6.52 %
32	- 0.29 %	5.42 %
48	-2.50 %	4.20 %

Table 5.10: Table showing percentage improvement of the proposed schemes over *Unmodified TCP* case (figure 5.14)

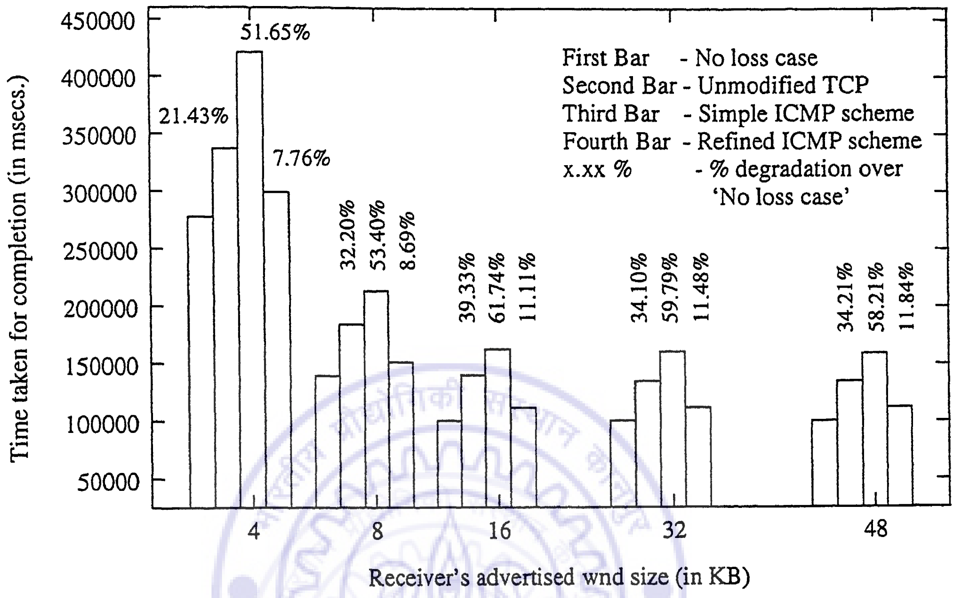

Figure 5.15: Propagation delay of wired link kept at 50 milli-seconds and mean_bad_state kept at 0.3 second

Receiver's advt. window size	% Improvement over *Un-modified TCP* case	
	Simple ICMP scheme	*Refined ICMP scheme*
4	-24.88 %	11.25 %
8	-16.03 %	17.78 %
16	-16.08 %	20.20 %
32	-19.16 %	16.86 %
48	-17.88 %	16.66 %

Table 5.11: Table showing percentage improvement of the proposed schemes over *Unmodified TCP* case (figure 5.15)

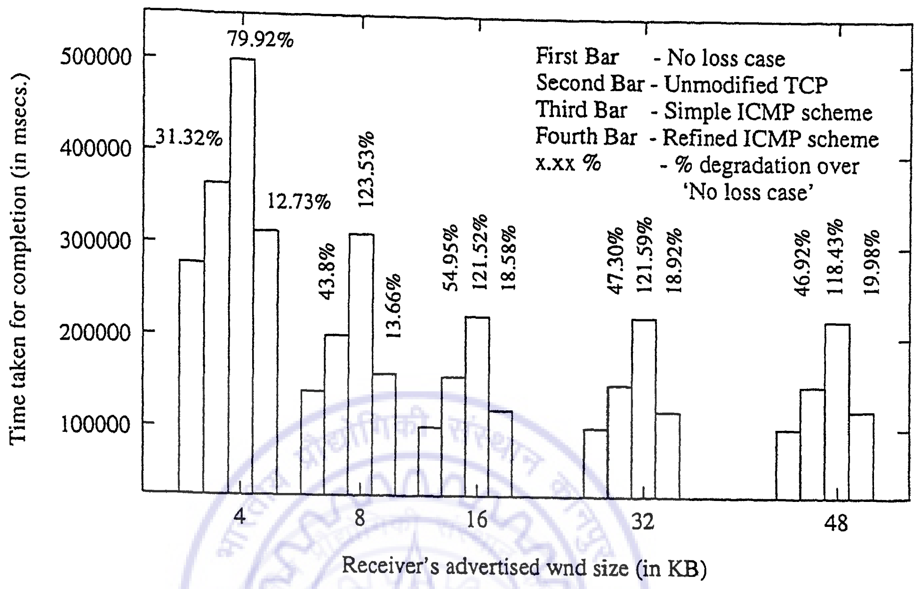

Figure 5.16: Propagation delay of wired link kept at 50 milli-seconds and `mean_bad_state` kept at 0.5 second

Receiver's advt. window size	% Improvement over *Un-modified TCP* case	
	Simple ICMP scheme	*Refined ICMP scheme*
4	-37.00 %	14.15 %
8	-55.45 %	20.95 %
16	-42.96 %	23.47 %
32	-50.43 %	19.26 %
48	-48.67 %	18.33 %

Table 5.12: Table showing percentage improvement of the proposed schemes over *Unmodified TCP* case (figure 5.16)

5.5.4 Advantages of Refined ICMP scheme

Below, we summarize the advantages of our scheme.

- Enhanced TCP, based on our scheme, performs very well over network paths involving wireless link, significantly improving the performance of TCP. Also, its performance over pure wired network and its ability to communicate with other TCP compliant hosts on the network is not affected because of enhancement.

- Overhead at the base station is minimal. The overhead of ICMP messages on the network is also very low. During simulation runs involving a 10MB file transfer (10,000 packets transfer), 19 to 552 ICMP messages were generated (for different values of propagation delay and for different values of receiver's advertised window size). This is 0.19 % to 5.52 % of the total packets sent.

- ICMP messages do not create any problem even if the source is not running enhanced TCP based on our scheme. These messages are silently discarded.

- Our scheme preserves the semantics of TCP, unlike the 'Split connection' approach.

- In our scheme, generation of ICMP messages is triggered by the link layer for the wireless link. So, in the case when a wireless host (host connected via wireless link to the wired network infrastructure) is acting as the source of packets, ICMP messages will be triggered by the link layer of the wireless host. This policy makes our scheme symmetric. It does not require any additional handling when a wireless host acts as the source.

- The scheme does not make any assumption about the location of the wireless link in the network path. We believe that there will be no change in performance if a wireless link is an intermediate link, but we have not studied it in detail.

Chapter 6

Conclusion and future work

In this chapter, we briefly summarize the contributions of this thesis work and discuss possible future extensions.

6.1 Summary

In this thesis work, we have proposed an ICMP based scheme to improve the performance of TCP over network paths involving wireless links. In our scheme, we have proposed enhancements to TCP to respond to certain control information from the network. This enables TCP to differentiate between losses on a wireless link and those on the wired network, resulting in significant improvement in performance. In our scheme, the control information is generated by the base station, and is carried to the source using new message-types in ICMP. The overhead of our scheme is also minimal.

As part of this thesis work, we have developed a network simulator for evaluating the performance of the proposed scheme in a wireless computing environment. The simulation results show that the proposed *Refined ICMP scheme* achieves upto about 30 % improvement in performance (measured in terms of the time take for completion of simulated data transfer) over the *Unmodified TCP* case. The results also show that the scheme is significantly more robust in face of burst errors on the wireless link and its performance gain relative to the *Unmodified TCP* improves with the increase in frequency of burst errors on the wireless link. It is also able to maintain its performance close to the ideal *No loss case.*

6.2 Future Work

6.2.1 Possible extensions to the proposed scheme

Following extensions may be made to the proposed scheme.

- In this thesis, we have concentrated on studying the behavior of the proposed scheme on a single, unilateral connection. Simulations of multiple two-way connections would provide more insights and would help in refining the proposed scheme.
- It would be interesting to study and optimize the performance of the proposed scheme over network paths involving multiple wireless links.
- A simulated environment represents a controlled environment, lacking randomness inherent in the real networks. It would be interesting to implement the proposed scheme on a wireless testbed and study its behavior and performance.
- Another possible extension to this work would be to extend the scheme for mobile computing environment. Since not all base stations in the world can be expected to use the proposed scheme, a challenging task would be to remove the dependence of the scheme on the control information from the network. The source TCP should try to infer all the information that is presently conveyed by the control information.

6.2.2 Possible extensions to the network simulator

Following extensions may be made to the network simulator.

- The simulator is an initial implementation. Its capabilities should be improved in future.
- An improved user interface would definitely enhance the ease of use of the simulator.
- Another possible extension would be to introduce components in the simulator that simplifies the task of analysis of simulator output. A simulation run of 10 MB file transfer typically produces an event trace

file of 150 KB size and other huge trace files of important parameters. Going through this information is a non-trivial task. It would be helpful if some filter may be added to the simulator that allows a user to quickly identify problem areas and concentrate on that.

Appendix A

IEEE 802.11 MAC Protocol

Study Group 802.11 was formed under IEEE project 802 to recommend an international standard for WLAN. The scope of the study group is to develop MAC layer and physical standards for wireless connectivity of fixed portable, and mobile stations within a local area. This appendix present the design objectives and the working of the recommended MAC protocol. The details presented in this appendix are taken from [CG96].

A.1 Design Objectives

An important design requirement for WLANs is that mobile/wireless hosts be able to communicate with other mobile/wireless hosts and wired hosts on other IEEE 802 LANs or networks in a transparent manner :

- A WLAN should appear as just another 802.x LAN to logical link layer control (LLC) and above layers.
- The response time should not be so large that the productivity of end-users is compromised.

The physical layers for which standards have been developed are direct sequence spread spectrum (DSSS), frequency-hopping spread spectrum (FHSS), and diffuse infrared.

A.2 IEEE 802.11 MAC Protocol

The 802.11 MAC layer protocol provides asynchronous, time bounded, and contention free access on a variety of physical layers. These functions are provided independent of the characteristics of underlying physical layers and/or data rates. The basic access method in 802.11 MAC protocol is the distributed coordination function (DCF) which is best described as the carrier sense multiple access with collision avoidance (CSMA/CA) protocol.

When using DCF, a station, before initiating a transmission senses the state of the channel to determine if another station is transmitting. The station proceeds with its transmission if the medium is determined to be idle for an interval that exceeds the distributed interframes space (DIFS). If the medium is busy, the station defers until after a DIFS is detected and then generates a random backoff period for an additional defer interval before transmitting. This minimizes collisions during contention between multiple stations. The backoff period is used to initialize the backoff timer. The backoff timer is decremented only when the medium is idle; it is frozen when the medium is busy. After a busy period, the decrementing of the backoff timer resumes only after the medium has been free longer than DIFS. A station initiates a transmission when the backoff timer reaches zero.

Since a transmitter cannot determine if the data frame was successfully received by listening to its own transmission, immediate positive acknowledgments are employed to determine the successful reception of each data frame. This is accomplished by the receiver initiating the transmission of an acknowledgment frame after a time interval, the short interframe space (SIFS), that is less than the DIFS, immediately following the reception of the data frame. The acknowledgment is transmitted without the receiver sensing the state of the channel. In case an acknowledgment is not received, the data frame is presumed lost.

Figure A.1 summarizes the basic access method.

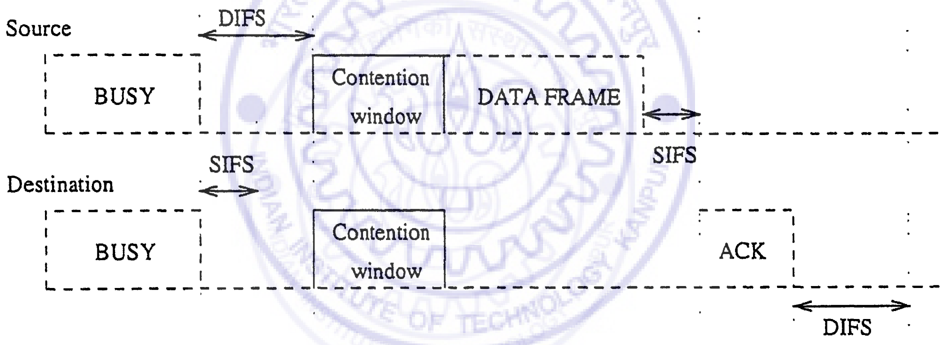

Figure A.1: Basic access method

Appendix B

User's Manual

B.1 Overview

The simulator takes as input a description of network topology and its components. It produces as output, trace of important attributes and a trace of events that took place during the simulation.

B.2 Simulator Input

The simulated network is modeled as a graph with nodes (vertices) and communication links (links) as its components. The input to the simulator is a description of these components. The input also describes some network wide attributes.

The simulator reads this input from a file. For the sake of simplicity, the name of this file has been fixed to `datafile.dat`). This file contains a list of network components along with the value of their attributes. The general structure of the file is as follows :

```
# comment1
name_of_component1
{
  name_of_attribute1 = value1
```

```
    name_of_attribute2 = value2
    .
    .
    .
}

# comment2
name_of_component2
{
    name_of_attribute3 = value3
    name_of_attribute4 = value4
    .
    .
    .
}

.
.
.
```

Note that each <attribute, value> pair must be specified on a separate line.

Comments may be placed anywhere in the file. A comment must be preceded by a #.

B.3 Network components supported

This section describes the different network components supported by the simulator.

B.3.1 Component simulator

It specifies the value of network wide parameters.

Parameters :
Following parameters must be specified :

- `system_simulation_time`
  It specifies the maximum value of simulated clock time (in milli-seconds). Simulation is continued till the value of simulated clock time is less than this value.

- `print_interval`
  The simulator supports the concept of periodic trace generation. `print_interval` specifies its frequency (in milli-seconds).

  The periodic trace contains the state information of each of the node present in this simulator. This trace is useful for detailed debugging purposes. It is generated as part of the event trace.

- `read_from_trace`
  This attribute is useful when *one and only one* wireless link is present in the simulated network topology.

  If `read_from_trace` is set to 0, the simulator *uses* a trace file to determine the time at which the wireless link should switch from GOOD state to BAD state and vice-versa, during the course of a simulation. This feature is especially useful when the desired behavior of a wireless link cannot be produced by a mathematical model.

  If `read_from_trace` is set to 1, the simulator *produces* a trace of time at which the wireless link switched from GOOD state to BAD state and vice-versa, during the course of a simulation.

  If `read_from_trace` is set to 2, a patch in the module defining behavior of a wireless link gets activated. This patch is useful in dropping specific packets over a wireless link by forcing it into BAD state at specific instants of time. The trace so generated records the desired behavior of the wireless link. This trace may then be used in subsequent simulations for defining the behavior of the wireless link.

Note that when read_from_trace is set to 2, the mean value of exponential distribution characterizing the duration of stay of a wireless link in BAD state, must be set to 0.

For the sake of simplicity the trace file used for the purpose is fixed to trace.dat.

Example :

```
simulator
{
  system_simulation_time = 4300000
#                    The time is specified in msecs.
  print_interval = 500
#                    The time is specified in msecs.
  read_from_trace = 1
#                    Value of 1 or 2 means trace_file will
#                    be generated. 1, no manipulations done.
#                    2, manipulations are also done.
#                    0 means trace_file will be used for
#                    determining the wireless link
#                    characteristics.
}
```

B.3.2 Component wired_source_node

Attributes :
Following attributes must be specified for this type of node :

- node_id
  It specifies an id for this node. There should be a unique id for every node present in the simulator.

- dest_id
  It specifies id of the destination node to which all the TCP packets will be sent.

- num_of_pkts_to_send
  It specifies the number of packets that will be generated by this node.

- pkt_size
  It specifies the size (of data portion) of each TCP packet (in bytes).

- recv_wnd_size
  It specifies the value of receiver's advertised window size (in KB). This value remains fixed throughout the simulation.

Example :

```
  wired_source_node
  {
      node_id = 211
#                       This should be numeric & unique.
      dest_id = 213
#                       This should be numeric
      num_of_pkts_to_send = 100
      pkt_size = 1024
      recv_wnd_size = 16
#                 The value of receiver's advertised window size.
   }
```

B.3.3 Component wireless_source_node

Attributes :
Same as those of wired_source_node.

B.3.4 Component normal_router_node

Attributes :
Following attributes must be specified for this type of node :

- node_id
  It specifies a unique id for this node.

- buffer_size
  It specifies the size of buffers (in bytes) available with this node. A packet is dropped if it is not possible to accommodated it in the available buffers.

- processing_delay
  It specifies a constant processing delay (in milli-seconds) for every packet processed by this node.

Example :

```
   normal_router_node
   {
      node_id = 212
      buffer_size = 512000
#             Specified in bytes
      processing_delay = 0
#             Specified in msecs
   }
```

B.3.5 Component base_station

Attributes :
Same as those of normal_router_node.

B.3.6 Component wired_sink_node

Attributes :
Following attribute must be specified for this type of node :

- `node_id`
  It specifies a unique id for this node.

Example :

```
wired_sink_node
{
    node_id = 213
}
```

B.3.7 Component wireless_sink_node

Attributes :
Same as that of `wired_sink_node`

B.3.8 Component wired_link

Attributes :
Following attributes must be specified for this type of link :

- `link_id`
  It specifies a unique id for this link. Every link must have a unique id.

- `bandwidth`
  It specifies the bandwidth of the link (in bits/sec).

- `latency`
  It specifies the propagation delay (latency) of this link (in milli-seconds).

- `node_id1` and `node_id2`
  They specify the id of the endpoints of this link.

- `loss_probability`
  It specifies the probability of a packet loss over this link.

- `random_seed`
  The random number generator for this link makes use of this seed.

Example :

```
  wired_link
  {
    link_id    = 101
    bandwidth = 8000000
#                  This is specified in bits/sec.
    latency = 492
#                  The time is specified in msec.
    node_id1 = 211
    node_id2 = 212
#                  The node ids of nodes that this link connect.
    loss_probability = 0
    random_seed = 2;
  }
```

B.3.9 Component `wireless_link`

Attributes :
Following attributes must be specified for this type of link :

- `link_id`
  Same as in `wired_link`
- `bandwidth`
  Same as in `wired_link`
- `latency`
  Same as in `wired_link`
- `node_id1` and `node_id2`
  Same as in `wired_link`

- mean_exp_distrib_good_state
  It specifies the mean value of the exponential distribution characterizing the duration of stay in GOOD state.

- mean_exp_distrib_bad_state
  It specifies the mean value of the exponential distribution characterizing the duration of stay in BAD state.

- random_seed
  Same as in wired_link

Example :

```
    wireless_link
    {
      link_id    = 102
      bandwidth = 1000000
#                   This is specified in bits/sec.
      latency = 1
#                   The time is specified in msec.
      node_id1 = 212
      node_id2 = 213
#                   The node ids of nodes that this link connect.
      mean_exp_distrib_good_state = 5.0
#                   The mean value of exponential distribution
#                   (for GOOD state)
      mean_exp_distrib_bad_state = 0.1
#                   The mean value of exponential distribution
#                   (for BAD state)
      random_seed = 3;
    }
```

B.3.10 Component shared_links

This does not define a new component of the simulated network but is used to modify the characteristics of a wireless link, defined earlier through the wireless_link entry. Its attributes are a set of link ids of wireless links.

A wireless link component defined through a wireless_link entry is by default *full duplex*. User may change its model to a *shared link* by mentioning its link id as part of the set of attributes of shared_links entry. All the wireless links referred to in a shared_links entry shares the same bandwidth although, each one has a different characteristic as defined by their corresponding wireless_link entry.

All the wireless links referred in a shared_links entry must have their corresponding wireless_link entry present earlier in the input file. This is the only restriction on the ordering of the components in the input file.

More than one shared_links entry may be present in the input file.

Attributes :

A attribute set is a sequence of link_id(s) of existing wireless links.

Example :

```
shared_links
{
    link_id = 102
}
```

B.4 Simulator Output

The output of the simulator is a set of trace files containing traces of important parameters. A trace file is generated in a two column format and can be viewed using *gnuplot*.

The name of the trace of a particular parameter is derived by concatenating the abbreviated name of that parameter with the id of the node. Thus,

$$<\textit{trace\_file\_name}> = <\textit{node\_id}>\text{-}<\textit{abbreviated-attribute-name}>$$

For eg., the trace of congestion window of node whose id is 211 will be placed in the file 211-congestion.

The rule for deriving the name of the trace file for wireless link characteristics is slightly different. The name is derived by concatenating the abbreviated parameter name with the node ids of both the endpoint nodes.

The following is the list of various attributes whose trace is generated by the simulator. In the brackets, the abbreviated names of these parameters are mentioned.

- Sequence number of the segments sent by the source (seqno)
- Acknowledgments received at the source (ackno)
- Congestion window (congestion)
- Slow start threshold size (threshold)
- Wireless link characteristics (wless_state)
- Sequence number of packets as they arrive at a router (router_pkt_arrival)
- Amount of unacknowledged data bytes at the source (swnd_size)

The simulator also produces a trace of events that took place during a simulation run. This event trace is produced on the standard output and may be redirected to a file.

A sample event trace is given below :

```
****** PKT CORRUPTED ******* (DLL)
(CLOCK : 945)
The following pkt was corrupted due to BAD status of the link
The contents of DLL frame are :-
<SRC, DST, SEQ, ACK, Type>:- <212, 213, 15, -1, DATA>
The contents of IP pkt are :-
<SRC ID, DST ID> : <211, 213>
The contents of TCP segment are :-
<SEQ, ACK, Len, Type> : <15360, -1, 1024, DATA>
```

```
****** PKT CORRUPTED ******* (DLL)
(CLOCK : 956)
The following pkt was corrupted due to BAD status of the link
The contents of DLL frame are :-
<SRC, DST, SEQ, ACK, Type>:- <212, 213, 16, -1, DATA>
The contents of IP pkt are :-
<SRC ID, DST ID> : <211, 213>
The contents of TCP segment are :-
<SEQ, ACK, Len, Type> : <15360, -1, 1024, DATA>

****** PKT CORRUPTED ******* (DLL)
(CLOCK : 967)
The following pkt was corrupted due to BAD status of the link
The contents of DLL frame are :-
<SRC, DST, SEQ, ACK, Type>:- <212, 213, 17, -1, DATA>
The contents of IP pkt are :-
<SRC ID, DST ID> : <211, 213>
The contents of TCP segment are :-
<SEQ, ACK, Len, Type> : <15360, -1, 1024, DATA>

****** PKT CORRUPTED ******* (DLL)
(CLOCK : 978)
The following pkt was corrupted due to BAD status of the link
The contents of DLL frame are :-
<SRC, DST, SEQ, ACK, Type>:- <212, 213, 18, -1, DATA>
The contents of IP pkt are :-
<SRC ID, DST ID> : <211, 213>
The contents of TCP segment are :-
```

```
<SEQ, ACK, Len, Type> : <15360, -1, 1024, DATA>

****** PKT CORRUPTED ******* (DLL)
(CLOCK : 989)
The following pkt was corrupted due to BAD status of the link
The contents of DLL frame are :-
<SRC, DST, SEQ, ACK, Type>:- <212, 213, 19, -1, DATA>
The contents of IP pkt are :-
<SRC ID, DST ID> : <211, 213>
The contents of TCP segment are :-
<SEQ, ACK, Len, Type> : <15360, -1, 1024, DATA>

****** PKT CORRUPTED ******* (DLL)
(CLOCK : 1000)
The following pkt was corrupted due to BAD status of the link
The contents of DLL frame are :-
<SRC, DST, SEQ, ACK, Type>:- <212, 213, 20, -1, DATA>
The contents of IP pkt are :-
<SRC ID, DST ID> : <211, 213>
The contents of TCP segment are :-
<SEQ, ACK, Len, Type> : <15360, -1, 1024, DATA>

****** PKT CORRUPTED ******* (DLL)
(CLOCK : 1011)
The following pkt was corrupted due to BAD status of the link
The contents of DLL frame are :-
<SRC, DST, SEQ, ACK, Type>:- <212, 213, 21, -1, DATA>
The contents of IP pkt are :-
<SRC ID, DST ID> : <211, 213>
```

```
The contents of TCP segment are :-
<SEQ, ACK, Len, Type> : <15360, -1, 1024, DATA>

****** PKT CORRUPTED ******* (DLL)
(CLOCK : 1022)
The following pkt was corrupted due to BAD status of the link
The contents of DLL frame are :-
<SRC, DST, SEQ, ACK, Type>:- <212, 213, 22, -1, DATA>
The contents of IP pkt are :-
<SRC ID, DST ID> : <211, 213>
The contents of TCP segment are :-
<SEQ, ACK, Len, Type> : <15360, -1, 1024, DATA>

****** PKT CORRUPTED ******* (DLL)
(CLOCK : 1033)
The following pkt was corrupted due to BAD status of the link
The contents of DLL frame are :-
<SRC, DST, SEQ, ACK, Type>:- <212, 213, 23, -1, DATA>
The contents of IP pkt are :-
<SRC ID, DST ID> : <211, 213>
The contents of TCP segment are :-
<SEQ, ACK, Len, Type> : <15360, -1, 1024, DATA>

Following frame could not be transmitted after 8 attempts :
The contents of DLL frame are :-
<SRC, DST, SEQ, ACK, Type>:- <212, 213, 23, -1, DATA>
The contents of IP pkt are :-
<SRC ID, DST ID> : <211, 213>
The contents of TCP segment are :-
```

```
<SEQ, ACK, Len, Type> : <15360, -1, 1024, DATA>

TCP_SRC:: Fast Retransmit
TCP_SRC(Node-id - 211):: following TCP segment RETRANSMITTED(CLOCK : 1174
The contents of TCP segment are :-
<SEQ, ACK, Len, Type> : <15360, -1, 1024, DATA>

Processing SIMULATION_OVER event.
Terminating the simulator.
Normal Termination.
```

Bibliography

[BB95] A. Bakre and B.R. Badrinath. I-TCP : Indirect TCP for mobile hosts. In *Proc. 15th International Conference on Distributed Computing Systems (ICDCS)*, May 1995.

[BBKT96] P. Bhagwat, P. Bhattacharya, A. Krishna, and S.K. Tripathi. Enhancing throughput over wireless LANs using channel state dependent packet scheduling. In *Proc. IEEE INFOCOM'96, San Francisco, CA*, pages 1133–1140, March 1996.

[BOP94] L.S. Brakmo, S.W. O'Malley, and L.L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proc. ACM SIGCOMM'94, London, UK*, volume 24, pages 24–35, August 1994.

[BP95] 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*, pages 1465–1480, October 1995.

[BPSK96] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. In *Proc. ACM SIGCOMM'96, Stanford, CA*, pages 256–269, August 1996.

[BPT95] P. Bhagawat, C. Perkins, and S.K. Tripathi. Network layer mobility: an architecture and survey. Technical Report CS-TR-3570, Department of Computer Science, University of Maryland, College Park, 1995.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, and R.H. Katz. Improving TCP/IP performance over wireless networks. In *Proc. 1st ACM*

International Conference on Mobile Computing and Networking (Mobicom), November 1995.

[CDJM91] R. Caceres, P.B. Danzig, S. Jamin, and D.J. Mitzel. Characteristics of wide-area TCP/IP conversations. In *Proc. ACM SIGCOMM'91, Zurich, Switzerland*, volume 21, pages 101–112, September 1991.

[CG96] H.S. Chhaya and S. Gupta. Performance of asynchronous data transfer methods of IEEE 802.11 MAC protocol. *IEEE Personal Communications*, pages 8–15, October 1996.

[Che94] K.C. Chen. Medium access control of wireless LANs for mobile computing. *IEEE Network*, September 1994.

[CI95] R. Caceres and L. Iftode. Improving the performance of reliable transport layer protocols in mobile computing environments. *IEEE Journal on Selected Areas in Communications*, 13(5), June 1995.

[DCY93] A. DeSimone, M.C. Chuah, , and O.C. Yue. Throughput performance of transport layer protocols over wireless LANs. In *Proc. Globecom'93*, December 1993.

[DFJ91] D. Duchamp, S.K. Feiner, and G.Q. Maguire, Jr. Software technology for wireless mobile computing. *IEEE Network Magazine*, pages 12–18, November 1991.

[DJ91] P.B. Danzig and S. Jamin. tcplib: A library of TCP internetwork traffic characteristics. Technical Report CS-SYS-91-01, USC Networking and Distributed Systems Laboratory, October 1991. URL : ftp://catarina.usc.edu/pub/jamin/tcplib.

[DMT96] R.C. Durst, G.J. Miller, and E.J. Travis. TCP extensions for space communication. In *Proc. 2nd ACM Conference on Mobile Computing and Networking (Mobicom'96)*, November 1996.

[DR92] D. Duchamp and N.F. Reynolds. Measured performance of a wireless LAN. In *Proc. 17th Conference on Local Computer Networks, IEEE, Minneapolis*, pages 494–499, September 1992.

[FF96] K. Fall and S. Floyd. Simulation based comparison of Tahoe, Reno, and SACK TCP. *ACM Computer Communication Review*, 26(3):5–21, July 1996.

[Flo95] S. Floyd. TCP and successive fast retransmits. URL : ftp://ftp.ee.lbl.gov/papers/fastretrans.ps, February 1995.

[FZ94] G. H. Forman and J. Zahorjan. Survey: The challenges of mobile computing. *IEEE Computer*, 27(4):38–47, April 1994.

[Gup95] V. Gupta. Simulation of TCP/IP over high-speed networks. Master's thesis, Indian Institute of Technology, Kanpur, April 95.

[Hey90] A. Heybey. NETSIM: The network simulator. September 1990.

[Hoe95] J.C. Hoe. Start-up dynamics of TCP's congestion control and avoidance schemes. Master's thesis, Massachusetts Institute of Technology, June 1995.

[Hoe96] J.C. Hoe. Improving the start-up behavior of a congestion control scheme for TCP. In *Proc. ACM SIGCOMM'96, Stanford, CA*, pages 270–280, August 1996.

[IDJ91] J. Ioannidis, D. Duchamp, and G. Q. Maguire, Jr. IP-based protocols for mobile internetworking. In *Proc. ACM SIGCOMM'91, Zurich, Switzerland*, volume 21, pages 235–245, September 1991.

[Jac88] V. Jacobson. Congestion avoidance and control. In *Proc. ACM SIGCOMM'88, Stanford, CA*, volume 18, pages 314–329, August 1988.

[Jai90] R. Jain. Congestion control in computer networks: Issues and trends. *IEEE Network Magazine*, pages 24–30, May 1990.

[Kat94] R. H. Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, 1(1), 1994.

[Kes88] S. Keshav. REAL: A network simulator. Technical Report 88/472, Computer Science Division, University of California at Berkeley, 1988.

[LKBP96] R.O. LaMaire, A. Krishna, P. Bhagwat, and J. Panian. Wireless LANs and mobile networking : Standards and future directions. *IEEE Communication Magazine*, pages 86–94, August 1996.

[MF95] S. McCanne and S. Floyd. NS (network simulator). URL : http://www-nrg.ee.lbl.gov/ns, 1995.

[MM96] M. Mathis and J. Mahdavi. Forward acknowledgment: Refining TCP congestion control. In *Proc. ACM SIGCOMM'96, Stanford, CA*, pages 281–291, August 1996.

[Nag84] J. Nagle. Congestion control in IP/TCP internetworks. RFC: 896, Network Information Center, January 1984.

[NED94] S. Nanda, R. Ejzak, and B.T. Doshi. A retransmission scheme for circuit mode data on wireless links. *IEEE Journal on Selected Areas in Communications*, 12(8), October 1994.

[NKNS96] G.T. Nguyen, R.H. Katz, B. Noble, and M. Satyanarayanan. A trace-based approach for modeling wireless channel behavior. In *Proc. Winter Simulation Conference*, December 1996.

[NNSK96] B. Noble, G. Nguyen, M. Satyanarayanan, and R. Katz. Mobile network tracing. RFC: 2041, Network Information Center, October 1996.

[Pos81a] J. B. Postel. Internet Control Message Protocol. RFC: 792, Network Information Center, ISI, University of Southern California, Marina del Rey, CA, September 1981.

[Pos81b] J. B. Postel. Internet Protocol. RFC: 791, Network Information Center, ISI, University of Southern California, Marina del Rey, CA, September 1981.

[Pos81c] J. B. Postel. Transmission Control Protocol. RFC: 793, Network Information Center, ISI, University of Southern California, Marina del Rey, CA, September 1981.

[Ste94] W. R. Stevens. *TCP/IP Illustrated, Volume 1 : The Protocols.* Addison-Wesley, Reading, MA, 1994.

[Ste97] W. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC: 2001, Network Information Center, January 1997.

[TYT91] F. Teraoka, Y. Yokote, and M. Tokoro. A network architecture providing host migration transparency. In *Proc. ACM SIGCOMM'91, Zurich, Switzerland*, volume 21, pages 209–220, September 1991.

[WS96] G. R. Wright and W. R. Stevens. *TCP/IP Illustrated, Volume 2 : The Implementation.* Addison-Wesley, Reading, MA, 1996.

[YB94] R. Yavatkar and N. Bhagwat. Improving end-to-end performance of TCP over mobile internetworks. *In Mobile 94 Workshop on Mobile Computing Systems and Applications*, December 1994.