Challenges to Reliable Data Transport over Heterogeneous Wireless Networks

by

Hari Balakrishnan

B. Tech. (Indian Institute of Technology, Madras) 1993
M.S. (University of California, Berkeley) 1995

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Randy H. Katz, Chair
Professor Robert W. Brodersen
Professor Paul K. Wright

1998

The dissertation of Hari Balakrishnan is approved:

Chair Date

Date

Date

University of California at Berkeley

1998

Challenges to Reliable Data Transport over Heterogeneous Wireless Networks

by

Hari Balakrishnan

Abstract

Challenges to Reliable Data Transport over Heterogeneous Wireless Networks

by

Hari Balakrishnan

Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Randy H. Katz, Chair

The Transmission Control Protocol (TCP) is the de facto standard for reliable data transmission in the Internet today. While TCP has been tuned to work well over traditional wired networks, its performance over wireless networks is much worse. This performance degradation results from several effects:

- The preponderance of channel error-induced packet losses compared to congestion-induced losses.
- Asymmetric effects and latency variation due to adverse interactions between media-access protocols and TCP.
- Small TCP transmission windows due to low wireless channel bandwidths.

While TCP adapts well to network congestion, it does not adequately handle the vagaries of wireless media. In this thesis, we address these challenges in detail and design solutions to them. These solutions incorporate link-layer techniques as well as enhancements to TCP at the sender and receiver.

The adverse effects of wireless bit-errors on TCP performance arise primarily because TCP misinterprets wireless losses as being due to congestion. We present the design and implementation of a novel protocol, called the Berkeley Snoop Protocol, that exploits cross-layer protocol optimizations to improve performance. In addition, we design a mechanism called Explicit Loss Notification that can successfully distinguish between congestion and channel error-induced packet losses to substantially enhance end-to-end performance. These mechanisms have been demonstrated to provide performance improvements between 100% and 2000% across a range of bit-error rates in various cellular topologies in a Lucent WaveLAN-based wireless LAN, for data transfer to and from mobile hosts and when wireless transit links are present.

Asymmetric networks pose a challenge to feedback-based protocols like TCP, because the characteristics of the reverse path used for acknowledgments (ACKs) have the potential to seriously affect forward performance. We classify asymmetry with respect to TCP performance into several categories, such as bandwidth asymmetry, latency asymmetry, loss-rate asymmetry, etc. We design several end-to-end and transport-aware link-layer solutions, such as ACK filtering, TCP sender adaptation and ACK reconstruction to obtain significantly better performance over such networks. These techniques lead to performance improvements for Internet access over packet radio networks such as Metricom's Ricochet network and for bandwidth-asymmetric networks such as Hybrid's wireless cable network.

Low channel bandwidths are common in many wireless networks as well as on many Internet paths today. This, and the short connections common in many Web transfers, lead to TCP windows being very small. This greatly affects its data-driven loss recovery mechanism and causes a large number of timeouts. In addition to implementing and evaluating the performance of TCP Selective Acknowledgments, we analyze data from packet-level traces of the Web server of the Atlanta Olympic Games to motivate enhancements to TCP's loss recovery mechanism. Our enhancements lead to an Enhanced Recovery scheme for TCP that greatly reduces the number of timeouts when windows are small.

In this research, we learn some general lessons about protocol design for heterogeneous networks. Specifically, we demonstrate the significant performance benefits of cross-layer protocol optimizations, of soft-state agents in the network infrastructure, and the advantages of data-driven loss recovery over timer-driven ones for better performance.

Today, wireless data networks are not widely deployed in the Internet despite the availability of a wide range of wireless technologies: one of the main reasons for this is their poor price-to-performance ratio for reliable data transfers. We believe (and hope!) that by significantly improving end-to-end TCP performance by up to an order of magnitude in some cases, our solutions will accelerate the widespread deployment and adoption of wireless data networks and make them an integral part of the Internet infrastructure of the future.

_____

Professor Randy H. Katz, Chair

To my great-grandmother, Seethalakshmi,

my grandmother, Janaki,

and my parents, Radha and Balakrishnan

Table of Contents

List of Figures

List of Tables

Acknowledgments

It has been almost five years since I started my graduate education at Berkeley, and when I sit back and think about the number of people who influenced me and helped me complete this dissertation, I am overwhelmed! There is no doubt that this would have been impossible without their help. I hope that I can remember everyone who helped me through this difficult yet rewarding process.

First and foremost, I would like to thank my advisor, Professor Randy Katz, for taking me on as a student about four years ago, even though he knew little about me at the time. It was an extraordinary piece of good fortune that led to my becoming his student. He has been an ideal advisor in every respect, both in terms of technical advice on my research and in terms of professional advice. My choice of career path has been greatly influenced by Randy and I hope that I can live up to his high standards.

I thank Professors Bob Brodersen and Paul Wright for serving on my qualifying exam and dissertation committees. I had a great time working on the Infopad project under Bob's supervision and learnt a lot from my extended association with Infopad. Paul taught one of the most interesting courses that I took, on intelligent manufacturing systems, which broadened my outlook and taught me what *real* engineers do!

I benefited greatly from the technical and career advice given to me by Deborah Estrin, Sally Floyd, Steven McCanne, Vern Paxson, K. K. Ramakrishnan and Srinivasan Seshan. I am grateful to them for this and look forward to interacting with them in the future.

I am grateful to Darrell Long, Patrick Mantey and Eric Rosen of U.C. Santa Cruz for continuous access to the Reinas wireless network. They were very tolerant users of the Snoop protocol and suffered through crashes and downtimes until several bugs were ironed out. I hope that this software continues to serve them well. I would also like to thank numerous members of the wireless and network research communities who downloaded the implementations of Snoop and TCP SACK, installed them (sometimes porting them to other systems), and sent me useful feedback. My work would have been impossible without *ns*, and I thank the members of the VINT project for the splendid simulation testbed.

The Bureaucracy at Berkeley can make life difficult for graduate students, but I had no trouble with Sproul Hall due to the tremendous efforts of Kathryn Crabtree. Her friendly attitude, approachability and sense of humor made it easy to get all administrative problems solved. Terry Lessard-Smith and Bob Miller were wonderful project administrators, responding promptly to my frequent requests for equipment and other goodies. I thank them for their efforts and more importantly, for their warm friendship.

My research was primarily supported by DARPA under contract DAAB07-95-C-D154 and by a Segasoft research grant from the Okawa Foundation. It was also supported by grants or equipment from Hughes Aircraft Corp., Hybrid Networks Inc., Metricom Inc. and the California MICRO program.

I am grateful to Mike Ritter, Mike Cunningham (Metricom Inc.) and Ed Moura (Hybrid Inc.) for helping in the installation of substantial infrastructure for my experimental work. I sincerely thank Ken Lutz, Brian Shiratsuki and Keith Sklower who helped run our testbed.

The Daedalus research group set the stage for this dissertation and the numerous discussions I had with its members greatly improved the quality of my work. I thank its members, past and present—Elan Amir, Tom Henderson, Todd Hodes, Dan Jiang, Giao Nguyen, Venkat Padmanabhan, Srini Seshan, Mark Stemm, Helen Wang and Tao Ye, and the members of the sister Glomop research group—Eric Brewer, Yatin Chawathe, Armando Fox and Steve Gribble—for an intense and fun intellectual environment. My collaborators in this group—Srini, Venkat, Elan and Mark—had an enormous impact in shaping my thinking about networking, systems, and on the contents of this thesis.

One of the distinguishing features of many Berkeley research groups is the semi-annual retreats and Randy's Daedalus group was no exception to this. These retreats, at Tahoe and Monterey, helped shape our research agenda and check our research progress. I got several suggestions and comments from the participants at these retreats over the past three years. In particular, B.R. Badrinath, Mary Baker, J.J. Garcia-Luna, Akhtar Jameel, Barry Leiner, Reiner Ludwig, Kevin Mills, Mike Ritter, Rob Ruth and Subir Varma offered useful suggestions and criticism on my work.

The stimulating atmosphere in 443 Soda with Elan, Venkat, Mark and Todd, and its logical extensions to Srini at IBM and Giao and Tom in 473 made it a pleasure to work there every day. The scope of discussions and arguments we had there over the past four years taught me a number of things that are sure to serve me well. Elan, in particular, has taught me a thing or two about being productive and simultaneously having fun; I admire (and hope to replicate!) his work ethic and his resolve to get things done efficiently.

There's more to life than just one's work. I was fortunate to have been part of a wonderful cricket team (Berkeley Cricket Club), have regular racquetball partners (Mark, Andrew Swan, and Srini), and have friends like Andrew, Mark and Ketan Patel introduce me to the vicissitudes of golf.

Srini Seshan played a major part in my graduate career. Over the years, he has listened patiently to numerous ideas, good and bad, and questions I have had and has given me extremely useful and incisive comments in return. I look forward to continue working with him and further developing our friendship.

I owe a special debt of gratitude to my parents and family. They have, more than anyone else, been the reason I have been able to get this far. Words cannot express my gratitude to my parents, my sister Hamsa, my grandmother and great-grandmother who give me their support and love from across the seas. They instilled in me the value of hard work and taught me how to overcome life's disappointments. My wife, Suchitra Raman, gives me her selfless support and love that make me want to excel. I am grateful to her for enriching my life.

"Begin at the beginning," the King said, gravely, "and go on till you come to the end; then stop."

Lewis Carroll, Alice in Wonderland, Chap. 12

The last thing one discovers in writing a book is what to put first.

— Blaise Pascal, Pensees, I., Chap. 19

Chapter 1

Introduction and Preview

1.1 Motivation

Among the several trends in networking and communication in the 1990s, two stand out prominently: (i) the rapid global expansion and use of the Internet for information access and (ii) the massive growth in the use of wireless cellular telephony for communication. The main reason for the first trend is the meteoric growth of the World Wide Web (WWW) [18] and the large amounts of information that one can find on it. The second trend, towards wireless connectivity, has been fueled by advances in communication hardware and cellular technologies that have enabled large-scale wireless telephone networks. Figure 1.1 shows both these rapidly growing trends.

One would therefore expect that the combination of these two growing technology trends—*Internet access* on the one hand and *wireless access* on the other—would result in a potent combination and that wireless data networks and services would dominate the marketplace. However, a survey of the commercial world shows that wireless data networks and services are floundering, with a lack of widespread proliferation of such systems.

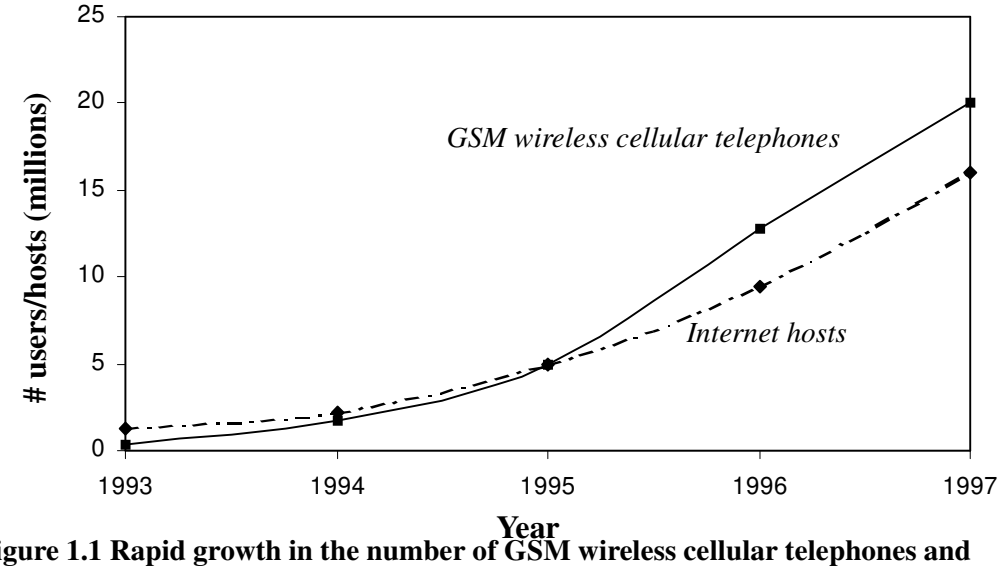

Figure 1.1 Rapid growth in the number of GSM wireless cellular telephones and hosts on the Internet as a function of time.
Sources: Ericsson, Inc. and Matthew Gray (MIT).

Why is this the case? Could it be that the inherent user demand for wireless data systems is non-existent? On the contrary, several user surveys show that this is not so, and that users do prefer the convenience offered by wireless Internet access [51]. Wireless data systems enable a plethora of new applications, especially for users on the move. For example, consider the following statement by George H. Heilmeier as a harbinger of the future [52]: *"People and their machines should be able to access information and communicate with each other easily and securely, in any medium or combination of media—voice, data, image, video, or multimedia—any time, anywhere, in a timely, cost-effective way."*

We believe that there are two important and closely related reasons that hamper the realization of this vision:

- *Poor performance:* Unfortunately, the performance of *reliable* data transport protocols over different wireless networks is unacceptably low today. Most Internet applications, including Web access, file transfers and electronic mail, require efficient reliable data transport. Therefore, the degraded performance of reliable transport protocols strongly discourages widespread use of wireless access networks, since the resulting price/performance ratio becomes too large compared to alternative wired access technologies.

- *Heterogeneity:* There is tremendous diversity in wireless access technologies ranging from wireless local-area networks to wide-area satellite systems. These technologies have different characteristics (e.g., bandwidth, latency, error characteristics, range, etc.) that make it hard to identify the reasons for poor performance. Current approaches to reliable data transport do not handle wireless heterogeneity well.

The vision of reliable and ubiquitous information access poses several challenges. In this thesis, we identify and solve the fundamental problems to efficient reliable data transport in heterogeneous wireless networks. The end result is an adaptive reliable transport protocol architecture that provides significantly improved end-to-end performance in heterogeneous networks of wired and wireless links, enabling a wide variety of wireless technologies to be profitably used in the Internet. We strongly believe that this work is an important step in making wireless devices first-class citizens in the future Internet.

The rest of this section elaborates on the issues of wireless heterogeneity and poor transport performance in wireless networks. In Section 1.2 we describe the protocol architecture of the Internet and its best-effort service model. Section 1.3 discusses the three fundamental challenges to efficient wireless data transport that we identify and overcome in this thesis. Section 1.4 presents an overview of our contributions and describes the organization of the rest of the thesis.

1.1.1 Wireless Heterogeneity

The recent growth in the number of wireless alternatives to gain access to the Internet has been remarkable. These technologies enable desktops, laptops and hand-held Personal Digital Assistants (PDAs) (e.g., [113]) to gain access to information sources on the Internet without any tethered connection. The wide range of wireless networks includes in-room Infrared networks, building-wide wireless LANs, campus-area packet radio networks, metropolitan-area cellular wireless networks, regional-area wireless cable networks and broadcast satellite networks.

Figure 1.2 shows four different wireless devices attached to a laptop computer to illustrate this point. These devices are different from each other in terms of maximum raw bandwidth, channel-access mechanisms, link protocols, frequency, power and mode of operation, etc. These differences make it hard to identify, analyze and solve performance problems.

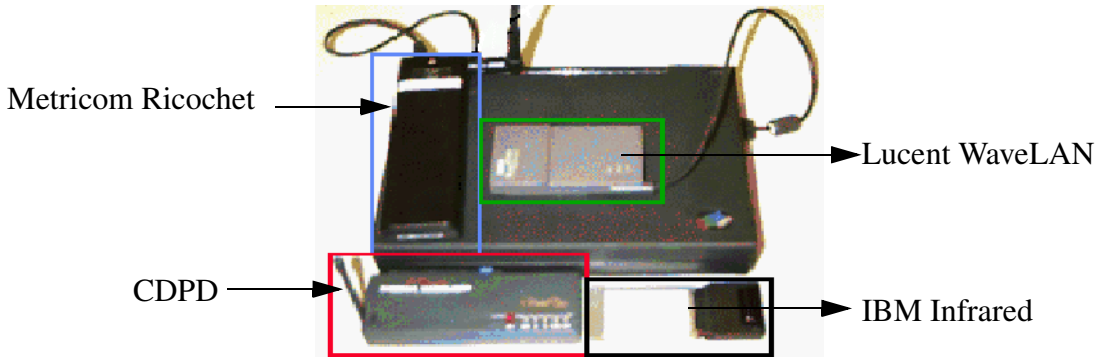

Figure 1.2 Wireless heterogeneity. Clockwise from top-left, the picture shows a Ricochet packet radio modem, a WaveLAN PC Card interface, an IBM Infrared interface and a Cellular Digital Packet Data (CDPD) modem.

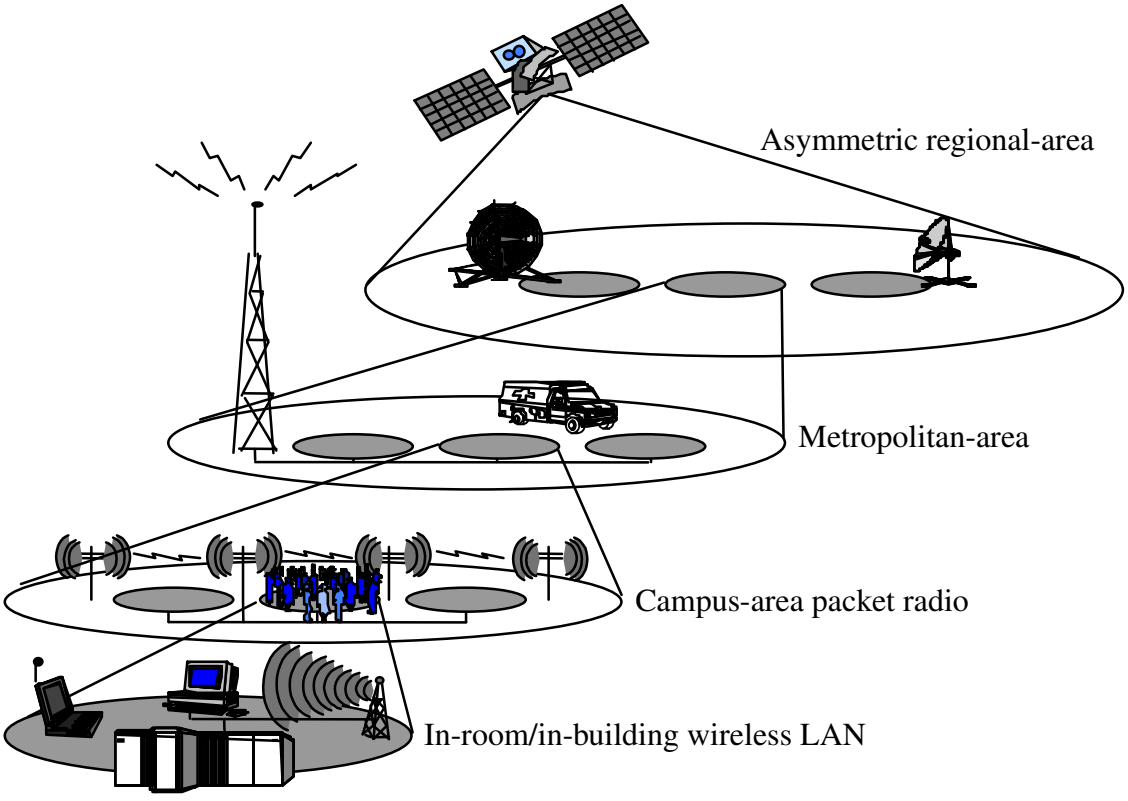

Figure 1.3 Wireless overlay network architecture.

One way of visualizing this diversity in access technologies is by using a tiered, overlay architecture as shown in Figure 1.3. At the bottom of this hierarchy are in-room and in-building networks that provide maximum bandwidths between 1 and 10 Mbps over a relatively small range. Exam-

Network	Max. raw bandwidth	1-way latency	Typical range	Salient characteristics
Room/Building wireless LAN (IR/RF)	> 1 Mbps	< 10 ms	< 300 meters	Pedestrian speed Shared medium
Campus-area packet radio (RF)	< 100 Kbps	100-500 ms (highly variable)	< 1 mile (via repeaters)	Slow vehicular Variable latency
Metro-area (Cellular, PCS, GSM)	< 20 Kbps	> 100 ms (highly variable)	Few miles	Vehicular Variable latency
Regional-area (Wireless cable)	10-30 Mbps (shared)	< 10 ms	Few miles (point-to-point)	Stationary Bandwidth asymmetry Relocatable mobility
Regional-area satellite networks	10-100 Mbps	>> 100 ms	Several hundred miles	Stationary Large latency Bandwidth asymmetry Relocatable mobility

Table 1.1 Characteristics of a variety of wireless data networks. These technology characteristics are current as of mid-1998.

ples of these include IBM's Infrared and Lucent's WaveLAN [145] technologies. The next tier is composed of campus-area packet radio networks like Metricom's Ricochet network [98]. These provide maximum link bandwidths of about 100 Kbps, within a range of a mile or less. Above these are metropolitan-area cellular data networks like the Cellular Digital Packet Data (CDPD) network [49] and the GSM data network [99]. These provide low bandwidths of less than 20 Kbps today, but their range is several miles long. Finally, at the top of this hierarchy are an intriguing set of regional-area networks, including wireless cable and direct broadcast satellites. These networks provide high bandwidth connectivity over a large shared area, but mainly in one direction alone. The opposite direction, used to request data and acknowledge successful data delivery, is over a much lower bandwidth path, often using a different technology like a telephone dialup line or a packet radio channel. Thus, these access technologies display bandwidth asymmetry, with differ-

ent available bandwidths in the two directions. Table 1.1 shows the salient features of these overlay network technologies.

The result of this technology diversity is a variety of problems that adversely affect reliable transport performance in these different networks, as explained below.

1.1.2 Poor Transport Performance

The performance of reliable transport protocols over most wireless networks today is unacceptably poor. Table 1.2 shows the throughput results of several experiments over four different wireless networks for a 1 MB bulk transfer. While there is a wide range in the measured performance depending on external conditions, the median throughputs are much closer to the lower end of the spectrum in every case. Thus, there is often an order of magnitude (or more) difference between what is rated for each technology and what is perceived by the user during actual usage. Our goal is to identify the reasons for this gap between perceived and rated performance and to bridge it.

Technology	Rated link bandwidth	Measured throughput
IBM Infrared	1 Mbps	100—800 Kbps
Lucent WaveLAN	2 Mbps	50 Kbps—1.5 Mbps
Metricom Ricochet	100 Kbps	10—35 Kbps
Hybrid wireless cable with dialup return	10 Mbps	500 Kbps—3 Mbps

Table 1.2 Rated link bandwidth and measured throughput for four different wireless technologies. In each case, median performance is close to the lower end of the spectrum than the higher. There is thus a wide gap between perceived TCP throughput and vendor-rated link performance.

Our research uncovers three fundamental challenges that significantly degrade the performance of reliable data transport protocols when the wired Internet is augmented with wireless access technologies:

- The preponderance of wireless bit-errors.
- Asymmetric effects and latency variation.

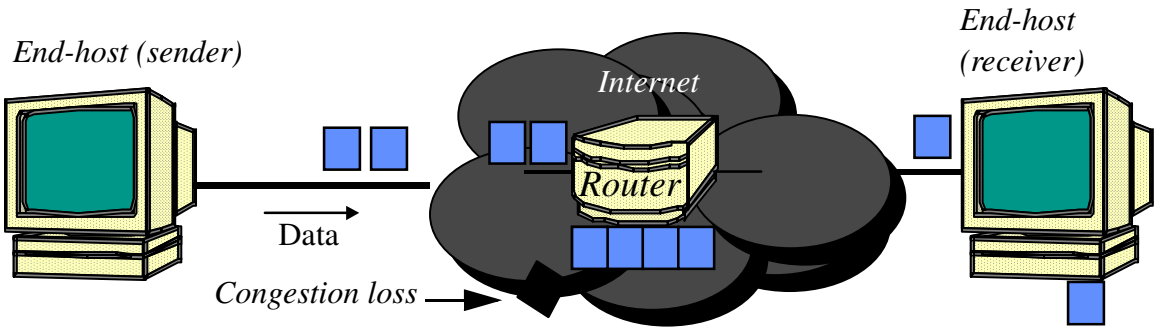

Figure 1.4 Packets between end-hosts on the Internet are routed through a set of routers. Router queues could overflow and cause packet loss during times of congestion.

- Small transmission windows due to low channel bandwidths.

Before discussing these challenges in detail (Section 1.3), we need to understand the protocol architecture in the Internet today and how reliable data transmission is achieved.

1.2 The Internet Protocol Architecture

The Internet is a *best-effort* network. Today's Internet infrastructure provides no guarantees on bandwidths or latencies and has no mechanisms to avoid packet losses. Packets between end-hosts are routed through a set of routers, as shown in Figure 1.4. Typically, in conventional wired networks, network congestion due to overload and router resource contention causes packet loss. This service model leads to a simple internal architecture that scales to handle a large number of communicating entities, but requires higher level protocols and applications to *adapt* to packet loss, varying available bandwidths and latencies.

1.2.1 Heterogeneity

The Internet is characterized by an enormous degree of heterogeneity. End-hosts range from powerful servers and desktop computers to so-called "thin clients" [113] and hand-held devices; network links range from multi-gigabit wired links to low-bandwidth dialup and wireless links; protocols range from unreliable unicast protocols (e.g., UDP [114]) to reliable multicast protocols [125]; and applications range from bandwidth-intensive video conferencing (e.g., [95]) to latency-sensitive remote terminal applications (e.g., [118). Despite this diversity, the Internet enables hosts to communicate with each other independent of the specific characteristics of the machines or

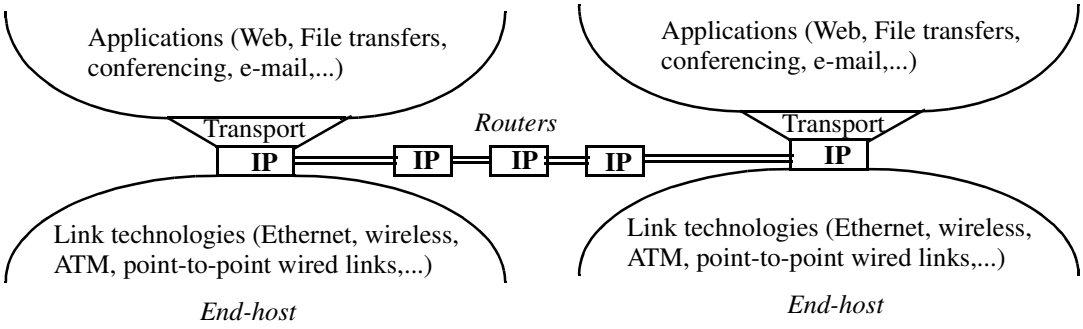

Figure 1.5 A "stovepipe" diagram showing a variety of different applications running on end-hosts communicating over a variety of different link technologies. The IP layer is the lingua franca for communication on the Internet, setting the standard for addressing nodes and routing packets between them.

access technology. This is achieved by using the Internet Protocol (IP) [115], which provides a common vocabulary for communication between hosts. That is, end-hosts can communicate with one another as long as they use IP for addressing and communication, via a network of routers that run IP. Thus, IP enables universal connectivity independent of host and link technology, making it the lingua franca of the Internet today (Figure 1.5).

1.2.2 Reliability

By bringing together a range of disparate technologies, IP solves one aspect of the heterogeneity problem: that of enabling basic connectivity between hosts. However, because the IP service model provides the abstraction of a best-effort network, packet losses can and do happen. There are numerous applications, such as World Wide Web access (e.g., using HTTP [44, 136]), file transfers (e.g., `ftp` [119]), electronic mail [117], interactive logins (e.g., `telnet` [118]), etc. that require *reliable, ordered* data delivery from the network. This is provided by a reliable transport protocol that runs over IP, that such applications can use via a well-defined socket interface [134].

To work well in the global Internet, a good reliable transport protocol must perform two critical functions: *loss recovery* and *congestion control.* Loss recovery is the ability of the sender to deduce that certain packets did not reach the receiver and retransmit them, either when requested by the receiver or without any explicit request from the receiver. Congestion control is the ability of the sender to recognize situations when the network is overloaded and react to it by reducing its transmission rate.

The *Transmission Control Protocol,* or *TCP* [116, 20, 135], is the de facto standard for reliable unicast data transport in the Internet today. It is used by virtually all unicast applications that require reliability. TCP achieves reliability by using *cumulative acknowledgments (ACKs)* that are sent by the receiver to the sender for all received data on a regular basis. A cumulative ACK acknowledges the successful receipt of all data received *in-order* so far. These ACKs are used by the sender to determine the successful delivery or loss of data. The TCP sender performs loss recovery by retransmitting packets that it considers missing, based on a lack of an ACK for it. Thus, TCP achieves reliability using an Acknowledgment-Repeat-reQuest (ARQ) [83] scheme with cumulative ACKs.

Until now, one of the main reasons the Internet has successfully withstood rapidly increasing growth is the sound congestion control principles on which TCP is based [59, 30]. TCP uses a window-based congestion and flow control mechanism, where data transmissions are governed by a dynamically changing window size that attempts to track available bandwidth and react to congestion. The TCP sender is *ACK-clocked*; it relies on timely ACK feedback from the receiver to transmit new data smoothly. Since TCP has no a priori knowledge of the state of the network, it uses the rate at which ACKs arrive to adjust its window and deduce the sustainable data rate for the connection. In response to any packet loss (at which time it also performs a retransmission), TCP reduces its window and hence its transmission rate by at least a factor of two, reducing the load on the network. An implicit assumption made by TCP's congestion control algorithm is that *all* data loss is caused by network congestion. While this is a valid assumption in conventional wired networks, it is not true in most wireless systems.

Thus, TCP's congestion control and loss recovery mechanisms are tightly coupled to each other, which leads to degraded performance in wireless networks where losses frequently occur for reasons other than congestion. Furthermore, any irregularity in the ACK stream skews TCP's ACK-clocking mechanism and manifests itself in the data stream in the immediate future. This situation arises in bandwidth-asymmetric and packet radio networks.

1.2.3 Incremental Deployment

In addition to enormous heterogeneity, the Internet is characterized by tremendous scale, as shown in Figure 1.1. By virtue of the fact that there are over 16 million hosts in the Internet today [86],

"legacy" systems are an integral part of the Internet landscape. It is therefore crucial to recognize the need for an incremental and evolutionary approach towards protocol design and deployment in large-scale heterogeneous networks like the Internet [46]. Otherwise, it is hard to make impact and influence its future evolution.

There are two important aspects to incremental deployment that Internet researchers must address: (i) developing a clear deployment path from where the world is today and where they would like to take it with their protocol enhancements and (ii) ensuring that there are no adverse interactions between their solutions and existing protocols and applications. Not meeting either of these requirements would greatly reduce the widespread acceptance and deployment of any proposed scheme.

We use the TCP/IP protocol architecture as the starting point for our solutions to wireless problems rather than designing a new protocol architecture completely from scratch. There are significant advantages in adopting this approach. The first is the ability to incrementally deploy our changes in the global Internet and realize the vision of making wireless devices first-class citizens in the Internet. The second is being able to operate with currently-deployed legacy protocol implementations at certain hosts in the network, obviating the (impossible-to-achieve) need for a "flag day," a day on which all Internet hosts need to change to a new protocol architecture.

Despite these benefits, it is still justifiable to design a new protocol architecture from scratch, *if* the features in the new protocol cannot be implemented in the TCP/IP framework. Our experience with a wide variety of algorithms and techniques demonstrates the flexible and extensible nature of the TCP/IP protocol architecture. Using a combination of TCP options, local link-layer solutions, and modifications and enhancements to TCP's algorithms, we have been able to comprehensively solve many observed problems to efficient wireless data transport. Our research demonstrates the flexibility of the TCP/IP protocol architecture and strongly discourages the need to design a new protocol framework to combat the problems observed in the heterogeneous networks investigated in this thesis. Thus, our solutions enable a variety of wireless technologies to be successfully and efficiently integrated into the Internet in an incremental fashion.

1.3 Preview: Three Fundamental Challenges

The challenges to efficient reliable data transport over networks composed of wired and wireless links arise primarily because of network heterogeneity and different characteristics of the underlying wireless technologies. These lead to problems that greatly degrade the performance of protocols like TCP in networks with wireless links.

There are three key independent factors that affect the performance of reliable data transport in heterogeneous wireless networks:

1. The preponderance of packet losses due to wireless bit-errors and user mobility.
2. Asymmetric effects and latency variation due to adverse interactions between media-access protocols and TCP.
3. Small transmission windows due to low channel bandwidths.

While TCP adapts well to network congestion, it does not adequately handle the vagaries of wireless media. In this thesis, we analyze these problems in detail and solve them, using a combination of link-layer techniques and modifications and enhancements to TCP at the sender and receiver. The rest of this section summarizes the problems caused by these factors and outlines our solutions to them.

1.3.1 Challenge #1: Wireless Bit-Errors

TCP performance in many wireless networks suffers because of packet losses induced by wireless bit errors, which occur in bursts because of the nature of the wireless channel. Unfortunately, TCP wrongly attributes these losses to network congestion because of the implicit assumptions made by its congestion control algorithms today. This causes the TCP sender to reduce its transmission window in response and often causes long timeouts during loss recovery that keep the connection idle for long periods of time. The result is degraded end-to-end performance. In addition, packet losses that occur due to user mobility cause the TCP sender to remain idle for long periods of time even after the handoff is completed, resulting in unacceptably low throughput [25].

Traditional approaches to addressing the bit-error problem either operate solely at the link-layer, or split the end-to-end transport connection in two. Both these approaches are fraught with problems.

For example, pure link-layer protocols often interact adversely with TCP's loss recovery and timer mechanisms [38, 14]. Split-connection protocols attempt to shield the sender from the wireless link by explicitly terminating the wired connection at the base station and using a separate transport connection over the wireless link [149, 9, 81]. However, they do not preserve end-to-end semantics because data may be acknowledged to the sender even before it reaches the receiver, complicate handoff procedures because they involve hard state in the network, and do not usually provide the best possible performance.

Our approach to solving this problem, called the *Berkeley Snoop protocol [16],* exploits *cross-layer protocol optimizations* to improve performance. In addition, we design a mechanism called *Explicit Loss Notification (ELN)* that can successfully distinguish between congestion and channel error-induced packet losses to substantially enhance end-to-end performance. The Snoop protocol uses only *soft state* [31] at agents in the network, which is periodically refreshed upon the arrival of data segments and ACKs. It also leverages TCP's reliability messaging, without generating its own independent protocol messages.

For data transfer *to* a mobile host, the protocol deploys a snoop agent at the base station that caches unacknowledged TCP segments destined for the mobile host. The snoop agent uses the loss indications conveyed by duplicate TCP ACKs and locally maintained timers to retransmit lost data from the base station. The agent also *suppresses* duplicate ACKs corresponding to wireless losses from the TCP sender, thereby preventing unnecessary congestion control invocations. Since local retransmissions are based primarily on TCP ACKs and duplicate TCP ACKs are suppressed from the sender, we call this a *transport-aware reliable link protocol* [14]. It is a cross-layer optimization where a link-layer agent is aware of transport messaging and semantics.

For data transfer *from* the mobile host, we use a technique based on ELN to achieve good performance. The base station uses queue length information to distinguish congestion from corruption and passes this information to the mobile host's TCP using ELN as an in-band control signal. Thus, the mobile TCP sender can perform retransmissions decoupled from congestion control, which makes it a *link-aware transport protocol*.

Performance measurements of different workloads show throughput improvements between a factor of two and twenty across a range of bit-error rates for transfers in either direction, measured

over a Lucent WaveLAN-based wireless LAN [145]. We present the results of experiments over an indoor installation in the Bay Area Research Wireless Access Network (BARWAN) [36], as well as over an outdoor installation in U.C. Santa Cruz's Reinas wireless network [85]. Chapter 4 presents the details of the Snoop protocol and performance measurements. In Chapter 5, we describe several experiments that precisely explain the reasons for the observed performance, demonstrate the benefits of cross-layer optimizations and compare our approach to several other schemes.

1.3.2 Challenge #2: Asymmetric Effects and Latency Variation

The second challenge arises due to the asymmetric nature of many wireless communication systems. An example of a network that exhibits asymmetry is a wireless cable modem network [24], where available bandwidth in the forward direction is much larger than that in the reverse path (a dialup line or low-bandwidth wireless link). Such *bandwidth-asymmetric* networks are becoming increasingly popular because of economic reasons and predominantly asymmetric Web workloads.

Latency asymmetry occurs when latencies in opposite directions are different and variable, as in many packet-radio networks. A key problem observed in many multi-hop packet radio networks is the large and variable latencies that arise due to the reliable link-layer (for error control) and MAC (for media access) protocols used in these networks. These variable latencies lead to ACK queueing at radio nodes and to highly variable round-trip times at the TCP sender. To ensure that packets currently in transit do not get retransmitted prematurely, TCP takes into account both the smoothed round-trip time estimate and the linear deviation in its timeout algorithm [71]. Thus, timeouts in these networks result in long idle periods on the order of several seconds.

Since protocols like TCP rely on timely ACKs for good performance, the capabilities and characteristics of the reverse path used for ACKs have the potential to seriously affect performance in asymmetric networks [70, 78, 13]. Our contributions to solving this problem include proposing a formal definition of asymmetry with respect to TCP performance in terms of how the characteristics of one direction affect performance in the other, and classifying different types of asymmetry into bandwidth, latency, media-access and loss-rate asymmetries. We then design and implement both end-to-end and link-layer solutions including *ACK filtering, TCP sender adaptation* and *ACK reconstruction* to overcome the adverse effects of asymmetry on TCP performance [13]. We evaluate the effectiveness of these algorithms under uni-directional and Web-like workloads in Hybrid

Networks' bandwidth-asymmetric wireless cable network and in Metricom's Ricochet packet radio network, which displays latency asymmetry.

The fundamental reasons for degraded TCP performance over asymmetric networks are the adverse interactions between TCP data and ACKs. In particular, the reliance on timely ACKs for the sender to smoothly transmit data hampers end-to-end performance. This observation that imperfect and variable ACK feedback degrades TCP performance regardless of the specific network technology motivates our two-pronged solution approach. We first develop techniques to reduce the frequency of ACKs to reduce adverse interactions between data and ACK flows. But minimizing these interactions alone is not enough, as TCP transmissions are clocked by arriving ACKs. Thus, we develop techniques to handle the reduced frequency of ACKs. Together, these techniques comprehensively solve the problems posed by asymmetry.

We use ACK Filtering (AF) to reduce the number of TCP ACKs. AF is a transport-aware link-layer technique that operates at the router feeding the constrained reverse link used by TCP ACKs. Since TCP ACKs are cumulative, later ACKs completely subsume the information contained in earlier ones. When a later ACK is enqueued, all earlier ones belonging to that connection are purged from the queue.

Sender Adaptation (SA) and ACK Reconstruction (AR) are two independent techniques to handle this reduced ACK frequency. The first is an end-to-end scheme where the sender increases its congestion window depending on the number of bytes acknowledged, rather than on the count of the number of ACKs (which is what TCP does today). In addition, the sender also *regulates* the transmission of packets by performing rate control based on its estimate of the current window size and round-trip time. This prevents bursts of packets from being sent out, thereby reducing congestion along the forward path. Together, these techniques reduce the reliance on timely ACK feedback that today's TCP senders rely on.

ACK Reconstruction is a transport-aware link-layer scheme that involves no changes to end-host TCP implementations and accomplishes the same results as SA. This scheme deploys a soft-state agent called the ACK reconstructor at the other end of the constrained reverse link. The reconstructor intersperses TCP ACKs whenever it observes large gaps in the ACK sequence, enabling the

sender to obtain consistent and regularly-spaced ACK feedback. It does not spuriously generate any new ACKs, thus maintaining the connection's end-to-end correctness.

1.3.3 Challenge #3: Low Channel Bandwidths

The bit rate commonly available between distant hosts in the Internet today is often limited and time-varying [17]. This is especially true in many wide-area wireless networks where maximum bandwidths (< 500 Kbps) are often orders of magnitude lower than their wired counterparts. In addition, today's dominant TCP application, the World Wide Web, uses short TCP connections because most Web transfers are short. For any connection, the average TCP transmission window size is roughly equal to the product of the available bandwidth and round-trip delay. This is because in the steady state, the sender sends a window's worth of data per round-trip and the ratio of these two is the available bandwidth. When the available bandwidth is low, so is the transmission window. When connections are short, the sender's window obviously does not grow beyond the length of the connection.

Small TCP transmission windows prevent the sender from recovering from losses without incurring expensive timeouts that keep the connection idle for long periods of time. As a result, end-to-end throughputs are significantly lower than the already-low maximum channel bandwidth: TCP transfers in these situations yield throughputs that are only a fraction of what is achievable. Thus, a key challenge is in enhancing TCP's loss recovery algorithms when low available bandwidths lead to small transmission windows.

Based on an extensive analysis of packet traces collected from the Atlanta Olympic Games Web server run by IBM [104], we show that not only are current loss recovery techniques grossly inadequate at preventing sender timeouts, but that proposed enhancements like Selective Acknowledgments (SACK) [92] are not likely to significantly change this because typical Internet transmission windows are not larger than a few segments. Based on the results of our analysis, we develop an *enhanced loss recovery scheme* for TCP that would have prevented about 25% of the timeouts in the particular packet trace, had it been in use, and thereby significantly improved end-to-end performance [15].

In this scheme, the sender enters an *enhanced recovery* phase when windows are small and transmits new probe segments when duplicate ACKs arrive for earlier segments. Upon successful

receipt of these new segments, additional duplicate ACKs are generated and reach the sender. When a threshold number of these arrive at the sender, it can accurately deduce that the original segment was indeed lost (and not temporarily reordered in the network) and go ahead and retransmit the segment *without* incurring a timeout. Because packet reordering is a common occurrence on many Internet paths [111], the sender must not retransmit a segment before enough later segments have reached. By sending new segment probes in accordance with the *conservation of packets* principle [59], the sender elicits additional duplicate ACKs from the receiver to distinguish between genuine loss and reordering. Packet conservation ensures that a new segment is sent only after one has been received, since it is triggered by a duplicate ACK; thus, the soundness of TCP's congestion control is preserved.

Using this scheme, we are able to realize a TCP that times out only under persistent congestion—(i) when an entire window is lost and (ii) when a retransmission or a new probe segment is lost—for which a timeout is indeed the correct action. In addition, the enhanced recovery scheme is orthogonal to the information provided by SACKs and can be used in conjunction with SACKs for better overall performance. The resulting protocol improves TCP performance for Web-like workloads on the Internet and substantially improves performance over wide-area wireless networks. Chapter 7 presents the design and implementation details of these enhancements and the results of performance experiments.

1.4 Contributions and Structure of Thesis

In this thesis, we analyze the problems posed by the above challenges and design, implement and evaluate solutions to them that result in significant improvements in end-to-end performance. Because there is tremendous heterogeneity and diversity in wireless access networks, and because there is enormous variation in their characteristics, there is no single panacea that cures all the observed problems. We recognize this and develop a suite of solutions and techniques to combat these problems, that together comprehensively solves these problems. The components of this solution suite include the following protocols and algorithms:

- **The Berkeley Snoop protocol:** This protocol combines a transport-aware link protocol and a link-aware transport protocol, which improves performance in networks composed of wired and single-hop cellular wireless links. The key idea here is the deployment of a soft-state agent at the base station to perform local loss recovery and shield the TCP sender from the vagaries of the wireless link.
- **Explicit Loss Notification (ELN):** ELN is a general framework by which receivers, base stations, or other elements in the network can inform the TCP sender of losses that occur for reasons other than congestion. When combined with algorithms to distinguish congestion losses from corruption, this framework provides a powerful way by which TCP senders can separate congestion control from loss recovery and recover from non-congestion-related losses without invoking congestion control.
- **Asymmetric TCP enhancements:** We present a suite of end-to-end and transport-aware link-layer algorithms to overcome the adverse effects of asymmetry. These involve techniques to reduce the frequency of ACKs and techniques to handle reduced ACK feedback. They improve performance in both packet radio networks that exhibit variable latencies and ACK queueing and in bandwidth-asymmetric networks with low-bandwidth ACK paths.
- **Enhanced TCP loss recovery:** This is an enhancement to TCP's data-driven loss recovery mechanism that improves performance by reducing the number of timeouts when transmission windows are small.

In addition to solving the specific problems in wireless networks discussed above, we also derive a set of general principles and lessons to design transport protocols for heterogeneous networks. These include:

- **Cross-layer protocol optimizations:** The principle of layering [151] has served us well for years as a good way to design network protocols and reason about them. For over a decade, network researchers and practitioners have recognized that a strict layering hierarchy results in an inefficient end-host implementation of a protocol suite, and in response have developed implementation techniques such as Integrated Layer Processing (ILP) [33]. In our work, we develop and refine the idea of *cross-layer protocol optimizations* as a design methodology by which information is exposed across multiple layers of the conventional protocol stack to optimize

end-to-end performance. For example, the Snoop protocol deploys a transport-aware link agent, which takes advantage of the reliability mechanisms of the transport layer and does not generate its own independent messages. It also suppresses certain duplicate TCP ACKs from the sender. Similarly, using ELN, the TCP sender is able to distinguish between congestion and corruption-induced losses. Another example is the design of the ACK filter and ACK reconstructor agents to combat the adverse effects of asymmetry; these agents perform TCP-aware connection-specific actions to achieve better end-to-end performance.

We believe that this is a powerful technique that can be generalized to many other situations where protocols are being designed with good network performance in mind. This design principle is orthogonal to ILP, which proposes that multiple layers be coalesced to achieve efficient end-host implementations; in contrast, cross-layer optimizations preserve the principle of layering, but expose certain critical parts of different layers to each other for improved network performance. Chapter 5 discusses this principle in greater detail.

- **Soft-state:** A second general design principle that we use in the design of many of the protocols presented here is the notion of soft-state agents in the network. To enhance performance, agents are often deployed in the network and have state associated with them. However, this state is *soft* [31], i.e., it is not essential for the correct end-to-end functioning of the connection. It is also periodically refreshed by the arrival of new packets and ACKs, which implies that even if it gets corrupted, the correctness of the connection is not compromised. This is in direct contrast to protocols that deploy hard-state agents in the network, such as split-connection methods for wireless TCP. Chapters 4, 5 and 6 discuss this principle in more detail in the context of specific network agents, such as the Snoop protocol, ELN-enhanced TCP and TCP-aware link-layer solutions for asymmetric networks.

  In our work, the design of soft-state agents has been guided by the end-to-end argument in systems design [127]. These agents must be viewed as performance optimizers, are not essential for correctness, and do not compromise end-to-end semantics. Another guiding principle has been the conformance to the IP service model; in particular, we use the fact that packets can be dropped or duplicated in an IP network to great advantage, by suppressing certain types of ACKs, generating local retransmissions, etc.

- **Data-driven loss recovery:** This applies in general to the design of network protocols that attempt to achieve reliability using retransmissions. There are two methods by which reliable transport protocols decide that packets are lost and then retransmit them: using timers (*timer-driven retransmissions*) and using information from later data packets that have reached (*data-driven retransmissions*). Because it is important for transport protocols not to retransmit packets that might still be in transit, retransmission timers are necessarily conservative [150]. Thus, to achieve low recovery latencies and avoid long idle periods that timeouts impose, it is important to make loss recovery as data-driven as possible. We achieve this in the Snoop protocol by taking advantage of the information provided by duplicate TCP ACKs (and SACK, when available) as well as in the enhanced TCP loss recovery for small window connections.

The rest of this thesis is organized as follows. In Chapter 2, we discuss background material in the area of reliable transport protocols and describe related work in improving transport performance over wireless networks. In Chapter 3, we describe the details of our wireless testbed and the experimental networks used in this work. We also discuss the details of our research methodology, which includes analysis, design, simulation, implementation and performance evaluation.

Chapters 4 through 7 form the core of this thesis. Chapter 4 describes the Berkeley Snoop protocol, designed to combat the problems of wireless bit-errors and provide good performance in the presence of wireless corruption.

Chapter 5 argues that exploiting cross-layer protocol optimizations and exchanging information across protocol layers is a powerful technique in the design of protocols to optimize performance. In this chapter, we present the results of several experiments to understand the precise reasons behind the good performance provided by the Snoop protocol and quantitatively compare it to various alternatives.

In Chapter 6, we address the challenges posed by asymmetry, starting with bandwidth asymmetry using Hybrid's wireless cable network as the experimental network. We then focus on understanding the problems posed by the interactions between media-access protocols and TCP in packet radio networks, using Metricom's Ricochet network as the testbed. After discussing these problems, we describe the details of our solutions to them.

In Chapter 7, we describe an enhanced loss recovery algorithm that greatly improves loss performance when TCP windows are small, the common case in the Internet today and in many wide-area wireless networks. We motivate this by a detailed analysis of packet traces collected from the Web server of the Atlanta Olympic Games. After describing the enhancements to TCP, we evaluate them over busy Internet and wireless paths.

Finally, in Chapter 8, we present a summary of our work and contributions. We conclude this chapter and this thesis with a discussion of general lessons learned about adaptive protocol design for heterogeneous networks and outline some directions for future research.

Anything but history, for history must be false.

— Sir Robert Walpole

[Remark to his son, who offered to read to him. *Walpoliana*, Vol. i.]

Chapter 2

Background and Related Work

The purpose of this chapter is to introduce the reader to related work in reliable transport protocols and reliable data delivery over wireless networks. We start with a description of the best-effort Internet service model, motivate the need for reliable transport, and discuss several reliable transport protocols proposed in the literature. In Section 2.2, we discuss the Transmission Control Protocol, TCP [116], the dominant reliable transport protocol in the Internet today. We follow this with a discussion of a number of recently proposed end-to-end enhancements to TCP in Section 2.3. In Section 2.4, we survey some key concepts in wireless networks. In Section 2.5, we discuss conventional approaches to tackling the problems caused by wireless bit-errors, highlighting their weaknesses. Sections 2.6 and 2.7 describe related work in transport over bandwidth-asymmetric and packet radio networks respectively. We conclude this chapter with a summary in Section 2.8.

2.1 Reliable Transport Protocols

Today's Internet is a *best-effort* network that does not guarantee reliable or ordered data delivery. Packets sent from an end-host are forwarded through a sequence of routers until they reach their destination, but might get lost, reordered or duplicated en route. While this permits a simple and

robust internal architecture and is responsible in part for the impressive scaling properties of backbone routers in the Internet, it leaves reliable data delivery to higher level protocols. Applications like the World Wide Web [18], file transfer [119], remote terminals [118] and electronic mail [117] that need *reliable* and *ordered* data delivery require a transport protocol to provide this functionality, freeing them of the need to achieve reliability on a per-application basis. Any reliable transport protocol must perform the following important functions:

1. *Loss recovery:* The major function of these protocols is to provide reliability and recover from lost packets. Most reliable transport protocols use some form of *Acknowledgment-Repeat-reQuest (ARQ)* retransmission scheme to recover from packet loss. Here, the receiver sends acknowledgments (ACKs) signifying successful receipt of data packets. The sender uses this information to deduce which packets are missing and retransmits them. There are three basic kinds of ARQ schemes. In *Stop-and-wait* ARQ, the transmitter sends a packet and waits for an ACK from the receiver. Upon receiving an ACK, it transmits the next packet. If the receiver sends a negative ACK (NAK) or if the transmitter times out waiting for the ACK, a retransmission occurs. While it is simple to implement, stop-and-wait is inefficient because its throughput is bounded by one packet per round-trip. *Go-Back-N (GBN)* protocols continually send data and receive ACKs spaced by a round-trip time. When a NAK arrives or a timeout occurs, the sender retransmits all packets sent thus far starting from the first missing one. The receiver implementation in these protocols is simple because no data buffering is required, but the protocol is inefficient because of the redundant retransmissions of already-received packets. *Selective repeat* protocols address this problem by having the receiver specify exactly which packets are missing (or conversely, which packets have arrived), so the sender can retransmit only the missing ones. These protocols are the most efficient among the three basic schemes, but are also the most complex to implement.

2. *Congestion and flow control:* Congestion control is the ability for the transmitter to recognize that the network is overloaded and respond to it by reducing its transmission rate. Congestion occurs when queues in the routers in the network are not empty; additionally, when a packet enters a full (or sometimes, non-empty) queue, packets are dropped. Flow control is the ability for the sender to recognize that it is overwhelming the receiver by sending data too rapidly and react to it. Continually overrunning the network or receiver buffers is not a stable situation because it leads to excessive losses, followed by retransmissions.

3. *Connection management:* Connection management involves the management of connection state and the machinery to initiate and terminate connections in the presence of packet loss, packet duplication, and router/host failures.

Several reliable transport protocols have been proposed in the literature for best-effort networks: Delta-t [144], NETBLT [32], VMTP [27], OSI/TP4 [105], XTP [137] and TCP [116]. In all these protocols, data items are identified by sequence numbers that are either byte-based or packet-based. These sequence numbers are used to detect losses, reordering and data duplication. Packet-based sequence numbers are simpler to implement but are less flexible because they often constrain the sender to use fixed-size packets. Byte-based sequence numbers indicate to the receiver the exact amount of missing data and permit the sender to precisely identify and retransmit lost data.

The rest of this section briefly surveys the key ideas in these protocols. Other than TCP, these are hardly used in the Internet today [141].

2.1.1 Delta-t

Delta-t [144] provides an end-to-end, stream-oriented, reliable transport service. Delta-t pioneered the concept of timer-based connection management and light-weight connections that are automatically established upon receipt of data and torn down after a timeout following the end of data transmission. Data is identified using byte-based sequence numbers and the receiver sends positive and optional negative ACKs signifying losses to the sender. In response, the sender implements a GBN retransmission strategy, retransmitting all missing data from the point of indicated loss.

2.1.2 NETBLT

The NETwork BLock Transfer protocol was developed for high-throughput bulk data transfer [32], especially over long-delay links. Connections using NETBLT are uni-directional and the unit of transmission is a *buffer,* several of which may be pipelined in transit as permitted by a flow control window. Transmissions are *rate-controlled,* where the number of buffers sent in unit time is controlled. NAKs and selective retransmissions are used for loss recovery, and a timeout mechanism causes all unacknowledged buffers to be retransmitted as a single block. Thus, while NETBLT has an efficient loss recovery mechanism, its weakness is the absence of a specified *adaptive* conges-

tion control scheme to set its rate parameters in response to network congestion. Furthermore, it is optimized for large transfers and does not perform well for short ones.

2.1.3 VMTP

The Versatile Message Transport Protocol (VMTP) was originally developed to provide communication for a networked distributed system [27]. The main focus of VMTP is on transaction-oriented, RPC-style communication that requires low latency for small data transfers. VMTP also has a streaming mode with rate control and selective repeat [28]. The other innovative aspect of VMTP was its support for multicast and call forwarding functions. However, it does not have an adaptive congestion control algorithm.

2.1.4 XTP

The eXpress Transport Protocol (XTP) was designed to operate efficiently over high-speed networks, with ease of VLSI implementation as the main objective [137]. It provides a variety of different services, including reliable bulk transfer, with implicit connection establishment/teardown, rate control, and selective retransmissions. It attempts to adapt its rate and parameters to the capabilities of the network and end-hosts.

2.1.5 OSI/TP4

A unique feature of OSI's TP4 [105] transport protocol is the ability to negotiate certain quality-of-service parameters such as expected throughput, loss rate, and delay. However, retransmissions are not data-driven; instead, all retransmissions are timer-driven, making loss recovery sluggish. OSI/TP4 also allowed an end-to-end connection to be split into many cascaded network connections with hop-by-hop reliability. It is unclear that this is a desirable property, since it adds enormous complexity in the internals of the network, but does not alter the need to perform end-to-end reliability. Viewed in terms of the end-to-end argument in systems design [127], this design choice is inappropriate since it does not usually lead to significant improvement in end-to-end performance.

An excellent survey of transport protocols for high-speed networks by Doeringer et al. appears in [40].

2.2 Transmission Control Protocol (TCP)

TCP is the de facto standard for reliable data transport in the Internet today. Measurements made in 1997 of the MCI backbone show that over 95% of all bytes, 90% of all packets and 75% of all flows use TCP [141]. This section discusses its salient features and highlights its main weaknesses.

While the original formal specification of TCP is in RFC 793 [116], numerous variants of it have been developed over the past several years, such as Tahoe, Reno, Vegas, etc. [135, 21]. This section discusses the Reno variant of TCP[1], which is the predominantly deployed version today. Section 2.3 discusses a number of recently proposed enhancements to TCP, such as Selective Acknowledgments.

2.2.1 Cumulative Acknowledgments

TCP is an ARQ-based reliable transport protocol that uses cumulative ACKs and byte-based sequence numbers for reliability. TCP provides a *fully reliable, in-order, byte-stream* delivery abstraction to the higher-layer application, which typically uses a socket interface [134] to interface with the transport layer. The basic unit of transmission is called a *segment,* which is a contiguous sequence of bytes identified by its 32-bit long start and end sequence numbers. The transmitted segments are smaller than or equal to the connection's maximum segment size (MSS), which is negotiated at the start of the connection.

A cumulative ACK from a receiver for byte k implies that all bytes less than k have been successfully received, and that a segment beginning at byte k has not yet arrived. During normal loss-free operation, the cumulative ACK sequence steadily increases as segments arrive in sequence. In what follows, we describe how TCP performs loss recovery, congestion control, and connection management, and highlight weaknesses in its congestion control scheme.

2.2.2 Loss Recovery

When the TCP sender discovers that data has been lost in the network, it recovers from it by retransmitting the missing segments. TCP has two mechanisms for discovering and recovering from losses: *timer-driven retransmissions* and *data-driven retransmissions*.

1. Unless explicitly mentioned otherwise, "TCP" in this section implies the popular Reno variant.

Timer-driven recovery: When the TCP sender does not receive a positive cumulative ACK for a segment within a certain timeout interval, it retransmits the missing data. To determine the timeout interval, it maintains a running estimate of the connection's round-trip time using an exponential weighted moving average (EWMA) formula, $srtt = \alpha * rtt + (1-\alpha) * srtt$, where *srtt* is the smoothed round-trip time average, *rtt* the current round-trip sample, and α the EWMA constant set to 0.125 in the TCP specification. It also estimates the mean linear deviation, *rttvar,* using a similar EWMA filter, with α set to 0.25. A timeout occurs if the sender does not receive an ACK for a segment within $srtt + 4 * rttvar$ since the arrival of the last new cumulative ACK[1]. Furthermore, the retransmission timer is exponentially backed off after each unsuccessful retransmission. The details of the round-trip time calculations and timer management can be found in [71, 59, 135].

It is important to use the mean deviation in the timeout calculation to reduce the number of spurious retransmissions for data that are still in the network. Timer-driven retransmissions are coarse-grained and conservative, typically with a 500 ms timer granularity, which keep the connection idle for long periods of time. Ideally, they should be used only when losses happen because of *persistent* congestion, allowing the state of congestion to dissipate before the retransmission. For reasons of efficiency and performance, a reliable transport protocol should not rely on timeouts as the primary mode of recovery; rather, timers should be used conservatively and only as a last resort when other methods fail, since it is fundamentally impossible for retransmission timers to be accurate [150].

Data-driven recovery: TCP's data-driven retransmission mechanism uses a technique called *Fast Retransmission*. It relies on the information conveyed by cumulative ACKs and takes advantage of the receipt of later data segments after a lost one. Because ACKs are cumulative, all segments after a missing one generate *duplicate* cumulative ACKs that are sent to the TCP sender. The sender uses these duplicate ACKs to deduce that a segment is missing and retransmits it, as shown in Figure 2.1.

1. Thus, a timeout occurs roughly $2 * srtt + 4 * rttvar$ after the original transmission in the steady state, since a window's worth of ACKs arrive in one round-trip.

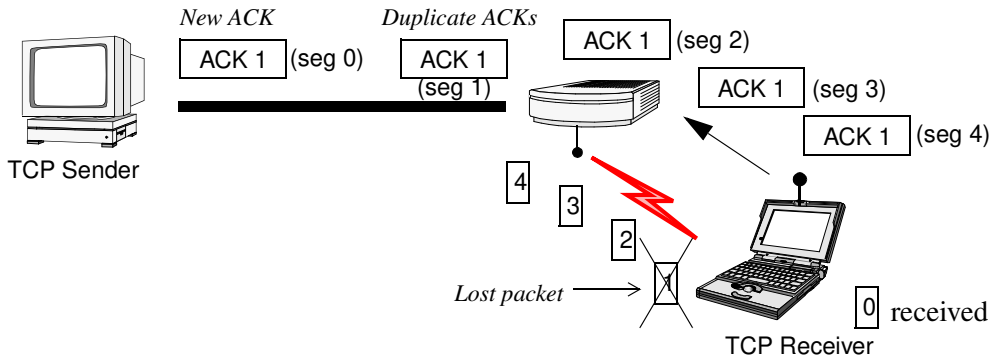

Figure 2.1 TCP Fast Retransmissions. When the receiver gets packet 0, it sends a new ACK. Then, when packets 1 through 4 arrive in succession, the cumulative ACK it sends remains at 1, which is the next packet it expects. Upon receiving three of these *duplicate* cumulative ACKs for 1, the sender retransmits packet 1.

However, the sender must not retransmit a segment upon the arrival of the very first duplicate ACK. This is because the Internet service model does not preclude the reordering of packets in the network; such reordering causes later segments to be received ahead of earlier ones, and triggers duplicate ACKs in the same way that losses do. Furthermore, the degree of packet reordering on the Internet seems to be increasing, according to studies by Paxson [111]. Thus, to avoid prematurely retransmitting segments, the sender waits for three duplicate ACKs, the current standard fast retransmit threshold.

This is followed by the *fast recovery* phase, where additional packets are transmitted after the sender is sure that at least half the current window has reached the receiver, based on a count of the number of received duplicate ACKs. Fast recovery ensures that a fast retransmission is followed by congestion avoidance and not by slow start. Since the arrival of duplicate ACKs signals to the sender that data is indeed flowing between the two ends, there is no reason to suddenly throttle the sender by invoking slow start.

Fast retransmissions are often not sufficient to recover from multiple losses in a single window, because all the cumulative duplicate ACKs arrive for the first loss in the window. This usually results in a coarse-grained timeout before the packet is retransmitted.

TCP currently makes the tacit assumption that *all* losses occur because of network congestion. Thus, coupled with either of these modes of retransmission is congestion control that reduces the

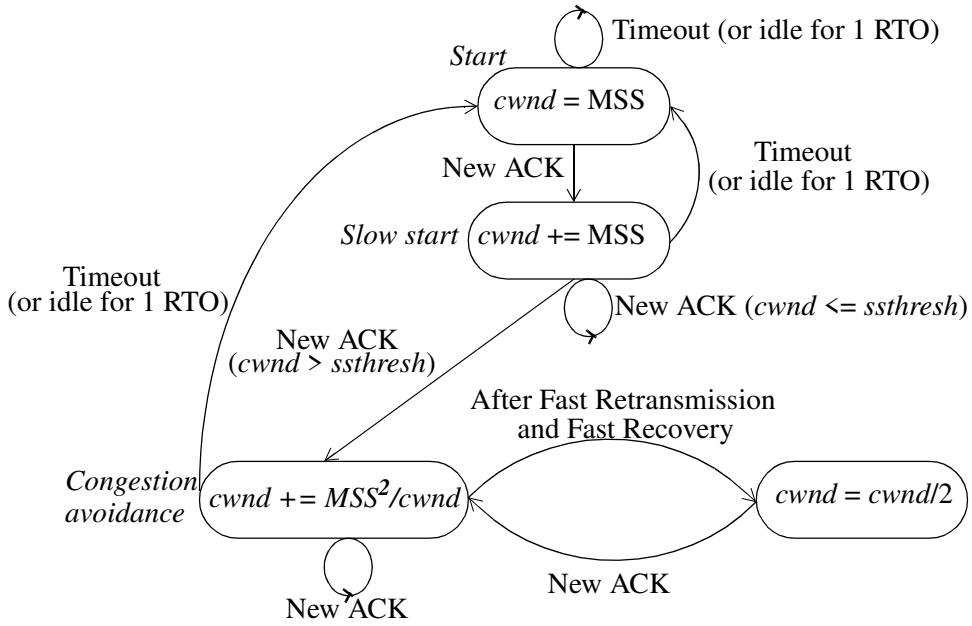

Figure 2.2 Dynamic evolution of TCP's congestion window. RTO is the retransmission timeout duration equal to *srtt + 4*rttvar*.

sender's transmission rate. The next section discusses how TCP manages congestion by probing for sustainable bandwidth and reacting to congestion.

2.2.3 Congestion Avoidance and Control

TCP's congestion management is based largely on Jacobson's seminal paper [59] and on the principles of multiplicative-decrease/additive-increase expounded by Chiu and Jain [30]. TCP uses a window-based algorithm to manage congestion, where the window is an estimate of the number of bytes currently unacknowledged and outstanding in the network.

The TCP sender performs *flow control* by ensuring that the transmission window does not exceed the receiver's advertised window size. It performs congestion control by using a window-based scheme, where the sender regulates the amount of transmitted data using a congestion window. When a connection starts or resumes after an idle period of time, *slow start* is performed. Here, the congestion window is initialized to one segment and every new ACK increases the window by one MSS. After a certain threshold (called the slow start threshold, *ssthresh*) is reached, the connection moves into the *congestion avoidance* phase, in which the congestion window effectively increases by one segment for each successfully transmitted window. In response to a packet loss, the sender

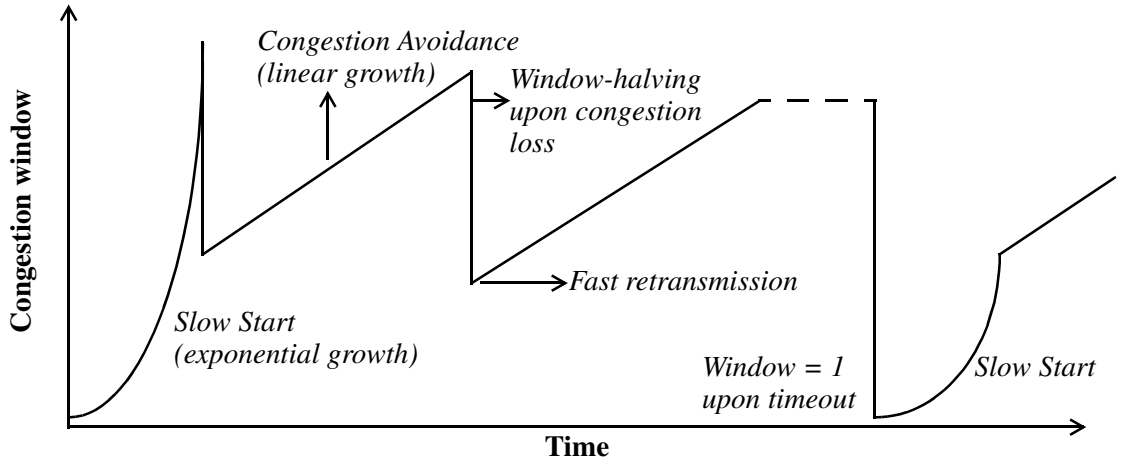

Figure 2.3 Dynamic time evolution of the TCP congestion window.

halves its congestion window; if a timeout occurs, the congestion window is set to one segment and the connection goes through slow start once again. This state evolution is shown in Figure 2.2 and the dynamic time evolution is shown in Figure 2.3.

In addition, *ssthresh* is set to half the value of *cwnd* at the time of loss, and at each stage slow start is performed until *cwnd* reaches *ssthresh*. At any point in time, the TCP sender ensures that the number of unacknowledged bytes is not larger than the smaller of the receiver's advertised flow control window and *cwnd*. In the steady state of the connection, the sender transmits a window's worth of data every round-trip, at a long-term rate equal to the bottleneck bandwidth. Since the ratio of window to round-trip delay equals the bottleneck bandwidth, TCP's congestion window size tends towards the connection's bandwidth-delay product. This is the number of outstanding bytes in transit in the steady state. Thus, it is clear that a low-bandwidth connection implies small transmission windows. In turn, this affects the fast retransmission mechanism because not enough duplicate ACKs arrive, leading to expensive timeouts.

Data transmissions are triggered and clocked by ACKs that arrive for previous segments; every time an ACK arrives, the transmission window shifts by an amount equal to the sum of the number of bytes acknowledged and the increase in *cwnd*, subject to the constraints imposed by the flow control window. This elegant notion of transmitting data based on incoming ACKs, called *ACK-clocking,* makes TCP a tightly-coupled feedback system and frees the sender from maintaining

explicit software timers for transmission. However, it also leads to significant performance degradation when ACK feedback is imperfect or variable.

2.2.4 Connection Management

TCP has a sophisticated connection initiation mechanism using a three-way handshake, where a SYN exchange occurs before the connection is established and data can flow. A connection is uniquely identified by the source and destination IP addresses and port numbers. Details of the connection setup, teardown and state transitions are orthogonal to our work (see [116, 135] for details).

2.2.5 Window-based vs. Rate-based Congestion Control

TCP's window-based congestion control algorithm is elegant because it does not need an explicit clock to pace its data transmissions, since ACK clocking ensures that new data segments are sent out triggered by incoming ACKs. However, it is well-known that several problems arise from using the sender's transmission window to perform both congestion and error control in window-based protocols such as TCP. These arise because of the tacit assumption that all losses are the result of network congestion, and the use of the same window for both loss recovery and congestion management. The alternate approaches proposed by several authors attempt to achieve a separation of congestion control and loss recovery by performing congestion avoidance using a *rate-based* scheme. The basic idea is that the sender controls the rate at which packets are transmitted into the network using a timer. That is, the ACK-clocking mechanism used in TCP, is avoided in favor of a mechanism that is more explicit and not window-based.

This separates the process of error control from congestion control, because loss recovery is no longer tied to the congestion window. Transport protocols in connection-oriented networks often use this method [40]. Other protocols that use this method include NETBLT [32] and XTP [137].

The fundamental problem with pure rate-based schemes is that all packet transmissions are triggered by timer events at the source. This results in the source becoming the transmission bottleneck in many circumstances, especially as network bandwidths and number of connections increase (such as at a busy Web server). Usually, rate-based flow and congestion control schemes rely on some sort of connection establishment phase at start-up, and either reserve bandwidth (e.g.,

in connection-oriented networks like Asynchronous Transfer Mode networks), or estimate bandwidth as in Keshav's packet-pair scheme [75] and adapt to it. When congestion occurs, the sender decreases its sending rate by a multiplicative factor (e.g., by halving its rate) to perform congestion control. The other problem with a pure rate-based scheme for congestion control is that it requires support from the internal routers in order to enforce the rate negotiated between the sender and receiver on a per-connection basis [63], or at least provide isolation between flows (as in the packet-pair scheme that requires routers in the network to perform fair queueing [37]).

2.3 End-to-end TCP Enhancements

Over the past decade, TCP has been tuned to work well in wired networks in the face of network congestion, reacting to congestion by reducing its rate, recovering from lost segments, and probing for bandwidth in a careful way. In this section, we survey some proposed enhancements to TCP that improve its ability to recover from losses in a timely manner and/or perform better congestion control.

2.3.1 Selective Acknowledgments (SACK)

It is well known that TCP performance suffers due to coarse-grained timeouts when multiple segments are lost in a single window because it uses only cumulative ACKs. There has therefore been recent[1] interest in adding *selective acknowledgments* (SACK) to the standard TCP specification to reduce the time it takes to recover from multiple losses in a window. Fall and Floyd [42] describe detailed simulation results highlighting the benefits of SACK in large delay-bandwidth product networks.

TCP Selective Acknowledgments can be implemented in many ways. One possible approach is to use Keshav and Morgan's SMART (Selective Mechanism to Aid ReTransmission) scheme, where the receiver communicates the segment number that just arrived in addition to the cumulative ACK, whenever it sends a duplicate ACK. We elaborate on this scheme in Chapter 5 in the context of wireless links, and discuss its performance. A second approach, described in RFC 2018, is cur-

1. Actually, RFC 1323 [61] written in 1992 outlined a SACK mechanism, but some unresolved technical details delayed its acceptance as a standard for several years. RFC 2018 resolves these details.

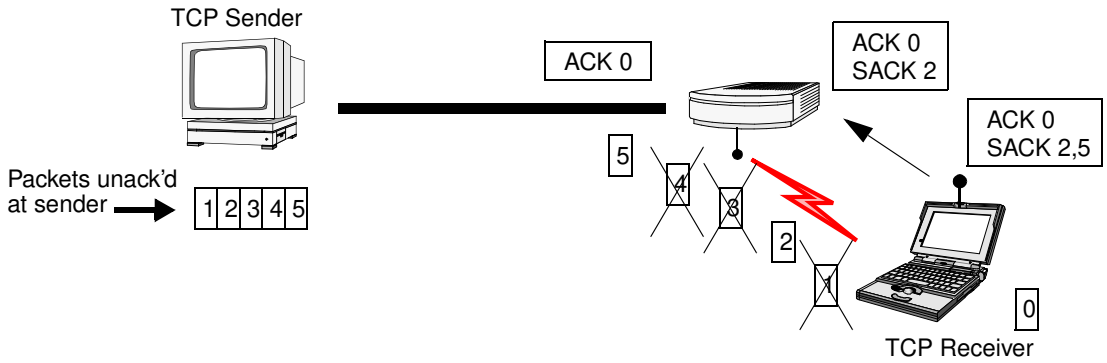

Figure 2.4 TCP Selective Acknowledgments according to RFC 2018. When segments 2 and 5 reach the receiver, the receiver sends two duplicate cumulative ACKs with the displayed SACK blocks to the TCP sender.

rently on track to become an Internet standard. In this scheme, the receiver reports up to three of the last received, out-of-order, maximal contiguous blocks of data, in addition to the cumulative ACK, so that the sender can accurately deduce which segments have reached the receiver (Figure 2.4). The information reported by the receiver in this scheme is a superset of the information reported in SMART. By reporting up to three blocks of data, it ensures that every block is reported at least thrice to the sender; this makes it robust to the loss of some SACKs on the reverse path.

While the RFC standardizes the format and mechanism used by the receivers, it specifies little about what the sender should do in response. The sender must ensure that appropriate congestion control actions continue to be taken as in TCP Reno. In addition to more responsive loss recovery, SACKs allow the sender to perform better congestion management since it now has better information about how many bytes have cleared the network pipe on each SACK arrival.

Chapter 7 describes the details of our SACK implementation and the sender mechanisms that use SACK information. It also describes the limitations of SACK when transmission windows are small, and presents a novel approach that significantly improves loss recovery when losses occur in a small window.

2.3.2 Newreno

Hoe's Newreno[1] modification to TCP Reno reduces the number of timeouts incurred by the TCP sender when multiple losses happen in a transmission window [55]. When multiple losses occur in

TCP Reno, a fast retransmission occurs after three duplicate ACKs arrive. Later duplicate ACKs are ignored until half the window is acknowledged, after which fast recovery sends a new segment for every incoming duplicate ACK. Now, when a new ACK arrives after the successful fast retransmission, its value will be within the original window (recall that there were multiple losses in the original window). In many cases TCP Reno would time out for this second loss if the second loss is "close" to the location of the first one, because the sender's window would now have been reduced to half its original value and the sliding window not have shifted the original window to beyond the original right edge.

In Newreno, however, the sender *remains* in fast recovery when this happens, when the new ACK arriving after a fast retransmission is *partial*. A partial ACK is defined as a cumulative ACK that does not acknowledge the original window completely. By remaining in fast recovery, the sender continues to send segments new segments, which would elicit more duplicate ACKs and eventually trigger another fast retransmission without incurring a timeout.

2.3.3 Forward Acknowledgments

Mathis and Mahdavi's TCP with Forward Acknowledgments (FACK) [91] uses SACK information at the sender to perform better congestion management. Rather than assume that an MSS-worth of data has been received by the receiver on every duplicate ACK, FACK calculates this precisely using information from the SACK fields. It explicitly maintains an "available window" variable, *awnd*, that keeps track of the number of bytes that are unacknowledged, to perform better congestion control. As long as *awnd* is smaller than *cwnd*, the sender is permitted to transmit more data.

2.3.4 NetReno

Concurrent with our work, Lin and Kung [82] discuss some mechanisms for improving TCP loss recovery and propose some modifications to TCP. They argue that their modifications are more sensitive to network conditions than current TCP is, motivating the name NetReno (for "Network-Sensitive Reno"). Some of our solutions in Chapter 7 on improving TCP loss recovery are similar to some of their independently proposed modifications.

---

1. To the best of our knowledge, Hoe's original paper did not use the term "Newreno." We saw it first in an earlier version of Fall & Floyd's paper [42].

2.3.5 Explicit Congestion Notification

In Section 2.2, we noted that TCP assumes that all losses signify congestion. In addition, TCP[1] has no other way of deducing that the network is congested, or that congestion is imminent. Deriving from earlier work in the DECBit scheme by Ramakrishnan and Jain [122], Floyd investigated the addition of Explicit Congestion Notification (ECN) to TCP [45].

When a gateway en route is congested or close to congestion, it sets a bit in the packet header and forwards it on. A *Random Early Detection (RED)* [47] gateway in marking mode is apt for this. A RED gateway detects incipient congestion by tracking the average queue size over a time window in the recent past. If the average exceeds a threshold, the gateway selects a packet at random and probabilistically marks it, by setting the *Explicit Congestion Notification* (ECN) bit in the IP packet header [45, 123]. This notification is echoed to the sender of the packet by the receiver. For TCP, this echo is piggybacked on an ACK. Now, when the sender receives and ACK with ECN, it multiplicatively reduces its congestion window, as it would if a packet loss had occurred. Thus, this mechanism with explicit support from gateways allows TCP to perform *proactive* congestion control, over and above the *reactive* one triggered by packet loss.

The ECN mechanism and the reactions of TCP to ECN are currently in the Internet Engineering Task Force (IETF) standards track [123].

2.4 Wireless Networks

The use of radio waves to communicate information has been known for over a century now due to pioneering work by inventors like Marconi, De Forest, and Armstrong. Starting from Marconi's demonstration of Trans-Atlantic radio communication in 1901, the field has grown by leaps and bounds to where it is today [80]. This section surveys today's state of wireless data technology.

The past three decades have seen numerous research and commercial efforts in building and deploying systems to transmit digital data over wireless media. In the late 1960s and early 1970s, the ALOHA project led by Norm Abramson at the University of Hawaii investigated packet-

1. Other recent proposals like TCP Vegas [21] perform proactive congestion control by comparing achieved with effective throughput, using the ratio *window/delay* to calculate throughput.

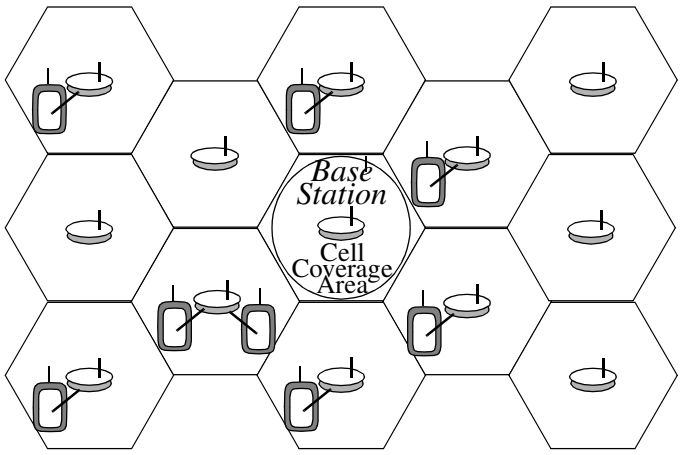

Figure 2.5 Schematic arrangement of cells in a cellular wireless network.
(Reprinted with permission from [129]).

switched networks over fixed-site radio links [1]. The contention resolution protocols developed in ALOHANET laid the framework for later Carrier Sense Multiple Access (CSMA) protocols for channel access. In 1972, the U.S. Defence Advanced Research Projects Agency (DARPA) launched the Packet Radio Program and its successor program on Survivable Adaptive Networks (SURAN) to study issues in packet radio networks such as channel access, link protocols, and routing [65, 79]. However, despite this work, wireless data communication remained largely a laboratory curiosity for several years.

In the 1990s, this has changed dramatically. Advances in hardware technology have enabled cheap and portable wireless devices for data communication. This, coupled with the rapid expansion of the Internet and the World Wide Web, has resulted in a variety of wireless technologies becoming commercially available.

The rest of this section surveys the key architectural ideas in wireless systems design and discusses some properties of wireless media. We then describe the pressing problems that hamper usability and degrade performance, as well as current solutions to these problems.

2.4.1 Cellular Network Topology

Many wireless and mobile networks are organized in a *cellular topology.* These topologies are composed of a wired, packet-switched, backbone network and a wireless network. The wireless network is organized into geographically defined cells, with a control point called a *base station*

(BS) in each of the cells, as shown schematically in Figure 2.5. These base stations are also directly connected to the wired network, routing packets between the wireless and the backbone network as shown in Figure 2.6. A mobile host (MH) receives data from a fixed host (FH) in the Internet routed via the BS of the cell it is currently in. As the MH moves between wireless cells, the task of forwarding data between the wired network and the mobile host must be transferred to the new cell's BS. This involves updating routing information in the wired infrastructure to reflect movement, and is known as a *handoff* [57, 58, 129, 100, 25]. The IETF standard for mobility is the Mobile IP protocol [112].

It is important for several applications and higher-level protocols that handoffs be as seamless as possible, incurring low latencies and causing little or no packet loss [25, 131, 22]. Numerous schemes have been proposed in the literature and several systems have been deployed to achieve these goals [e.g., 131, 50, 2, 26].

2.4.2 Wireless Technology Overview

Wireless data networks typically operate in the Radio Frequency (RF) (3 KHz to 300 GHz) or Infrared (IR) (1000 GHz to 30000 GHz) frequency range. IR offers high speeds over limited distances. Experimental research IR networks that offer up to 50 Mbps over a few meters [66], while commercial, IR-based wireless LANs have maximum bandwidths between 1 and 10 Mbps today. The IR frequency range is very close to that of visible light, which precludes transmission through walls or floors. Thus, while its range is more constrained than its RF counterparts, it offers a much higher degree of frequency reuse. The existence of impenetrable objects and reflections often causes IR dead spots in rooms, which in turn causes packet loss. The Infrared Data Association (IrDA) [6] has standardized the lower layer protocols and interfaces for IR devices, which is almost certain to be a part of every portable computer and handheld device in the future. Commercial IR-based LAN products available today include IBM's Infrared wireless LAN and InfraLAN.

In contrast to IR, RF can usually penetrate walls, which makes it an ideal technology for networks that must breach walls and floors. However, compared to IR, RF devices typically suffer from higher levels of interference and noise, which degrade the received signal and cause bit-errors. Because a large number of users in the same frequency band can degrade overall usability and performance, the use of different wireless frequencies in the United States is governed by the FCC and

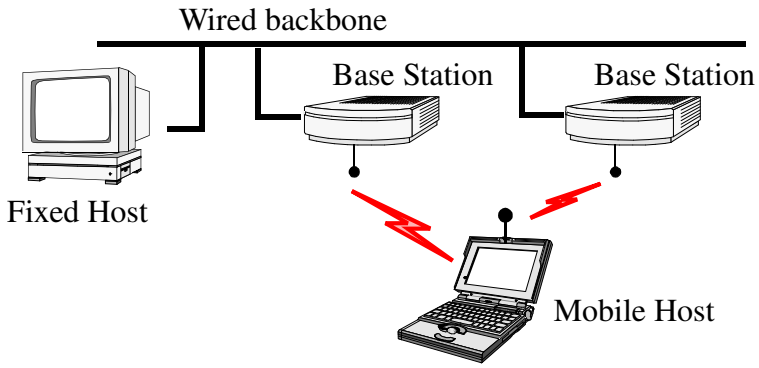

Figure 2.6 Network topology of a cellular wireless network. At any instant, one of the base stations routes packets between the wired backbone and the wireless network to the MH.

licenses are required to operate devices in many frequencies. An exception to this are the *unlicensed bands* discussed at the end of this section.

For several years, the application of RF to data communication exploited *narrowband* technology. Here, the input signal is modulated over a constant frequency wave, called the *carrier wave*, and the resultant signal is transmitted through the sender's antenna. When it reaches the receiver (often tainted by interference or noise), the receiver picks up this signal through its antenna, demodulates it, filters out the carrier frequency, and attempts to estimate the signal that was transmitted based on certain *decoding rules*. An error could occur in the decoding process if the level of noise or interference is too high—this manifests itself in the form of corrupted bits, which higher-level data protocols or applications must handle.

One of the major problems with narrowband systems that makes it vulnerable to interference is its lack of *frequency diversity*, since the entire carrier is forever concentrated in a single frequency. *Spread spectrum* [128] is a technique to combat this problem. Here, the transmitted signal is spread over a broad frequency band, which makes it more robust to interference or jamming and more secure against eavesdropping.

There are two common kinds of spread spectrum techniques: *direct-sequence* and *frequency-hopping*. In direct-sequence spread spectrum, the input signal is transmitted simultaneously over a broad range according to a pre-assigned code. In frequency-hopping spread spectrum, the transmitter sends data on one narrowband frequency for a short amount of time, then jumps to another narrowband frequency, using a pseudo-random hopping sequence. The receiver synchronizes with the sender's hopping sequence and tunes in to those frequencies to receive and decode the signal.

While frequency-hopping is more robust to sources of interference than spread-spectrum, maximum data rates are typically lower. Thus, it is not as suitable for high-speed wireless LAN access as for slower, wide-area links.

Examples of RF-based wireless LAN products include Solectek's AirLAN, Aironet's ARLAN [5], Motorola's Altair Plus-II, Proxim's RangeLAN products [120], and Lucent Technologies' WaveLAN [145]. Most of these operate in the unlicensed Industrial, Scientific and Medical (ISM) bands at 915 MHz and 2.4 GHz that have been set aside by the FCC for experimental purposes. In our work, we experimented with Lucent's WaveLAN product operating in the 915 MHz range. In this frequency range, products have to follow the "etiquette rules" for transmission mandated by the FCC [43].

Today's wireless LAN technologies typically provide peak link bandwidths in the 1 to 4 Mbps range. In the future, this is sure to increase; for example, the proposed HiperLAN network technology is rated at 20 Mbps. In Chapter 4, we "time travel" and evaluate TCP performance over these higher bandwidth wireless networks.

Examples of wide-area RF networks include Metricom's Ricochet network [98] that operates in the 915 MHz band and uses frequency-hopping spread spectrum technology, the Cellular Digital Packet Data (CDPD) network [121], and the GSM data service [99]. Peak link bandwidths in these outdoor networks today are typically between 10 and 100 Kbps, while future systems are likely to be in the 100-500 Kbps range (e.g., Metricom's proposed "Autobahn" network [126]). In Chapter 3, we discuss the details of the specific networks we experimented with in our work.

2.5 Wireless Bit-Errors and User Mobility

Figure 2.7 shows a typical loss situation over a wireless link. Here, the TCP sender is in the middle of a transfer to a mobile host. At the depicted time, the sender's congestion window is 5 packets. Of the five packets in the network, the first two packets are lost on the wireless link. TCP tacitly assumes that all packet losses are the result of network congestion, which is an invalid assumption in networks that are error-prone due to wireless link corruption or loss-prone due to user mobility.

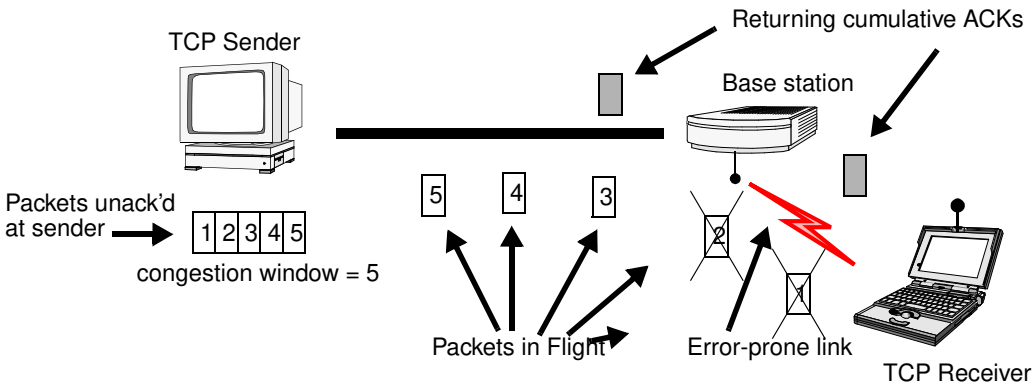

Figure 2.7 An example loss situation over a wireless link.

2.5.1 Reasons for Wireless Data Corruption

Wireless bit-errors occurs for a variety of reasons such as *path loss, fading, noise,* and *interference.* Path loss is defined as the ratio of signal power at the receiver to that at the sender and is a strong function of the distance between the receiver and transmitter. There are numerous path loss models that apply to different propagation mechanisms, such as free space, reflection, diffraction, and scattering. Rappaport [124] provides an excellent presentation of these empirical models. Typically, the path loss relationship has the form $L = P_r/P_t = K/f^2d^n$, where P_r is the received power, P_t is the transmitter power, f is the central frequency of transmission, d is the distance between transmitter and receiver, n is the path loss exponent, and K is a constant. In free space, $n = 2$, whereas it varies between 3 and 5 for typical outdoor environments [124]. Thus, the higher the path loss exponent, the higher the bit-error rate for the same transmitter power.

Shadow fading [84] occurs because of objects and obstructions in the path of the transmitted signal. Various studies have shown that the power (in dB) of signals subject to shadow fading exhibits a Gaussian distribution, with the random attenuation due to shadow fading changing as the mobile host moves in a region. *Multipath (or small-scale) fading* occurs because of interference between multiple versions of the same signal arriving at the receiver at different times due to reflections from surrounding obstacles. Unless they interfere constructively, an uncommon occurrence, this results in signal degradation because of destructive interference.

Wireless networks experience interference from a variety of other sources. For example, *co-channel interference* can occur due to frequency reuse, where users in nearby cells using the same fre-

quencies interfere with each other. Adjacent frequency channels in narrowband technologies cause interference too, although this can be ameliorated using spread spectrum techniques (Section 2.4.2) or *notch filters* [128], which selectively block a band of interfering frequencies.

Several techniques can be applied to radio and cell design to alleviate the severe problems that degrade wireless transmissions [49, 84]. However, despite these techniques, wireless bit-error rates that must be dealt with by higher-layer protocols and applications are often high and time-varying. Several studies have shown average bit-error rates to vary between 10^{-2} and 10^{-8} as conditions change [93, 102, 12].

Because TCP tacitly assumes that *all* losses are due to network congestion, it reacts to these corruption-induced losses by reducing its congestion window. In addition, the burst nature of wireless losses often leads to coarse-grained timeouts, especially in the absence of SACKs and when the transmission window is small. The result is unacceptably low throughput and under-utilization of available bandwidth.

2.5.2 Link-layer Protocols

Link-layer protocols are the conventional approach to handling error-prone wireless links [7, 69, 101]. They attempt to shield the TCP sender from the wireless link by performing local loss recovery and present it with a link with an effectively lower error rate. The main attraction of employing a link-layer protocol for loss recovery is that it fits naturally into the layered structure of network protocols. Traditional approaches include mechanisms such as Forward Error Correction (FEC), Acknowledgment-Repeat-reQuest (ARQ), and hybrid approaches that combine these two techniques [83, 7]. FEC is effective when errors occur at random and when the error characteristics of the channel are well-understood. Such schemes are not well-suited to links where the underlying bandwidths are not large (less than a few Mbps) and where the link latency has to be low. FEC schemes to tackle burst errors and long error sequences tend to use interleaving and increase latencies. In addition, if the error characteristics are not known in advance, the effective available bandwidth available for data communication after FEC overhead is only a fraction of the maximum. This makes ARQ and hybrid ARQ/FEC techniques attractive for low to medium bandwidth wireless systems, compared to pure FEC techniques.

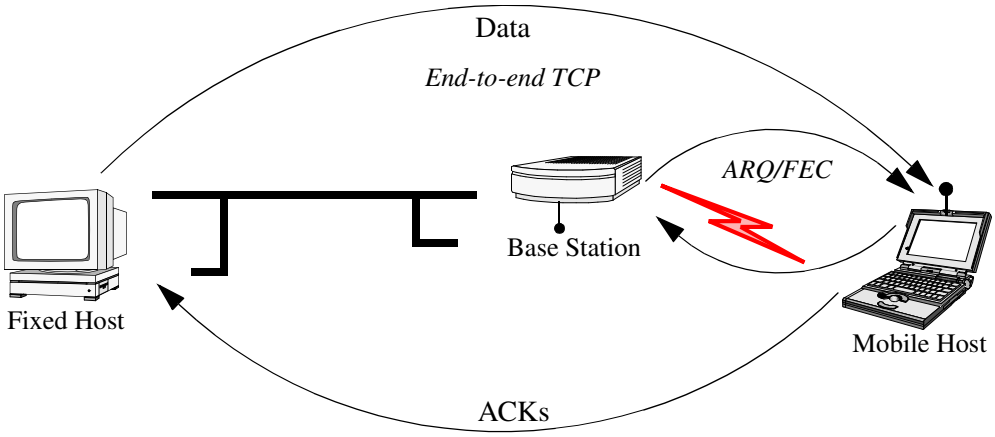

Figure 2.8 Reliable link-layer protocols perform local loss recovery over the wireless link, but could interact adversely with independent end-to-end mechanisms.

Standard link-layer ARQ schemes are usually stop-and-wait, Go-Back-N (GBN), or selective-repeat. Stop-and-wait ARQ is inefficient, especially when there are only a small number of wireless hops as is the case in cellular wireless networks, since it reduces the effective transmission window size to one. GBN is also inefficient because the transmission window does not slide, and an error anywhere in the window causes the retransmission of the entire window starting from that point. Selective repeat using a small window is the best scheme among these, and can be implemented using negative acknowledgments. It is important for the link-layer to be very simple and efficient to permit an implementation in hardware or firmware, and a link-layer protocol should not attempt to do too much at this layer (e.g., implementing a TCP-like protocol with almost complete reliability and in-order delivery) since that makes it very complex. Since higher layer protocols provide end-to-end reliability, it is wasteful to implement a very complex and sophisticated link-layer protocol—this is a consequence of the well-known end-to-end argument in system design [127], applied to network protocol design.

However, there are several occasions when end-to-end performance does not improve when conventional link-layer protocols are used. There are three chief forms of adverse interaction that could arise between independent reliable transport and link layers:

1. *Timer interactions:* Independent timers at both layers could trigger at the same time, leading to redundant retransmissions at both layers and degraded performance [38]. As a result, the transport sender is not shielded from the problems over the wireless link by the link-layer protocol. While this is a concern in general, TCP does not suffer from this problem in practice because of the coarse-grained and conservative timeout intervals.

2. *Fast retransmission interactions:* This arises when a link layer protocol achieves reliability by local retransmissions, but does not preserve the in-order sequential delivery of TCP segments to the receiver. Then, although local recovery occurs, the receipt of later segments causes duplicate ACKs from the receiver as well, leading to redundant sender fast retransmissions, sender window reduction, and reduced throughput [14]. This is a common occurrence in many link-layer protocols that do not preserve packet ordering. In this case, while local loss recovery is quick, the sender is not shielded from wireless losses because of the duplicate ACKs propagating back to it.

3. *Large round-trip variations:* If a link layer protocol is designed to be overly robust and attempt, for example, to provide a TCP-like service, it often results in long latencies and large round-trip time deviations at the TCP sender. This leads to long and conservative timeouts when congestion-related losses occur on the path. Other problems that arise include a large number of redundant retransmissions if a sender timeout occurs for a wireless loss [87], a problem observed in GSM networks running the RLP protocol [99]. The fundamental problem in this approach is that the link-layer protocol tries to do too much in terms of providing reliability.

Thus, we see that while link-layer protocols provide local loss recovery, they do not always shield the TCP sender from wireless errors. This leads to sub-optimal performance.

2.5.3 Split-Connection Protocols

In the mid-1990s, split-connection approaches gained in popularity as a way to improve TCP performance over error-prone wireless networks. They were motivated by the observation that the problems arose because the sender was imperfectly shielded from the error-prone link. To effect better shielding, split-connection protocols [e.g., 149, 10, 81, 143] split each TCP connection between a sender and receiver into two separate connections at the base station—one TCP connec-

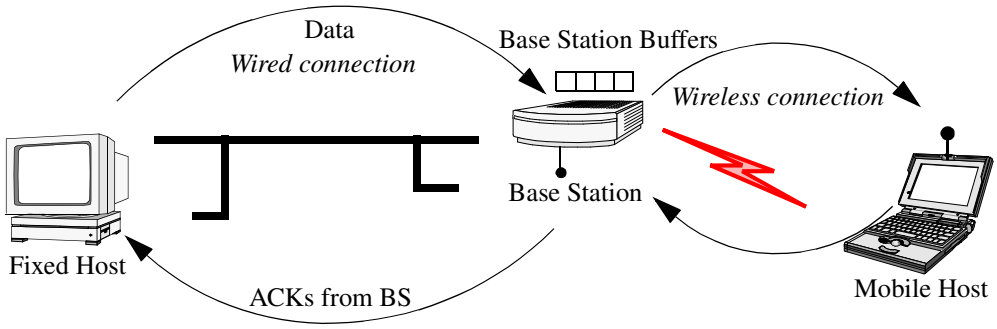

Figure 2.9 Split-connection protocols split the end-to-end TCP connection in two, with the sender's TCP terminating at the base station

tion between the sender and the base station, and the other between the base station and the receiver (Figure 2.9). Over the wireless hop, a specialized protocol tuned to the wireless environment may be used. Now, loss recovery for wireless losses can be done on the second connection without the data sender retransmitting data or reducing its window.

In [149], the Yavatkar and Bhagwat propose two protocols: one in which the wireless hop uses TCP and another in which the wireless hop uses a selective repeat protocol (SRP) on top of UDP. They study the impact of handoffs on performance and conclude that they obtain no significant advantage by using SRP instead of TCP over the wireless connection in their experiments. However, our experiments demonstrate benefits in using a wireless-optimized simple SACK protocol over the wireless connection to combat errors (Chapter 5).

Bakre and Badrinath's Indirect-TCP (I-TCP) [10] is a split-connection solution that uses standard TCP for its connection over the wireless link. Like other split-connection proposals, it attempts to separate loss recovery over the wireless link from that across the wireline network, thereby shielding the original TCP sender from the wireless link. However, as our experiments in Chapter 5 demonstrate, the choice of TCP over the wireless link does not really improve performance. Since TCP is not well-tuned for the error-prone link, the TCP sender of the wireless connection often times out, causing the original sender to stall. In addition, every packet incurs the overhead of going through TCP protocol processing twice at the base station (as compared to zero times for a non-split-connection approach), although extra copies are avoided by an efficient kernel implementation. A major disadvantage of split connections is that the end-to-end semantics of TCP ACKs is violated, since ACKs to segments can now reach the source even before the packets actually reach the mobile host. This makes the system vulnerable to base station failures or temporary changes in

routing to or from the MH. Also, since split-connection protocols maintain a significant amount of state at the base station per TCP connection, handoff procedures tend to be complicated and slow [8], often taking over an order of magnitude longer than alternate approaches [131].

Chapter 5 investigates the performance of two split-connection protocols and compares them to other approaches. We show that the performance improvements of these solutions compared to TCP are due to the special wireless-optimized techniques that can be deployed over the wireless connection, and *not* due to the split architecture. Furthermore, we show that wireless optimizations can be designed and implemented using a transport-aware reliable link protocol without incurring the architectural drawbacks of the split-connection approaches.

2.5.4 M-TCP

Inspired by the transport-aware ideas in the Snoop protocol (Chapter 4), Brown and Singh [22] propose a protocol called M-TCP that is designed to provide good performance when bit-error rates are low but disconnections due to mobility are frequent. They develop a split-connection approach that does not violate the end-to-end semantics of TCP ACKs, by taking advantage of the persist state in the TCP specification [20]. They throttle the sender completely when the mobile host is disconnected and resume activity when connectivity to the mobile host is reestablished. In the meantime, they ensure that no sender timeouts or retransmissions occur by sending ACKs that put the sender into persist mode. M-TCP does not readily address the problems due to varying and non-negligible error rates, focusing instead on disconnections due to mobility or link outages.

2.5.5 User Mobility

Caceres and Iftode [25] address the issue of TCP performance when communication resumes after a handoff. The TCP sender interprets the delay caused by a handoff process to be due to congestion. In many systems, handoffs complete quickly (relative to retransmission timer granularities), in between a few tens to a few hundred milliseconds, and long waits are required by the mobile host before timeouts occur at the sender and packets are retransmitted. They propose a fast retransmit approach to mitigate this problem by having the mobile host send a certain threshold number of duplicate ACKs to the sender and thereby trigger a fast retransmission. Thus, a connection can now resume without incurring a timeout.

Manzoni et al. [90] use the above results to motivate changes to TCP sender implementations to explicitly recognize ongoing mobility using messages sent from the base station. During this time, they extend the slow start phase to reduce the adverse effects of packet loss due to host mobility. We believe that the previous approach [25] solves the key problems in a simpler manner because it requires no changes to TCP fixed host implementations.

Maltz and Bhagwat propose the use of transport layer mobility and TCP splicing [89] to overcome the problems associated with mobility on transport performance. The innovative aspect of their approach is the use of a split connection approach that does not violate the end-to-end semantics of TCP ACKs. The connection is "split" in the kernel of the base station by changing certain contents of the TCP/IP header before forwarding the segment to the mobile host. This prevents any disruption during a handoff. A transport-aware link protocol like the Snoop protocol can coexist with this scheme, providing good performance when wireless corruption occurs.

We observe that the above solutions mainly address the problems due to mobility and not the error characteristics of wireless links. In Chapter 4, we show how the Berkeley Snoop Protocol and a multicast-based low-latency handoff scheme combine to provide good performance in the face of both wireless bit-errors and user mobility.

2.6 Asymmetric Networks

A wide variety of asymmetric technologies are becoming an increasingly popular alternative for Internet access. Bandwidth asymmetric networks, where the bandwidth in one direction (the *forward* direction) is several times that in the reverse, are becoming commonplace. The motivation for such networks is primarily economic—it is much cheaper and easier to provide high bandwidth in one direction than in both—and it is possible to take advantage of existing telephone or cable infrastructure to do this. In addition, today's dominant Internet application, Web access, displays asymmetric characteristics, with more data typically originating from a server than from a client.

Typical bandwidth ratios between the forward and reverse directions in these networks vary from between 10 and 1000. Prominent examples of bandwidth asymmetric networks include cable modem networks, Direct Broadcast Satellite (DBS) networks and wireless cable networks. In wired cable modem networks, the forward channel uses conventional cable lines that are also used

for cable television distribution. The reverse channel is either out-of-band, using a dialup telephone line, or is in-band using the same cable lines but at significantly lower bandwidth than in the forward direction. DBS networks such as Hughes' DBS system [39] provide large forward bandwidths over a large area. The reverse channel is usually via a telephone dialup line or a wireless link. Wireless cable networks are similar to their wired counterparts in terms of packet framing and the media-access protocols, but use an entirely wireless forward path. This is the network we use in our research; more technical details of this network are presented in Chapter 3.

There have been some recent efforts to analyze and solve the problems raised by bandwidth asymmetry on TCP performance. These problems arise because of slow ACK feedback, loss of ACKs, or the presence of bi-directional traffic in the network.

2.6.1 Asymmetric Satellite Networks

Durst et al. [41] investigate the performance of TCP over asymmetric, error-prone satellite networks. They propose the following extensions to TCP for this environment, as part of a the Space Communications Protocol Standards-Transport Protocol (SCPS-TP): signaling different types of loss to the sender (congestion, corruption and link outage) based on explicit default assumptions about link loss characteristics, delaying ACKs based on receiver estimates of round-trip time from the timestamp option [61], header compression that tolerates some packet losses (unlike the scheme for serial links in RFC 1144) and the use of selective negative ACKs (SNACK) for better loss recovery.

Inspired in part by the Service Specific Connection-Oriented Protocol (SSCOP), an ATM adaptation layer protocol, Henderson and Katz discuss the design and implementation of the Satellite Transport Protocol (STP) for better transport performance over loss-prone satellite networks [53]. An STP sender does not rely on continuous ACK feedback from the receiver, which helps it perform better than TCP Reno over asymmetric paths. Periodically, the receiver sends a STAT (status) message in response to a sender POLL, which informs the latter of missing packets or blocks of data, which the sender retransmits. The frequency of STATUS messages is a tunable parameters; in addition, the receiver can send an unsolicited status (USTAT) message informing the sender of its state.

2.6.2 Bandwidth Asymmetry Analysis

Lakshman and Madhow [78] study the problems caused by bandwidth asymmetry using analysis and simulation. They define the *normalized asymmetry ratio* as the ratio of the bandwidth ratio to the average packet size ratio, and show that performance degrades if this quantity is greater than one. They also propose the use of the drop-from-front strategy on ACKs in the constrained reverse channel buffer as a way to alleviate the problems that arise.

2.6.3 Bi-directional Traffic

Kalampoukas et al. [67] study the problems that arise when bi-directional traffic is present in a bandwidth-asymmetric network, using a combination of analysis and simulation. They show that *ACK compression,* where ACKs arrive in quick succession disrupting smooth ACK clocking, sometimes occurs in these environments reducing throughput. They investigate three approaches to counter this: (i) prioritizing ACKs over data on the reverse channel, which unfortunately results in starvation, (ii) using a connection-level backpressure mechanism to limit the maximum amount of data buffered in the reverse channel router, which unfortunately results in unfair bandwidth utilization and (iii) using connection-level bandwidth allocation, which provides flow isolation.

2.6.4 ACK Congestion Control

In [14], we propose the use of ACK Congestion Control (ACC) as a way to adaptively extend TCP's delayed ACK approach at the receiver, using in-band congestion feedback from the constrained reverse router and TCP sender. This approach was further investigated by Padmanabhan

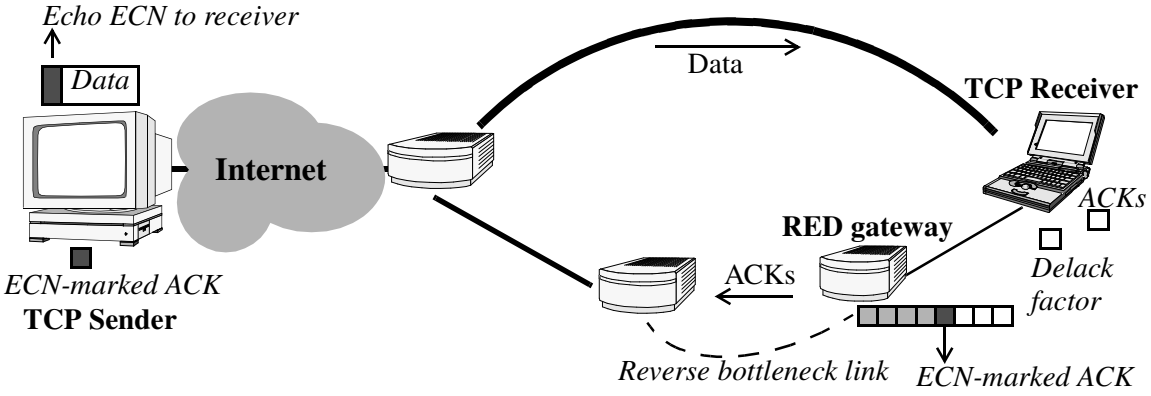

Figure 2.10 ACK congestion control using a RED gateway at the reverse router with ECN support at the sender and receiver to dynamically vary *delack*.

especially in the context of bi-directional traffic [107]. When used in conjunction with a scheduling mechanism that prioritizes ACKs over data on the reverse channel, ACC achieves close to optimal throughput for transfers in both directions, sharing the reverse bottleneck resources fairly.

The idea in ACC is to extend TCP congestion control to TCP ACKs in addition to data, since ACKs consume scarce resources at the low-bandwidth bottleneck link in the reverse direction. In many systems (e.g., BSD/OS), ACKs occupy slots in the reverse channel buffer, whose capacity is often limited to a certain number of *packets* (rather than bytes).

ACC relies on the Random Early Detection (RED) algorithm [47] coupled with ECN [123] at the gateway of the reverse link to aid congestion control (Section 2.3.5). In ACC, *both* data packets and TCP ACKs are candidates for being marked. The TCP receiver maintains a dynamically varying delayed ACK factor, *delack*, and sends an ACK every *delack* data packets. When it receives a packet with the ECN bit set (echoed by the data sender to it), it increases *delack* multiplicatively, thereby multiplicatively reducing the frequency of ACKs. Then, in each subsequent round-trip time (determined using the TCP timestamp option [61]) during which it does not receive an ECN-marked packet, it linearly decreases *delack*, thereby additively increasing the frequency of ACKs. Thus, the receiver mimics the standard congestion control behavior of TCP senders in the manner in which it sends ACKs, extending TCP congestion control to ACKs as well (Figure 2.10).

There are bounds on the delayed-ACK factor *delack*. Obviously, the minimum value of *delack* is 1, since at most one ACK is sent per data packet. The maximum value of *delack* is determined by the sender's window size, which is conveyed to the receiver in a new TCP option. The receiver should send at least one ACK (preferably more) for each window of data from the sender. Otherwise, it could cause the sender to stall until the receiver's delayed-ACK timer (usually set at 200 ms, but no more than 500 ms in the TCP specification) forces an ACK to be sent.

The advantage of ACC is that it is an end-to-end scheme where most of the machinery is at the end points of the TCP connection. The only assumption made of the network is that the reverse bottleneck deploy a RED gateway in marking mode. With the current rapidly increasing interest in deploying RED gateways in the Internet, this does not seem to be a problematic assumption. However, there is a time constant on the order of a round-trip time before the state of congestion is inferred and dealt with by the TCP receiver. In addition, the protocol requires that the appropriate

machinery be deployed at both the sender and receiver, as well as the reverse bottleneck gateway, which makes incremental deployment challenging. In Chapter 6, we present some performance results of ACC and compare it with the other schemes described in that chapter for combating the adverse effects of asymmetry in different networks.

2.6.5 TCP over Hybrid Fiber Coaxial Broadband Access Networks

Cohen and Ramanathan [35] study TCP performance over a hybrid coaxial cable distribution system using simulations of that network. They conclude that it is important for receivers to tune their socket buffer sizes; their proposed modifications include an initial window of two segments as opposed to one, smaller segment sizes to overcome problems with delayed TCP ACKs and ACK losses, a finer granularity for TCP timers, and fast retransmission triggered by a single duplicate ACK for better loss recovery. In general, while some of these techniques improve performance in their network, they are not universally applicable and lead to worse performance in other networks. For example, retransmitting a segment upon the arrival of the first duplicate ACK is not correct in general because of the significant amount of packet reordering in the Internet. Our philosophy, in contrast, has been to improve transport performance heterogeneous wireless networks without compromising performance in other situations.

The authors of this paper also claim that the Snoop protocol [16] requires significant overhead in the network and is ill-suited to large deployments. In Chapter 4, we discuss the scaling behavior of the Snoop protocol via simulation and analysis and show that this concern is unfounded.

2.7 Packet Radio Networks

A packet radio network is a network of nodes that communicate with each other over wireless, routing packets through a set of radio nodes. These network nodes may be either static or mobile. We are interested in topologies where end hosts with wireless interfaces use the radio network, traverse multiple wireless hops, and gain access to the wired Internet infrastructure via a gateway to the packet radio network. In this section, we briefly discuss the problems to transport performance that arise in these networks and describe related work in the area.

The first issue is *reliability*, since wireless links are error-prone (Section 2.5): a reliable link-layer protocol is therefore essential. The second issue is *channel access*: a contention-resolution proto-

col is essential to prevent the network from collapsing under even moderate load. The third issue is *packet reordering*: many packet radio networks explicitly route packets via different parallel routes, leading to potential reordering in the network that the end-points must handle.

The fundamental problems to TCP performance in these networks arise because of interactions between data and ACKs, which lead to imperfect ACK feedback. In this respect, these problems are similar to the ones observed in bandwidth asymmetric networks. In Chapter 6, we discuss this in detail and present solutions to them; to the best of our knowledge, there has been little progress to date on these problems in packet radio networks, despite their importance.

Metricom Inc. [98] has deployed a network using Ricochet modems and poletop packet radios in several areas in the United States. Our work is based mainly on their network and technology. Cheshire and Baker [29] present results of measurements of the Ricochet modems, focusing primarily on end-host overhead and latencies through the modems. Lai et al. [77] log traces of real traffic over an eight-day period over Ricochet modems. They show that mean round-trip latencies in their traces are over 700 ms and that higher delays occur due to packet loss and reordering.

2.8 Summary

This chapter started with a description of the best-effort Internet service model and motivated the need for reliable transport protocols discussing the salient features of several protocols of historic interest. We then described the salient features of TCP, the protocol that makes today's Internet reliable and discussed several recently proposed end-to-end enhancements to it. In Section 2.4, we surveyed some key concepts in wireless networks. In Section 2.5, we discussed and critiqued conventional approaches to tackling the problems caused by wireless bit-errors and user mobility. In Section 2.6 and Section 2.7, we described background material about bandwidth-asymmetric and packet radio networks respectively.

We observe that current approaches towards handling wireless bit-errors, asymmetric effects and small transmission windows do not comprehensively solve observed problems. The material described in this chapter sets the stage for our work. The next chapter outlines the methodology used in our work and describes the details of the wireless testbed and experimental networks that serve as our research platform.

"You know my methods, Watson."

— Sherlock Holmes in 'The Crooked Man,' by Sir Arthur Conan Doyle

Chapter 3

Experimental Methodology and Wireless Testbed

In this chapter, we describe the methodology and tools used to perform the research reported in this thesis. We start by describing our research methodology and the network simulator we used, followed by the details of the Bay Area Research Wireless Access Network (BARWAN) experimental testbed and the different networks that form the testbed. We conclude this chapter with a description of our measurement techniques and performance metrics.

Our goal is to improve the performance of TCP in heterogeneous networks comprised of wireless links. To achieve this goal, we use a combination of analysis, simulation, design, implementation and performance measurement. Figure 3.1 is a schematic illustration of our general research methodology, which involves three phases. In the first phase, we *analyze* the problems that affect transport protocol performance by measuring existing implementations in deployed wireless networks. This analysis is done using different TCP-related workloads, whose control parameters include the transfer size (bulk vs. short), bit-error and congestion rates, scale (e.g., number of simultaneous connections), etc. Using the tools described in Section 3.3, we collect *packet-level traces* of these transfers and analyze them in detail. This analysis yields a better understanding of the reasons for degraded performance, and also gives us data on realistic values for the different parameters in the network elements, such as bandwidths, delays, error patterns, etc.

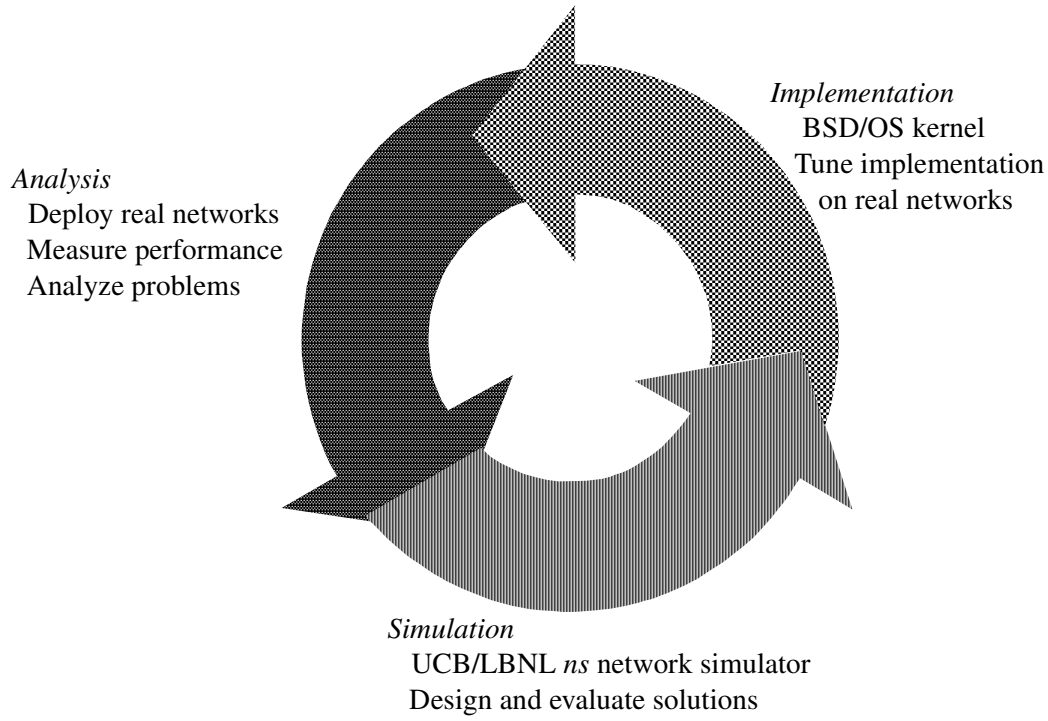

Figure 3.1 Three phase research methodology involving analysis, simulation and implementation.

The gathered data and intuition on the nature of the problems serve as the starting point for the second phase, *simulation*. Simulation is a powerful approach that enables rapid prototyping of various ideas and solutions, and allows us to explore the design space of parameters and algorithms more thoroughly than in a direct implementation. It also helps us "time travel" and explore possible technology trends that help us predict what performance using different protocols might be in the future. The main advantage of simulation is in having a controlled environment in which we can analyze problems and design solutions, explore a wide variety of alternatives, and compare them on an even basis.

Once we obtain a good understanding of the problem from measurement and simulation, we move to the third phase, *implementation*. Here, we implement the promising solutions from the simulation phase in our experimental testbed. Most of our implementation involves changes to the TCP stack at the end hosts, or to the link-layer software at intermediate nodes in the network. Currently, all our experimental machines are PC's, running BSD/OS 3.0 from BSDI [23]. The networking stack in BSD/OS 3.0 is similar to that in the 4.4 BSD-Lite distribution [97], and is similar to other

widely used systems like FreeBSD and NetBSD. This TCP stack is a mature implementation that has been widely studied and researched in the networking community [e.g., 148], making it an ideal choice for our work.

After completing an initial implementation, we proceed to the *performance evaluation* phase, which is shown in Figure 3.1 leading back to the initial analysis phase. Here, we measure the performance of our new algorithms and protocols under different network conditions and workloads, and gather packet-level traces to understand the reasons for observed performance. When we cannot expect to completely explore how the implementation performs under a wide variety of conditions (which simulation is ideally suited for), we use *emulation* to achieve this. For example, in Chapters 4 and 5 we use emulation to evaluate the real implementations under a variety of emulated channel error conditions. Finally, we compare the results of these measurements to the results obtained from simulation. This helps us tune the implementation to perform well, and also helps identify shortcomings in the simulations that we correct to accurately reflect reality.

The rest of this chapter discusses our simulation environment, the experimental networks in our wireless testbed, and the measurement techniques and performance metrics we used.

3.1 Simulation Environment

Simulation is an invaluable tool in networking research and our work is no exception to this general rule. The advantages of simulation are the ease with which a variety of different policies and algorithms can be tested and evaluated, before implementing the most promising ones in real environments. It also helps us vary network parameters such as bandwidths, latencies, error rates, offered workload and system scale to better understand resulting performance under a wider variety of conditions than is possible in real networks with real implementations.

In the early stages of our research, we used Keshav's REAL network simulator, version 4 [74]. We added several modules to REAL to simulate a typical wireless network—wireless base stations, mobile hosts, a bit-error generator—and implemented in it an early version of the Snoop protocol (Chapter 4). The key benefit of using REAL were its realistic TCP modules, implemented based on the BSD code. It greatly facilitated our understanding of the dynamics of TCP and its problems over wireless links, guiding our design of the protocol features to improve its performance.

However, REAL 4.0 was somewhat inconvenient and inflexible to use in some crucial ways. Because it was written entirely in C with parts in assembler, it was hard to extend existing modules or reuse code easily. Furthermore, simulation scenarios had to be specified in a custom language that was limited in power and expressiveness compared to a general purpose programming language[1]. This made it cumbersome to perform large-scale simulations. Due to these reasons, we moved our simulation efforts to another simulator, the UCB/LBNL network simulator *ns* (Version 2) [103], now widely used as part of the VINT project [142].

ns is an event-driven simulator originally derived from REAL. Written mainly in C++ [138] and MIT's Object Tcl (OTcl) [146], its object-oriented software architecture makes it easy to add new modules and extend existing ones using object inheritance. Simulations are programmed in OTcl, an object-oriented extension to Tcl [106], a popular scripting language. Moreover, there is a tight coupling between objects in OTcl and C++, which makes it easy to move functionality between the two programming realms. The tight coupling between objects in C++ and OTcl is achieved by the use of *split objects* [96]. These objects reside simultaneously in both language realms and permit access to instance variables from either language. These variables are *bound* using a special `bind()` procedure, after which any modification to their state initiated from either language is visible in the other.

Typically, the choice of language involves a trade-off between performance and ease of use: OTcl makes it easy to rapidly prototype and experiment with new algorithms and parameters, while C++ is apt for large simulations that require high performance. As a rule of thumb, functionality that requires per-packet processing in *ns* is best implemented in C++, while the more infrequent and experimental code fragments are best implemented in OTcl.

We briefly describe two important enhancements we made to *ns*[2]: (i) Local Area Network (LAN) support and (ii) channel error models.

1. REAL 5.0 corrects several of these shortcomings.
2. In collaboration with Giao Nguyen (UC Berkeley), whom we gratefully thank for his contributions.

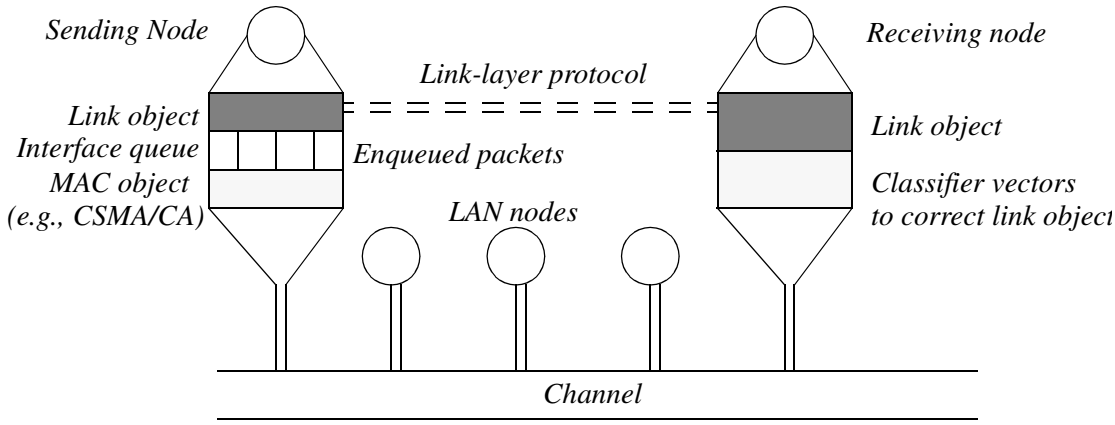

Figure 3.2 Schematic representation of LAN objects in *ns*.

3.1.1 LAN Objects

We added a set of modules to model LANs, especially wireless LANs. Prior to our work, the only link abstraction in *ns* was that of a point-to-point link. We extended this to a more general LAN abstraction, adding support for link-layer protocol modules, interface queues, media-access modules and channel modules. The details of the implemented LAN architecture can be found in [103]. In our work, we experiment with link layer protocols, interface queues with standard drop-tail and RED [47] algorithms, media access protocols like CSMA/CA and packet radio MAC protocols, and shared channels with statistical error models.

A LAN is a list of nodes each of which has a link object that implements a link-layer protocol. At the sender, each packet passes through this link object and is received by an interface queue that enqueues all packets destined for nodes on the same LAN, regardless of destination. The node's media access (MAC) object for the LAN dequeues one packet at a time from the interface queue and passes it to the channel shared by all nodes on the LAN as mandated by the implemented MAC protocol. The channel uses the packet's MAC header to vector the packet to the appropriate agent (called a *classifier*) on the receiving node. The receiving classifier then vectors to the appropriate link object based on where the packet came from (i.e., its source on the LAN). Thus, link objects are pairwise between nodes on the LAN, a single output interface queue and MAC object is shared by all link objects at each node on the same LAN, and all entities on the LAN share the same channel. This model accurately reflects reality and is illustrated in Figure 3.2.

Although the above description is for a general LAN with many nodes, it subsumes the important special case of point-to-point communicating entities with link, MAC, and channel objects. For example, a point-to-point wireless link in a multi-hop wireless network is modeled this way, even though that link is not strictly a "LAN."

3.1.2 Error Models

To understand and investigate the performance of different protocols under controlled error conditions, we simulate statistical error models in *ns*. The function of an error model is to generate errors based on statistical distributions, with errored packets being corrupted and dropped on the link. There are several design issues that need to be resolved before we can describe the statistical details of these error models.

Object structure: While in reality wireless bit errors occur due to the characteristics of the underlying wireless channel, it is not appropriate to generate them as part of the channel module described above. For if we did this, we would not be able to precisely control the *pairwise* distribution and rate of wireless errors between nodes, without maintaining a significant amount of link-layer state as part of the channel's error module. We therefore abstract the error module into an entity similar to the link-layer module, whose internal state and statistics are per communicating pair of nodes. This makes the error module self-contained and easy to control. In addition, there is a choice of where to place this module—at the sender or at the receiver on the wireless LAN. By placing it at the receiver, we automatically ensure that corrupted packets traverse the channel and consume bandwidth, accurately modeling reality. The other reason to place the error module at the receiver is to enable the use of a time-based error model, for which it is important to know how long the packet transmission took. This information is unavailable at the sender prior to packet transmission. Thus, we place the error module at the receiver, between its MAC and link objects; if the error module determines that the packet be corrupted, an error bit is set in the its header before it is passed on to the link object.

We implement three types of error models: packet-based, byte-based (or equivalently, bit-based), and time-based. These define the units over which the statistical error generators in the error modules operate. Packet-based error models introduce errors on a per-packet basis, independent of the time since the last error or the size of the packet. They are ideal to simulate congestion losses

where packets are dropped upon queue overflow. To simulate wireless channel errors, we typically use either the byte-based or time-based models since they model reality more accurately. In particular, time-based models capture effects like channel fades that last for time periods independent of the size or number of packets being transmitted.

Analytic state models: The simplest error models are *one-state models* that are characterized by a single *rate* parameter and a statistical distribution that determines the error rate. For example, a one-state exponential error model generates errors at a certain rate based on an exponential distribution in the appropriate domain (packet, byte, or time). More complex *multi-state* error models are implemented by *composing* one-state models and gluing them together in OTcl. Such models have been extensively studied in the literature and are characterized by a number of states organized as a Markov chain. Each state is a one-state error model governed by a certain rate and distribution, with probabilistic transitions occurring between states according to a transition matrix of probabilities. Thus, complex multi-state modules are automatically composed from simple one-state ones.

Empirical distribution-based models: While analytic state models are sound in theory, it is hard to determine in practice what the parameters of these models should be. The process of fitting experimental channel error data in this framework is not straightforward. While using simple one-state models enables us to quantify the relative performance as a function of a base error rate, it does not give us any insight into what "typical" performance might be like in a real wireless environment under realistic error conditions. This motivates us to develop an error model based on actual traces from a wireless environment and use that model to evaluate our protocols. This section describes how we do this.

Our methodology for constructing a realistic wireless error model is motivated by the approach presented in [102], where the authors analyze in-building traces of wireless traffic and construct empirical models for them. Rather than use the traces they collected, we decided to obtain fresh ones. This is because we could validate the performance observed in simulation using these error traces by comparing these results to actual implementation performance.

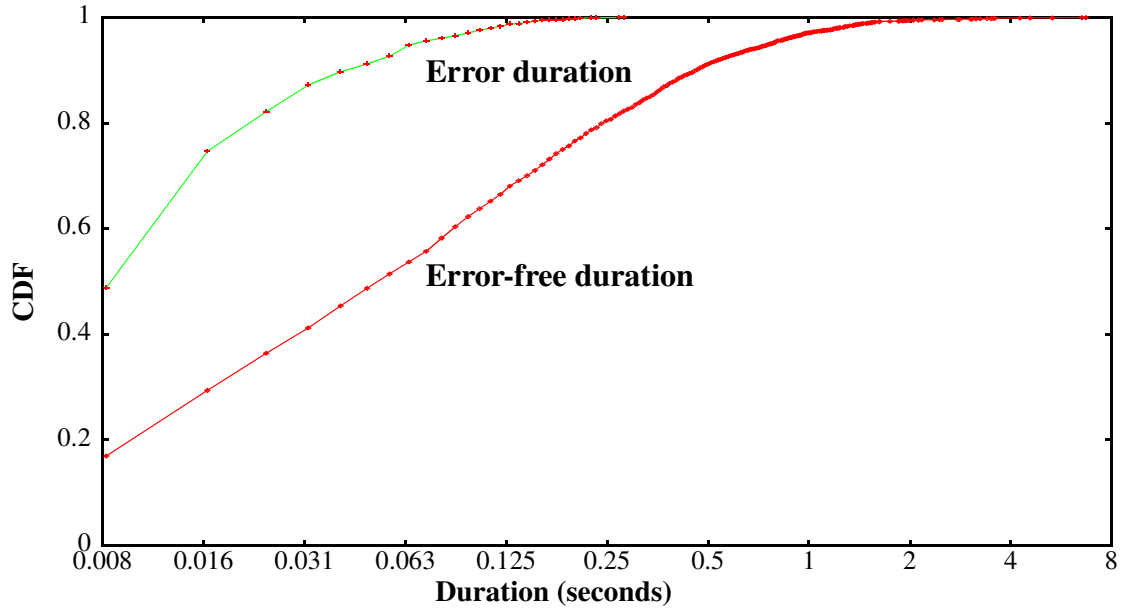

Figure 3.3 Cumulative Distribution Functions (CDF) of empirically measured error and error-free durations. The CDF was computed over of 100,000 packets sent over 800 seconds in the Reinas outdoor wireless network.

To develop an empirical error model of the channel, we proceed as follows. We transmit a constant rate UDP stream at rate R bps, with a fixed packet size, S bits. We ensure that the transmitted rate is smaller than the sustainable link rate, and that the size of the packets is large enough that the number of packets per second does not overwhelm the receiving processor, which needs to take an interrupt for every reception. Then, we send a large number of packets, N, with an application-level sequence number on each packet. Thus, the receiver can now determine which packets have arrived correctly, and which have not. UDP checksum processing at the receiver is enabled, so any packet whose lower-layer headers are correct but whose payload is corrupted will be discarded. Of course, no packets are retransmitted.

Using this data, the receiver constructs histograms of the lengths of error and error-free sequences. We obtain the distribution of the number of successive packets that were successfully received, as well as the number of missing ones once there was a loss. At this point, this histogram is in terms of packets, rather than time. However, the underlying characteristics of the channel's error properties are temporal, related to time-dependent factors like the average fade duration and duration of inter-symbol interference due to multi-path effects (Section 2.5.1). Hence, we need to convert our

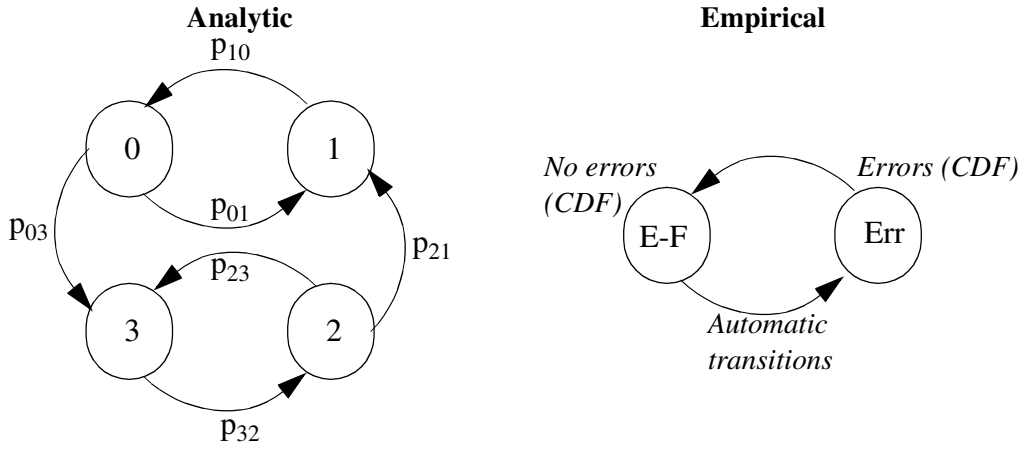

Figure 3.4 Conventional multi-state analytic error models (left) and empirical distribution-based models (right). In analytic models, the different states correspond to different base error rates, and the arc probabilities (p_{ij}) govern state transitions. Our empirical models are easy to fit to observed experimental data and contain two states—error-free and error, which have CDF's associated with them that determine state occupancy. State transitions are automatic when this time elapses.

packet-based distribution into a time-based one, which we achieve by multiplying the number of packets in any histogram bin by the packet transmission time, *S/R*.

We used the experimental network deployed in the Reinas Environmental Monitoring project at the University of California, Santa Cruz to obtain our data. This network, described in Section 3.2.1, has a number of point-to-point long-haul WaveLAN links as well as lower-bandwidth RF links.

We performed eight separate experiments using this approach, with *N=100,000* packets in each experiment, packet size *S = 8000* bits and *R = 1* Mbps. The resulting error and error-free temporal distributions are shown in Figure 3.3. In the experiments that follow, we picked one of these data sets. In this data set, 10,591 packets were lost due to wireless errors, a gross loss rate of 10.6%. While this might seem high, it is in fact quite representative of the quality of the outdoor wireless links in the Reinas network. The typical loss rates in the network vary from 1% to over 30%, depending on external factors and environmental conditions.

The cumulative distribution functions (CDF) of error and error-free durations are used to generate bit errors in the following manner. There are two linked states, corresponding to the error-free and error durations, that the error model transitions between. It starts in the error-free state where no errors occur, and stays there for an amount of time determined by a lookup into the corresponding

CDF. After this time has elapsed, there is an *automatic* transition to the error state, where all incoming packets are corrupted. The amount of time spent in this state is, as before, determined by the corresponding CDF. Thus, this empirical error model shown in Figure 3.4 is a two-state Markov chain, but is quite different from the conventional analytic models. Here, the states themselves correspond to time, with automatic state transitions. The analytic models are richer in that each state has a different base error rate, with explicit transition probabilities between states. However, it is hard to reconcile experimental data to meaningful parameters that can be used in these models.

3.2 BARWAN: Bay Area Research Wireless Access Network

Our work was done on the wireless testbed of the BARWAN project at Berkeley. This section discusses the project vision and the details of the wireless testbed.

Despite the publicity surrounding wireless data, networks and services based on wireless data have thus far been more of a promise than a reality. The vision of the BARWAN project is that success for wireless data depends on the development of a digital communications architecture that integrates and inter-operates across regional-area, wide-area, metropolitan-area, campus-area, in-building, and in-room wireless networks [72]. Unfortunately, network planners continue to think in terms of homogeneous wireless communications systems and technologies, without appreciating the challenges posed by heterogeneity, especially as far as performance is concerned. Future mobile information systems are likely to be built upon heterogeneous *wireless overlay networks*, extending the traditional wired Internet to mobile hosts over wireless link technologies. A schematic representation of an overlay network is shown in Figure 1.3, which shows a tiered hierarchy of wireless technologies with different characteristics and capabilities. The BARWAN project solves problems associated with routing and handoffs [129, 133], reliable transport (this thesis), Web transport [107], proxy-based application support [48, 4] and service location [54] to achieve this vision.

In the BARWAN testbed, we have deployed a number of experimental wireless networks. Below, we present an overview of three of these networks that we investigate in this thesis: Lucent Technologies' WaveLAN, Metricom Inc.'s Ricochet, and Hybrid Networks Inc.'s wireless cable network.

3.2.1 Lucent's WaveLAN

Lucent's WaveLAN is a wireless LAN product that operates in the unlicensed ISM band. There are two products, one in the 915 MHz frequency range, and the other in the 2.4 GHz range; we use the 915 MHz product in our experiments. The underlying technology is a direct sequence spread spectrum modulation technique (Chapter 2) with a 11-chip sequence spread across a 26 MHz band. The maximum raw signaling bandwidth of this network is 2 Mbps, shared amongst all hosts on the wireless LAN. The maximum transmission power of WaveLAN devices is 100 mW, governed by the FCC etiquette rules [43].

Hosts on the same LAN communicate with each other directly without base station intervention, using the contention-based CSMA/CA media-access protocol [139]. In our testbed, base stations are Pentium PC's with an ISA WaveLAN card, while laptop mobile hosts use the PCMCIA version of the card.

Under ideal conditions when the communicating nodes and antennae are close to each other, TCP throughput on the WaveLAN is about 1.5 Mbps. This is the maximum possible throughput, in the absence of losses or long delays. Typical minimum round-trip latencies are between 3 and 6 ms. However, in actual settings, measured TCP throughput typically varies between 50 Kbps to 1.5 Mbps, and is generally closer to the lower end of the spectrum. This is primarily because of wireless errors being mistaken for congestion by the TCP sender (Section 2.5), causing it to reduce its window and incur long timeouts that keep the link under-utilized. Chapters 4 and 5 investigate this problem in detail and present solutions that significantly improve TCP throughput.

In addition to the WaveLAN deployment in the BARWAN testbed, we also had access to a WaveLAN-based wireless network deployed at the University of California, Santa Cruz, as part of the Reinas environmental monitoring project [85]. Here, the communicating nodes are separated by up to a few miles and communicate on a point-to-point basis across the wide-area using high-gain Yagi antennae.

3.2.2 Metricom's Ricochet

Metricom, Inc. has deployed the Ricochet packet radio network in a few places around the United States, including the San Francisco Bay area, Seattle, Washington DC, and several major airports.

Figure 3.5 Ricochet's SX packet radio modem.
Picture courtesy: Metricom Inc. (http://www.metricom.com).

Subscribers either purchase or lease radio modems that attach to the serial port of a computer and typically use PPP [132] to communicate with the rest of the Internet. There are a variety of portable modems available, such as the one shown in Figure 3.5. The Ricochet network operates in the unlicensed 915 MHz ISM band and uses frequency-hopping spread spectrum technology to achieve frequency diversity during communication.

The radio units in the network are *half-duplex*, implying that they cannot simultaneously transmit and receive data. The media-access protocol used in this network is a variant of the 802.11 RTS/CTS-based scheme, where a handshake occurs before every communication to synchronize peers.

3.2.3 Hybrid Networks' Wireless Cable System

Hybrid's wireless cable network is an example of an asymmetric bandwidth network, with differing bandwidths in the forward and reverse directions. The available bandwidth in the forward direction from the head-end to the client is a relatively plentiful 10 Mbps, shared amongst clients. However, the reverse direction has much lower bandwidth, typically using a dialup phone line or packet radio channel. In many of our experiments, we used the latter option, resulting in a fully wireless access topology. Because the reverse channel is constrained, TCP ACKs are dropped or delayed, under-utilizing the capacity of the forward channel.

The cable modem RF transmitter and receiver (shown in Figure 3.6) provide the physical layer for broadband communication over coaxial plant, hybrid fiber/coax (HFC) microwave (MMDS) or

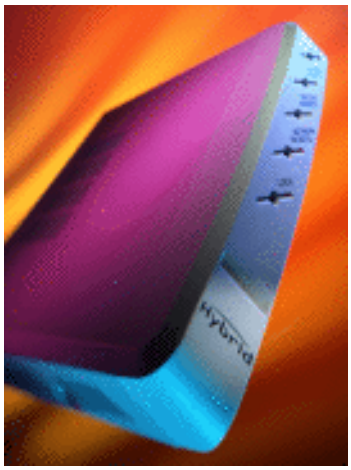

Figure 3.6 Hybrid Inc.'s cable modem for asymmetric Internet access.
Picture courtesy: Hybrid Inc. (http://www.hybrid.com).

other wireless systems. In the forward direction, the subscriber shares 10 Mbps bandwidth, or equivalently, one of three 2 MHz sub-channels within the standard 6 MHz TV channel. An modulation technique based on 64-Quadrature Amplitude Modulation [56] allows the maximum bandwidth to be as high as 10 Mbps. In our testbed, the head-end located on campus beamed the signal to a transmitter located on a hill, which in turn broadcast the signal to a receiving antenna, from where local redistribution occurred.

3.3 Measurement Techniques and Performance Metrics

In this section, we discuss the tools and techniques used to measure and evaluate TCP performance and our improvements in these wireless systems.

We wrote `netperf` [130], a general network performance utility to measure performance for TCP workloads. This program has numerous options and workload models including bulk transfers, Web workloads, and transactions. We also use `netperf` to transmit unreliable UDP data at a constant rate to obtain data for channel error modeling. To evaluate performance using a Web-like workload, we use an empirical model based on data collected by Mah [88] of client-side HTTP packet traces from a subnetwork in U.C. Berkeley's Computer Science Division. This empirical model is characterized by CDFs of HTTP request lengths, server response lengths, number of concurrent streams, client inter-arrival times, and user idle times between successive downloads.

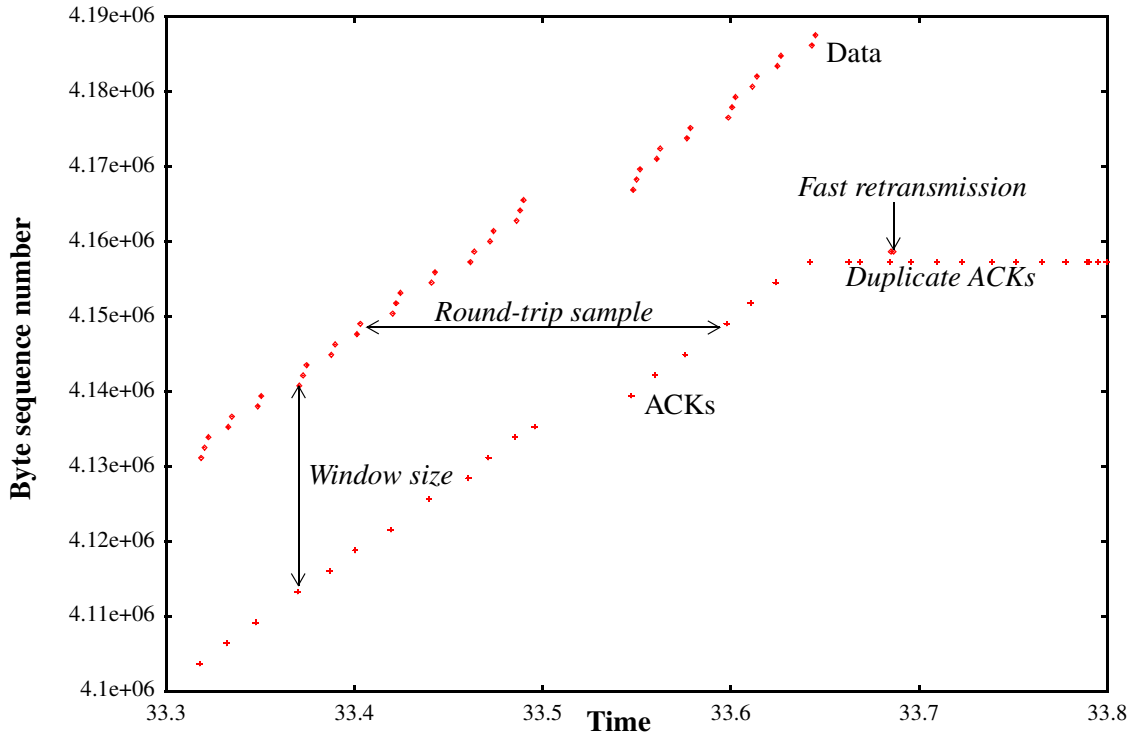

Figure 3.7 Sequence number plot of a portion of a TCP transfer.

To perform a microscopic analysis and understand the reasons for measured performance, we collect detailed packet traces of transfers using the `tcpdump` network monitoring tool [94]. It is often helpful to view `tcpdump` data as a sequence number plot. A sequence number plot is a graph that plots TCP segment and ACK sequence numbers as a function of time (e.g., Figure 3.7). These plots are useful in visualizing the progress of a TCP transfer and identifying interesting events during the transfer, including timeouts, fast retransmissions, burst transmissions, etc. In addition, in many of our experiments we instrumented the TCP sender's kernel to log interesting TCP events for accurate offline analysis. Logged kernel variables include the congestion window, round-trip samples and running average estimates, retransmissions, slow start threshold, etc.

The metrics we use to evaluate performance include *throughput, goodput, utilization,* and *fairness index*. For any network application, we define throughput as the ratio of the total number of received bytes to the total transfer time. It is measured in Kbps (Kilobits per second) or Mbps (Megabits per second). Goodput may be thought of as the effective throughput of the transport layer: we define it as the ratio of the total number of *useful* (to the application) bytes to the total

number of bytes transmitted by the sender or over a link. It is always less than or equal to 1, and is usually reported as a percentage. The closer it is to 1, the more efficient the system; ideally, it should be equal to *1 - p*, where *p* is the base loss rate for the connection.

When there are multiple connections simultaneously contending for network resources, there are two relevant metrics to quantify: *utilization* and *fairness*. Utilization (reported as a percentage) measures how often the contended resource is idle, while fairness measures how evenly the connections share the scarce resource amongst themselves. We use Jain's *fairness index* [62] to measure fairness in throughputs. If there are n simultaneous connections and x_i their throughputs, then the fairness index, f, is defined as:

$$f = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2}$$

The fairness index always lies between 0 and 1 for non-negative throughputs, and is equal to (k/n) if k of the n connections receive equal throughput and the remaining $(n - k)$ none. Thus, f cannot be less than $1/n$ in a network with n connections. We use the fairness index to understand and analyze the scaling properties of the network when multiple connections are simultaneously active. In some cases, we analyze the memory consumption of our proposals to evaluate their scaling behavior when the number of connections increases.

3.4 Summary

This chapter described our research methodology, which includes analysis, simulation, design, implementation, and performance evaluation. We described *ns*, the event-driven simulator we used and the details of how we derived empirical distribution-based error models. We also described our wireless testbed consisting of Lucent's WaveLAN, Metricom's Ricochet and Hybrid Inc.'s wireless cable networks. Finally, we defined the metrics used to evaluate performance in the rest of this thesis.

There are more things in heaven and earth, Horatio,
Than are dreamt of in your philosophy.

— Hamlet [I. v. 166], William Shakespeare

Chapter 4

The Berkeley Snoop Protocol

This chapter describes the design and implementation of the *Berkeley Snoop protocol*, which significantly improves the end-to-end performance of TCP in error-prone cellular wireless networks. It achieves this by exploiting the idea of *cross-layer protocol optimizations* for enhanced end-to-end performance.

4.1 Introduction

In Chapter 2, we described the major problems that arise when a TCP connection traverses wireless links, communicating with hosts that are mobile. An important characteristic of wireless links is the occurrence of bit-errors due to physical and environmental factors. This is often coupled with periods of intermittent connectivity and packet losses that could occur during cellular handoffs, as described in Section 2.5. Protocols and applications have to adapt to these channel errors to achieve good performance.

In this chapter, we will be concerned primarily with cellular network topologies (Section 2.4.1). In particular, we will focus on topologies similar to the one shown in Figure 4.1, that are composed of base stations and single-hop wireless links. Here, an end-host is directly connected via a wireless

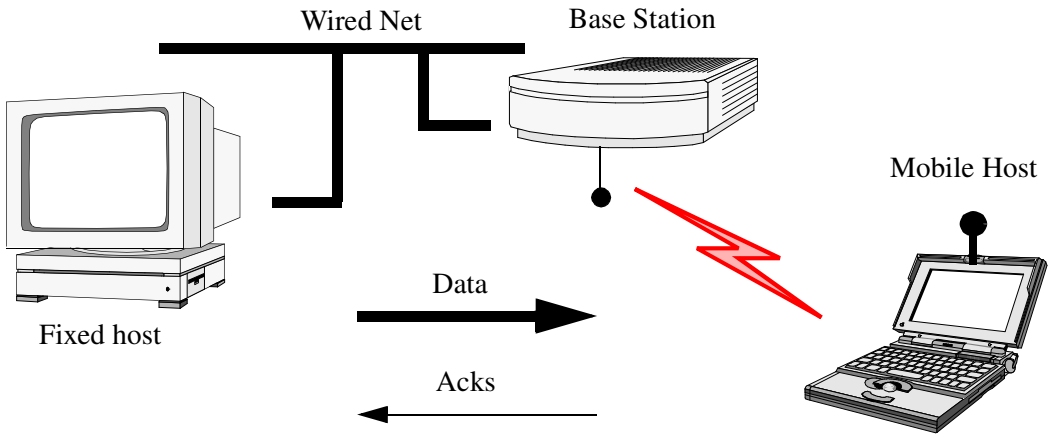

Figure 4.1 Wireless topology for the Snoop protocol.

link to a *base station (BS)*, which is then connected over a number of wired hops to the rest of the Internet. We call the end-host connected via the wireless link, the *mobile host (MH)*, and any host on the wired Internet communicating with it, the *fixed host (FH)*. Our goal is to achieve good end-to-end performance for reliable data transfers in both directions.

For data transfer *to* a mobile host, the Snoop protocol improves TCP performance by deploying an agent at the base station, called the snoop agent. This agent aids in the quick, local recovery of data loss by taking advantage of the information conveyed in TCP acknowledgments (ACKs) from the receiver. The internal state it maintains is soft, in that it is not absolutely necessary for the correct end-to-end functioning of the protocol. Furthermore, it is very easy to regenerate and periodically refresh if it is lost or corrupted.

For data transfer *from* a mobile host, the protocol works by using a mechanism called *Explicit Loss Notification*. These are notifications generated by the base station and sent to the mobile sender as part of the ACK stream, which reacts to them by performing retransmissions, but *not* congestion control. In the first case (transfer to a mobile host), the Snoop protocol is an example of a *transport-aware* link protocol, while in the second case (transfer from a mobile host) it is a *link-aware* transport protocol.

The rest of this chapter is organized as follows. In Section 4.2, we specify the design goals and present an overview of the key ideas in the protocol. In Section 4.3 we describe the details of data transfer to a mobile host and in Section 4.4 the details of data transfer from a mobile host. In

Section 4.5, we describe how the Snoop protocol achieves good performance in the face of user mobility using a multicast-based low-latency handoff protocol. We discuss the implementation details in Section 4.6, performance results from our implementation in Section 4.7, and the results of detailed simulations on different workloads using an empirical error model in Section 4.8. We conclude this chapter with a summary in Section 4.9.

The next chapter contains a detailed performance evaluation of the Snoop protocol and quantitative comparisons with other protocols. There, we discuss the relative benefits of each of several techniques used in this and other protocols, and argue that exploiting cross-layer protocol optimizations is key to achieving good performance in error-prone wireless environments.

4.2 Overview

We start by presenting some preliminary measurements of TCP over a wireless network to illustrate the problem and understand the reasons for poor performance. We then discuss our design goals and present an overview of the Snoop protocol.

4.2.1 Preliminary Measurements

Figure 4.2 shows the packet sequence trace of a 2 MByte bulk TCP transfer between a fixed host connected to the base station via 16 Internet hops, communicating with a mobile host over a 2 Mbps WaveLAN link. These experiments were performed when there was little network congestion over the wide-area wired Internet, so any losses that occurred were due to wireless corruption. In the first case, we placed the base station and receiver radio units close to each other and ensured that no wireless losses occurred, while in the second case the radio units were in their normal locations in our testbed. We find that when wireless errors occur, TCP performance degrades significantly—in these experiments, a coarse packet error rate of about 5% led to a performance degradation of a factor of 4.5 from ideal.

A closer analysis of the packet sequence trace reveals the reasons for this. During the course of the transfer, packets are lost over the wireless link and need to be retransmitted. Every time the TCP sender detects the loss of a packet, it retransmits it, but also reduces its congestion window, ascribing the loss to be due to network congestion. This is the correct interpretation for wired networks,

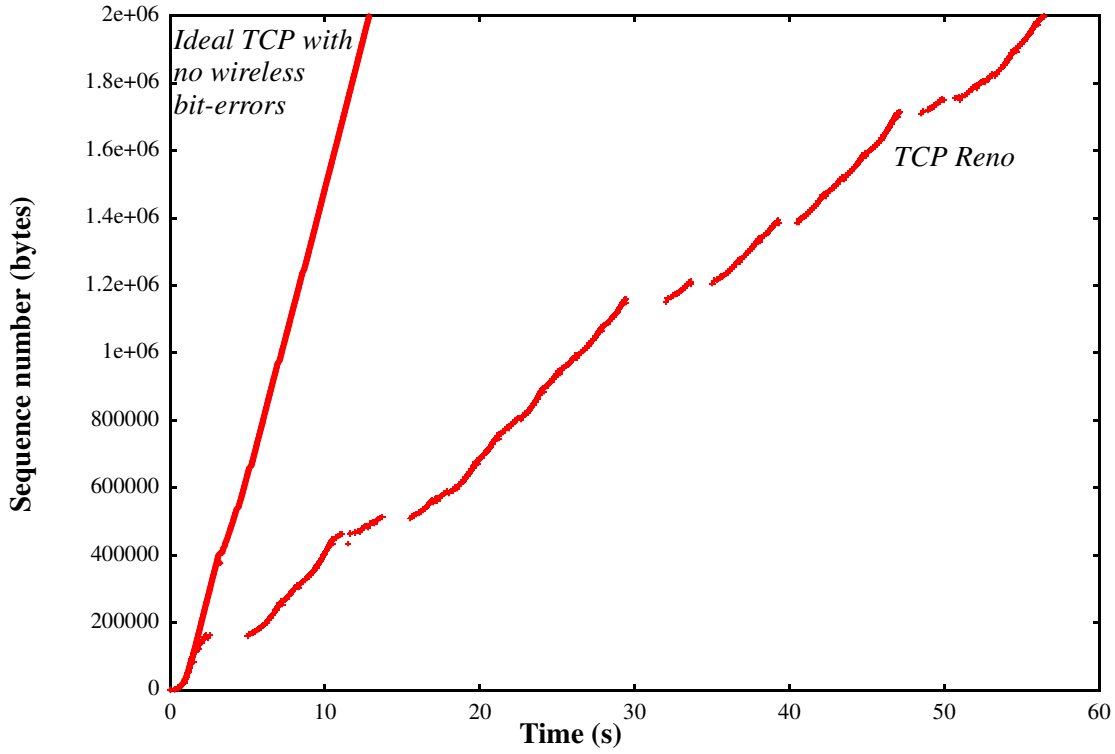

Figure 4.2 Sequence trace of TCP transfers over an error-free wireless link and a normal, error-prone link. The performance degradation compared to the ideal case is a factor of 4.5 in this experiment, because the sender misinterprets corruption losses as a sign of network congestion and incurs expensive timeouts that cause long idle times.

but is usually the incorrect response in networks that have wireless links. As a result, average windows are small and timeouts are frequent. Figure 4.2 shows the stalls that occur in the progress of the connection due to timeouts, leading to degraded performance.

Our goal is to improve the performance of TCP in such networks where non-congestion losses are common, in both local-area and wide-area network topologies.

4.2.2 Design Goals

In this section, we articulate our key design goals in developing a solution to the problems described above.

- *Develop a local solution:* When data is transferred from a fixed Internet host to a mobile host across both wired and wireless links, the problems caused by wireless links are *local* problems. A local solution to these problems is better than a sender-based one because loss recovery is faster in the former case and wired bandwidth is conserved. The Snoop protocol is one such mechanism—a localized wireless link protocol that improves end-to-end performance.
- *Eliminate adverse interactions between protocol layers:* In Section 2.5, we described the adverse interactions between conventional link-layer and end-to-end transport protocols. Our design explicitly avoids these interactions.
- *Enable incremental deployment:* Most current network applications that require reliable unicast transmission use TCP as their transport protocol. Therefore, it is desirable to achieve our goal of improving its performance in our network without changing existing TCP implementations in the fixed network. This will enable incremental deployment of our solutions.
- *Preserve end-to-end semantics:* We use the end-to-end argument in system design [127] as a guiding principle in our design. It states that a service should be implemented in a subsystem only if the service can be completely implemented in that subsystem, or if by *partially* implementing the service in the subsystem, *it substantially improves end-to-end performance.* We design our link-layer solution carefully and do not change the service model or semantics provided by TCP.
- *Use only soft state:* Another key design goal is the use of only *soft state* [31] in the protocol. The state maintained by the protocol must be *soft,* so that its loss does not impact the correct end-to-end functioning of the connection. If the state associated with the protocol is lost, it can be rebuilt easily upon the arrival of the next packet and ACK.

4.2.3 Protocol Description

The Snoop protocol modifies the networking software only at the base station and the mobile host connected over the wireless link. In particular, no modifications are required to TCP implementations elsewhere in the Internet. The protocol works by deploying a *snoop agent* at the base station. The agent mainly performs the functions of loss detection and loss recovery via retransmissions.

For transfer of data from a fixed host to a mobile host, we make modifications only at the base station. These modifications include caching unacknowledged TCP data and performing local retransmissions based on policies that depend on TCP ACKs from the mobile host and timeouts of locally-maintained timers. By using duplicate ACKs to identify packet loss and performing local retransmissions as soon as this loss is detected, the snoop agent shields the sender from the vagaries of the wireless link. In particular, transient situations of very low communication quality and temporary disconnectivity are hidden from the sender.

For transfer of data from a mobile host to a fixed host, the snoop agent detects segments corrupted over the wireless link at the base station by snooping on packets arriving from the mobile host and identifying holes in the transmission sequence. Then, when it detects duplicate ACKs from the receiver signifying a loss of data due to corruption, it sets a bit in their TCP headers and forwards them to the mobile sender. The sender uses this *Explicit Loss Notification (ELN)* information to identify that the loss was unrelated to congestion, and retransmits the packet *without* invoking any congestion-control actions. In this case, the protocol requires modifications to both the base station and mobile hosts.

Together, these mechanisms improve the performance of the connection in both directions, without sacrificing the end-to-end semantics of TCP or modifying host TCP code in the fixed Internet. This combination of greatly improved end-to-end performance, preservation of end-to-end semantics, and local recovery techniques is the main contribution of this work. In Chapter 5, we analyze the detailed reasons for improved performance and arrive at some general conclusions about the design of simple, yet highly-effective, link-layer protocols that do not interact adversely with TCP.

4.3 Data Transfer from a Fixed Host

We first describe the protocol for transfer of data from a fixed host to a mobile host through a base station. The base station's networking software is modified by adding the snoop agent, which monitors every packet that passes through the connection in either direction. Packets that arrive at the base station and destined for the mobile host now pass through the snoop agent between the IP and link layers. We emphasize that no transport layer code runs at the base station; rather, there is transport-aware link-level software that operates on all packets and ACKs.

The snoop agent maintains a cache of TCP packets[1] sent from the FH that have been forwarded to, but not yet been acknowledged by the MH. This is easy to do since TCP has a cumulative ACK policy (Section 2.2.1). When a new packet arrives from the FH, the agent adds it to its cache and passes the packet on to the forwarding code which performs the normal packet forwarding functions. The snoop agent also monitors TCP ACKs sent from the mobile host, using new ACKs to clean its cache and maintains only unacknowledged packets there. It detects the loss of a packet by the arrival of duplicate ACKs or by a timeout based on locally maintained round-trip timers. When it does, it retransmits the lost packet to the MH if it has the packet cached, since these are packets that are lost due to wireless corruption. Additionally, it *suppresses* duplicate ACKs corresponding to wireless losses from the TCP sender, thus ensuring that false congestion control invocations associated with fast retransmissions are avoided. If the agent does not have the packet cached, then it was lost earlier in the path due to congestion. The corresponding duplicate ACKs are forwarded to the TCP sender, allowing fast retransmissions and congestion control to occur as usual. Thus, the snoop agent at the base station hides non-congestion-related wireless packet losses from the FH sender using local retransmissions and by suppressing certain duplicate ACKs; at the same time, it ensures that the sender continues to react appropriately to network congestion.

The snoop agent has two main components: *data processing* and *ACK processing*. The flowcharts summarizing the salient features of the algorithms for these steps are shown in Figures 4.3 and 4.4. The remainder of this section describes these algorithms in detail.

4.3.1 Data Processing

In this phase, the snoop agent processes incoming data segments from the fixed host. A TCP segment is identified by the sequence number of its first byte of data and its size[2]. At the BS, the snoop agent keeps track of the last sequence number seen for the connection. At any stage in the protocol, one of several kinds of packets can arrive at the BS from the FH, and the snoop agent processes them in different ways:

---

1. We use the terms “packet” and “segment” interchangeably. Where the distinction is important, we explicitly point it out.
2. On a sender retransmission, TCP implementations may re-segment data. We handle this case correctly in the implementation of the snoop agent.

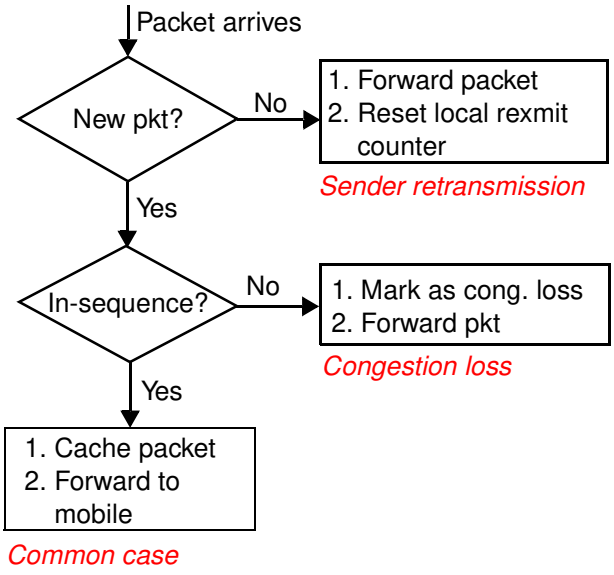

Figure 4.3 Flowchart for data processing.

1. *A new packet in the increasing TCP sequence:* This is the common case, when a new packet in the normal increasing sequence arrives at the BS. In this case the packet is added to the snoop cache and forwarded on to the MH. The agent does not perform any extra copying of data while doing this. In addition, it logs the current transmission time to later estimate the round-trip time of the connection over the wireless link.

2. *An out-of-sequence packet that has been cached earlier:* Although this is an uncommon case, it can and does happen. First, if there have been multiple losses in a single window due to congestion on the wired path, there are times when a timeout and slow start occur after a fast retransmission. This could lead to a sender retransmission of a previously cached segment. Second, if there is persistent congestion over the wireless link, local retransmissions are not triggered from the base station's snoop agent. This implies that the TCP sender recovers from such losses by retransmitting packets. Finally, if the wireless link has been error-prone for several hundred milliseconds, the sender could time out and retransmit the missing segment, leading to this situation. In our experience, the first two situations are the common reasons for the occurrence of this condition.

   If the sequence number is greater than the last ACK seen, it is very likely that this packet didn't reach the MH earlier, and so it is forwarded on. If, on the other hand, the sequence number is less than the last ACK, this packet has probably already been received by the MH. At this point,

there are several possible actions the agent could take. One possibility would be to discard this packet and continue, but this is not always the best thing to do. The reason for this is that the original ACK with the same sequence number could have been lost due to congestion while going back to the FH. The second possibility, to facilitate the sender getting to the current state of the connection as fast as possible, is for the snoop agent to send a TCP ACK corresponding to the last ACK seen at the BS (with the source address and port corresponding to the MH) to the FH. However, the disadvantage of this is that if the information in the snoop agent's state is wrong for any reason, the correctness of the end-to-end protocol is compromised. The third option is to simply forward the packet to the MH, and await information from subsequent ACKs to refresh the state at the agent. This is the option the agent uses, in keeping with our soft-state design philosophy. The agent also resets the number of local retransmissions to zero, and updates the transmission time of this segment to correctly estimate the round-trip time when its ACK arrives.

3. *An out-of-sequence packet that has not been cached earlier:* In this case the packet was either lost earlier due to congestion on the wired network, or has been delivered out-of-order by the network. The former is more likely, especially if the sequence number of the packet (i.e, the sequence number of its first data byte) is more than one or two packets away from the last one seen so far by the snoop agent. This packet is cached by the agent and then forwarded to the MH. It is also marked as having been retransmitted by the sender. The ACK processing algorithm uses this information to process duplicate ACKs that arrive for this packet from the MH.

4.3.2 ACK Processing

In this phase, the snoop agent monitors and processes the ACKs sent back by the MH and performs various operations depending on the type and number of ACKs it receives. These ACKs fall into one of four categories:

1. A *new*, *expected* ACK: This is the common case that occurs when no recent segments have been lost, and signifies an increase in the packet sequence received at the MH. This ACK initiates the cleaning of the snoop cache and all acknowledged segments are freed. The round-trip time estimate for the wireless link is also updated at this time. Finally, the ACK is forwarded to the FH.

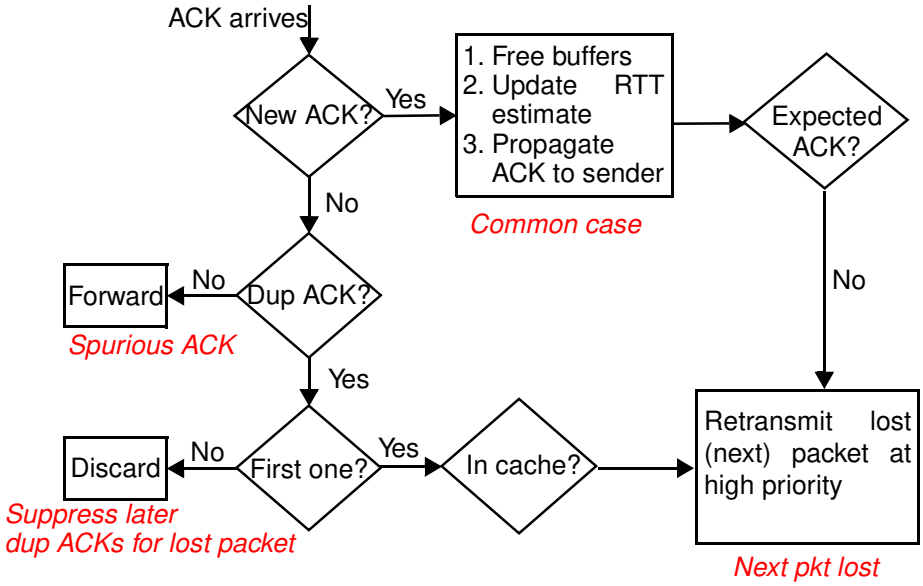

Figure 4.4 Flowchart for ACK processing.

2. A *spurious* ACK: This is an ACK less than the last ACK seen by the snoop agent, and is a situation that rarely occurs. Just as in case (2) of the data processing algorithm, it could be discarded. Instead, we forward it to the FH, to guard against the possibility that the internal state of the snoop agent may be incorrect.

3. A *duplicate* ACK (DUPACK): This is an ACK that is identical to a previously received one. In particular, it is the same as the highest cumulative ACK seen so far, and does not advertise a different receiver window from the previous one. In this case the next packet in sequence from the DUPACK has not yet been received by the MH. However, some subsequent packets in the sequence have been received, since the MH generates a DUPACK for each TCP segment received out of sequence. One of several actions is taken depending on the type of DUPACK and the current state of the snoop agent:

- The first case occurs when the DUPACK is for a packet that is either not in the snoop cache or has been marked during data processing as having been retransmitted by the sender. If the packet is not in the cache, it needs to be resent from the FH, perhaps after invoking the necessary congestion control mechanisms at the sender. If the packet was marked as a sender-retransmitted packet, the DUPACK needs to be propagated to the FH because the TCP sender

maintains state based on the number of duplicate ACKs it receives when it retransmits a packet. Therefore, both these situations require the DUPACK to be routed to the FH. This is especially important to help the sender perform fast recovery (Section 2.2.2) correctly.

- The second case occurs when the snoop agent gets a DUPACK that it does not expect to receive for the packet. This typically happens when the first DUPACK arrives for the packet after a subsequent packet in the stream reaches the MH, following a packet loss. The arrival of each successive packet in the window causes a DUPACK to be generated for the lost packet. To make the number of such DUPACKs as small as possible, the agent retransmits the lost packet as soon as the loss is detected at a *higher priority* than normal packets (how this is done is described in Section 4.6). In addition, the snoop agent estimates the maximum number of DUPACKs that can arrive for this packet. This is done by counting the number of packets that were transmitted after the lost packet prior to its retransmission. The snoop agent also updates its estimate of the next expected new ACK, once the losses in this window are recovered. This is equal to one more than the last data byte sent prior to the loss.
- The third case occurs when an "expected" DUPACK arrives, based on the above maximum estimate. The missing packet would have already been retransmitted when the first DUPACK arrived (and the estimate was zero), so no retransmissions occur at this stage. In practice, the retransmitted packet reaches the MH before most of the later packets do (because it was retransmitted at a higher priority) and the BS sees an increase in the ACK sequence before all the expected DUPACKs arrive.

  In addition, we *suppress* this DUPACK by discarding it, thereby preventing the sender from receiving and reacting to it. This is because we have already ensured a retransmission attempt for the corresponding segment, and do not want the sender to perform a fast retransmission and invoke congestion-control actions in response to this perceived loss.

4. A *new, unexpected* ACK: During the processing of DUPACKs following a set of losses in a window, the snoop agent keeps track of the first new ACK it expects to see when all losses are recovered from. If the new ACK is smaller than this estimate, we call it "unexpected." The most likely cause for this is that the next segment was lost as well, for otherwise the new ACK would

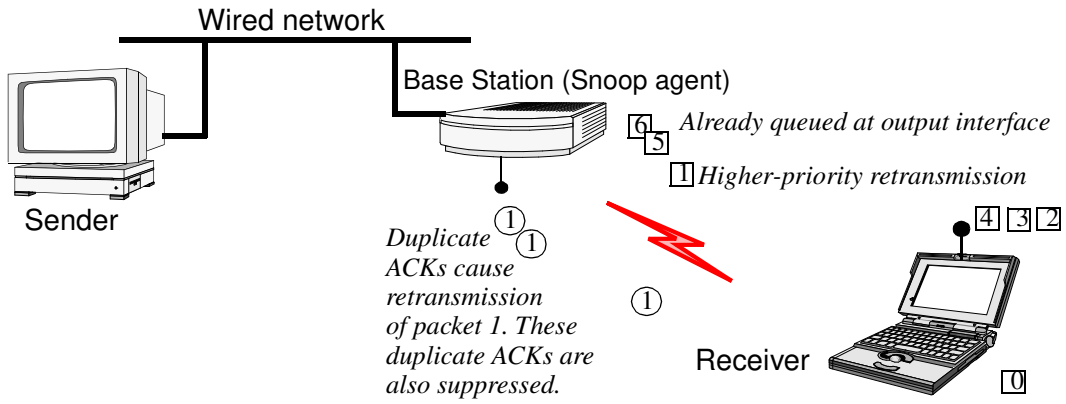

Figure 4.5 Higher-priority retransmissions from the Snoop agent ensure that the new ACK generated is smaller than if retransmissions were queued at the tail of the interface queue. In this example, packet 1 was lost and is being retransmitted at higher priority. When it reaches, the new cumulative ACK would be 4. If the retransmission were not done at higher priority, two more duplicate ACKs would be generated (when 5 and 6 reach), and the new ACK generated when 1 eventually reaches would be 6 (compared to 4).

have equalled the next expected one. In addition, no DUPACKs may arrive for this loss because all the DUPACKs in the window arrived for the previous loss. Thus, after cleaning the cache and freeing all acknowledged segments, we retransmit the next missing one.

Retransmitting packets at a higher priority improves performance at all error rates. This enables retransmitted packets to reach the mobile host sooner, reducing the number of duplicate ACKs. It reduces the sensitivity of the protocol to accurately maintaining a count of the number of expected duplicate ACKs, since ACKs could get lost during loss periods. In the common case when the wireless link is the connection's bandwidth bottleneck, the first new ACK generated after a higher-priority retransmission is less than what it would be without such prioritization, as shown in Figure 4.5. This implies that the number of segments transmitted by the sender in a burst is smaller in the former case, since the sender's sliding window would shift by a smaller amount. This in turn reduces the degree of congestion in the network by reducing the occurrence of bursts.

The snoop agent also keeps track of the number of local retransmissions for a packet, and resets this number to zero if the packet arrives again from the sender following a timeout or a fast retransmission. In addition to retransmitting packets depending on the number and type of ACKs (*data-driven retransmissions*), the snoop agent also performs retransmissions driven by timeouts. This is described in more detail in Section 4.6.

4.4 Data Transfer from a Mobile Host

We now proceed to discuss the case of data transfer from a mobile host. While this has conventionally been a less-common scenario for wireless information access, it is rapidly gaining in importance. There are an increasing number of Web and other information servers that serve pages from behind wireless links [85, 64].

It might be tempting to come to the conclusion that a protocol very similar to the one described in earlier sections for data transfer to the mobile host can be used in this case as well. In particular, one of the important features of the previous case was that it required changes only to the base station and not to any other entities in the network. However, it is unlikely, if not impossible, that a protocol with modifications made *only* at the base station can substantially improve the end-to-end performance of reliable bulk data transfers from the MH to other hosts on the network, while preserving the precise semantics of TCP ACKs. For example, simply caching packets at the base station and retransmitting them as necessary will not be particularly useful, since packet corruption is likely to occur en route from the mobile host to the base station. Running a snoop agent at the mobile host will be inappropriate because the same duplicate ACKs signify both corruption and congestion losses. There is no way for the mobile sender to know if the loss of a packet happened on the wireless link or elsewhere in the network due to congestion.

There are at least two ways in which the basic snoop scheme can be enhanced to achieve good performance for this case—the first involves the use of negative acknowledgments and the second the use of Explicit Loss Notifications (ELN). We first describe the first approach, and then the second, which we believe to be an elegant solution to the problem that has the advantage of simplicity. In the next chapter, we show how the ELN-based algorithm naturally generalizes to the case when the wireless link is a transit link bridging two or more wired networks.

4.4.1 Using Negative Acknowledgments

An agent at the base station keeps track of the packets that were lost in any transmitted window and generates negative acknowledgments (NAKs) for those packets back to the mobile sender. This is especially useful if several packets are lost in a single transmission window, a situation that happens often under high interference or in fades where the strength and quality of the signal are low. These NAKs are sent when either a threshold number of packets (from a single window) have

reached the base station or when a certain amount of time has expired without any new packets from the mobile. Encoding these NAKs as a bit vector can ensure that the fraction of the sparse wireless bandwidth consumed by NAKs is relatively low. The mobile host used these NAKs to selectively retransmit lost packets.

NAKs can be implemented using TCP's Selective Acknowledgment (SACK) option [92, Section 2.3.1]. The snoop agent could use SACKs to enable the mobile host to quickly (relative to the round-trip time of the connection) retransmit missing packets. The only change required at the mobile host is to enable SACK processing. No changes of any sort are required in any of the fixed hosts. Also, SACKs are generated by the snoop agent when it detects a gap corresponding to missing packets; we emphasize again that no transport-layer code runs at the base station to do this.

There is a danger of a possible violation of end-to-end semantics in this scheme, because the snoop agent would send SACK information about segments it receives, but they may not yet have been received by the intended receiver. This will lead to problems if the corresponding segments get lost later in the path. However, this is not strictly true because of the semantics of TCP SACKs: they are *advisory* in nature, and only cumulative ACKs are binding. What this means is that even if a TCP sender receives SACKs for certain data blocks, it must not clean those segments from it's transmission buffer or assume successful reception, until cumulative ACKs arrive for them.

However, this approach based on SACKs is inelegant because it relies on a relatively obscure feature of the SACK specification. Of course, we could implement an explicit NAK scheme, but that would require complex modifications at the sender, which would anyway implement SACKs in the future. In addition, we would like to avoid explicit protocol messaging as far as possible, and both SACKs and NAKs generated from the base station do not provide this. All these issues led us to investigate alternate protocols; what follows is one such scheme.

4.4.2 Using Explicit Loss Notifications

Explicit Loss Notification (ELN) is a mechanism by which the reason for the loss of a packet can be communicated to the TCP sender. In particular, it provides a way by which senders can be informed that a loss happened because of reasons unrelated to network congestion (e.g., due to wireless bit errors), so that sender retransmissions can be decoupled from congestion control. If the receiver or a base station knows for sure that the loss of a segment was not due to congestion, it

sets the ELN bit in the TCP header and propagate it to the source. In the situation at hand, this ELN message is sent as part of the same connection (and not in a separate way, using ICMP for instance). This simplifies the sender implementation as it receives messages in-band. ELN is a general concept that has applications in a wide variety of wireless topologies. In this section, we describe it in the context of data transfer from a mobile host connected to the rest of the Internet via a cellular wireless link. The next chapter discusses how it helps improve end-to-end performance over wireless transit links.

The snoop agent running at the base station monitors all TCP segments that arrive over the wireless link. However, unlike in the previous case, it does not cache any TCP segments since it does not perform any retransmissions. Rather, it keeps track of *holes* in the sequence space as it receives data segments, where a hole is a missing interval in the sequence space. These holes correspond to segments that have been lost over the wireless link. However, it could also be the case that the packet was lost due to congestion at the base station. To avoid against wrongly marking a congestion hole as having been due to a wireless loss, it only adds a hole to the list of holes when the number of packets queued on the base station's input interface is not close to the maximum queue length.

When ACKs, especially duplicate ACKs, arrive from the receiver, the agent at the base station consults its list of holes. It sets the ELN bit on the ACK if it corresponds to a segment in the list before forwarding it to the data sender. It also cleans up all holes with sequence numbers smaller than the current ACK, since they correspond to segments that have been successfully received by the receiver. When the sender receives an ACK with ELN information in it, it retransmits the next segment, but does not take any congestion control actions. The sender also makes sure that each segment is retransmitted at most once during the course of a single round-trip, as the snoop agent would flag an ELN for *each* duplicate ACK following a loss.

4.5 Mobility

As a mobile host moves, it may get out of the range of its current base station but still be within the range of other neighboring base stations. To maintain the mobile host's connectivity, a *handoff* is performed to re-route traffic to and from the mobile host via the new base station. However, depending on the details of the handoff algorithms, this procedure could lead to packet losses and

reordering, which in turn could cause significant deterioration in the performance of ongoing TCP transfers [25].

In Section 2.5.5, we discussed the details of some schemes to improve TCP performance when users are mobile [25, 8, 131]. In this section, we describe how the Snoop protocol is integrated with a low-latency handoff protocol that leads to good TCP performance in the presence of both bit-errors and user mobility.

A small amount of buffering at and retransmission from base stations prevents packet loss during the short handoff period. In [26], the buffering happens at the mobile host's old base station, which forwards packets to the new base station at the time of handoff. We use a multicast-based scheme, where one or more base stations in the vicinity of the current one join a multicast group corresponding to the mobile host and receive all packets destined to it [131]. When the handoff occurs, the new base station is readily able to forward the buffered and the newly arriving packets without incurring any losses or introducing any delay or reordering. This prevents unnecessary invocations of TCP fast retransmissions.

In contrast to these schemes that operate at the network layer, handoffs in a split-connection context (e.g., I-TCP [10]), involve the transfer of transport-layer state from the old base station to the new one. This results in significantly higher latency; for example, [8] reports I-TCP handoff latencies of the order of hundreds of milliseconds in a WaveLAN-based network. As we show in Section 4.7.2, this is over an order of magnitude slower than our scheme.

4.5.1 Multicast-based Low Latency Handoffs

As in Mobile IP [112], all packets arriving from a fixed host are intercepted by an entity called a home agent. Corresponding to each mobile host is a pre-assigned multicast address, which the home agent knows. The home agent encapsulates every packet intended for the mobile host using this multicast address and forwards it as shown in Figure 4.6.

At the same time, the mobile host chooses a small number of base stations in its current neighborhood (based on metrics like the received signal strength, error rates, etc. computed from periodic beacon signals) to join this multicast group on its behalf. One of these base stations is set up as the primary and is the *forwarding* base station. The other secondary base stations are *buffering* base

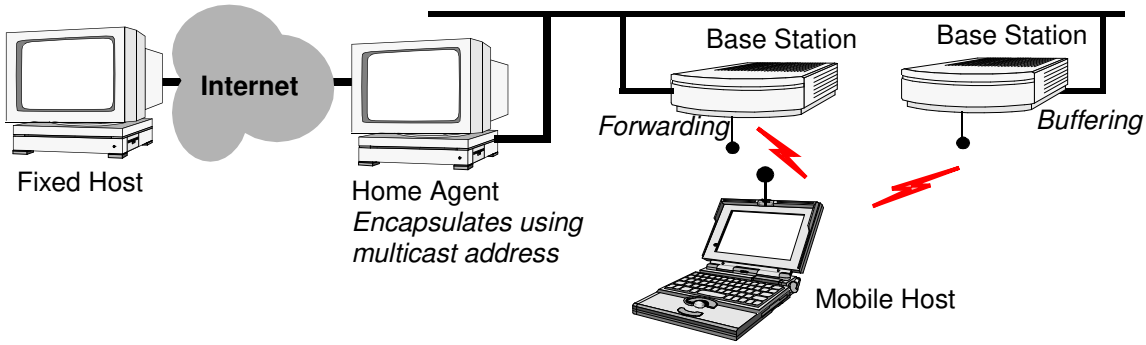

Figure 4.6 Multicast-based handoffs.

stations, storing a small number of recent packets in anticipation of a handoff. In our system, handoffs are initiated by the mobile host, which chooses one of the other base stations (typically a buffering base station) to become the primary forwarding one and sends a message to the current primary base station informing it to stop forwarding packets.

The snoop agent relies on a cache of unacknowledged packets at the base station to improve end-to-end TCP performance. After a handoff, the new base station must have the current set of unacknowledged packets in its cache to provide improved performance based on local retransmissions, when losses occur due to packet corruption. It is clear how this is achieved by using our multicast-based scheme.

Multicast allows the snoop agents at the buffering base stations to *mirror* the state present in the primary base station for the different TCP connections. Using these packets, the agents build up their caches for various connections to the mobile host. However, during this period ACKs *from* the MH continue to pass only through the *primary BS*, the mobile host's default router. Therefore, the nearby buffering base stations cannot snoop on any ACKs for the caches they are mirroring. This prevents them from promptly freeing up packets that have safely reached the MH. The caches at these base stations are maintained using a "discard-oldest" policy, since we would like to have the most recent packets at a new base station as soon as a handoff occurs.

Once the handoff occurs, ACKs begin to pass through the new forwarding BS. The first ACK the BS receives identifies which packets have been received by the MH on that connection. The BS uses this information to clean and synchronize the contents of the snoop cache to the last state of the previous primary BS.

The state of the new and previous BSs may not be identical after the synchronization. The new primary BS may be missing several packets from its snoop cache. This is because it may have either missed several packets while it was in buffering mode (due to congestion) or buffered packets for only a short period of time before the handoff. However, none of this affects the correct functioning of the connection because this state is soft. Since the snoop agent does not change any of the end-to-end semantics of TCP, the protocol is resilient to these gaps in its state. As the connection progresses the gaps in the agent's state are resolved. Our experience has been that it takes only a very small number of packets (usually less than four) before the Snoop agent at the new base station reaches the current state.

The duration of a handoff is short since no state is explicitly transferred between BSs. The state transfer is avoided because the new primary BS mirrors the state of the previous primary BS prior to handoff. This handoff scheme attains low-latency for TCP streams since multicast provides a simple, lightweight way to mirror state changes at the Snoop agents.

4.6 Implementation

We have implemented the Snoop protocol on a testbed consisting of IBM ThinkPad laptops and Pentium base stations running BSD/OS 3.0 from BSDI, communicating over a Lucent WaveLAN. The implementation currently supports bulk transfers to and from mobile hosts and is integrated with mobility software that provides smooth, low-latency handoffs described in the previous section.

Recall from Section 4.2.2 that one of the key design goals in the Snoop protocol was the use of *soft state*. This makes crash recovery and state management during mobility easy, unlike in approaches involving hard state, where a crash might compromise an existing connection or user mobility might entail explicit state transfer. This design goal was faithfully followed in our implementation, where care was taken to ensure that there was little permanence or criticality to the state maintained by the snoop agent, as far as the *correct* working of the end-to-end connection was concerned. The state maintained by the snoop agent can be reconstructed from scratch by snooping on a few packets and ACKs.

Although the Snoop protocol is conceptually a transport-aware *link* protocol when the direction of data transfer is from a FH to a MH, we implement it in the network (IP) layer of the BSD kernel. This is done primarily for reasons of simplicity, because we have ready access to the IP and TCP headers of the packet without any Ethernet-level headers at this point. This also enables easy integration with handoff modules that reside at the network layer of the stack.

When a TCP segment arrives from or is intended for a MH, it passes on to the Snoop protocol control function. This function determines which specific agent to invoke, depending on the connection, and if necessary, instantiates one.

4.6.1 State Management

When a TCP segment arrives from or destined for a MH, the snoop control code attempts to obtain the per-connection data structure from its hash table of connections. If this attempt fails, a new agent is created and initialized for the connection. A new agent is also created if the attempt succeeds, but the segment that just arrived is a SYN segment, because this signifies the start of a new connection. At the same time, the old agent (if there is one) is freed and its space reclaimed. We do not constrain the creation of a new segment to happen only when a SYN segment arrives, in keeping with our soft state design philosophy. By doing this, we ensure that agent setup is spontaneous and automatic upon the arrival of any segment.

The state maintained by the snoop agent is cleaned, and any cached packets freed, when a segment with the FIN or RST flags arrive. However, clean-up of agent state is not dependent on the arrival of a FIN—state is also cleaned if there has been no activity on the connection for a time specified by the *garbage collection timer,* set to 20 seconds in our implementation. If the end-to-end connection has not terminated and is in fact continuing after a long idle time, nothing untoward happens— agent state is implicitly instantiated upon the arrival of the first segment on the connection. In contrast, hard state implementations like split connections are constrained to keep the TCP connection open and consume control block state at the BS even when the connection is idle for long periods of time. Furthermore, they are vulnerable to mistakes in BS state or to BS crashes, a drawback not present in the soft state implementation that the Snoop protocol enables.

Finally, we note that state management is complicated by the bi-directional nature of TCP connections. Data can flow in both directions of the same connection, and ACKs are sometimes piggy-

backed on data segments. When a FIN segment arrives at the BS, only the state in the agent for that direction of the connection is cleaned; in general, the other direction will still be active. The garbage collection timer cleans state in both directions, since it is triggered only when both directions have been idle[1]. Another complication arises because of temporary routing pathologies when a MH moves between cells. There could be periods of time during which data packets go through one BS (via an active snoop agent), but ACKs may go through another BS. This situation can be detected, since the difference between the last sequence number and the last ACK seen at the BS would be larger than the maximum transmission window size. When this happens, the agent ensures that the replacement policy in the snoop cache is LRU; i.e., recently arriving segments are cached by replacing the oldest ones. The agent also disables the local retransmission timer until the routing inconsistency is resolved.

4.6.2 Data Transfer from a Fixed Host

To achieve good TCP performance in this case, we need to modify only the software at the BS. The mobile host and fixed hosts on the Internet are unchanged.

Data structures: The data used by the snoop agent are hierarchically organized. At the highest level is a hash table that uses the source and destination ports as a key and hashes into a pointer to the per-connection state. The per-connection state consists mainly of the snoop packet cache and some other associated information. The snoop cache is maintained as a circular (ring) buffer of packets, consisting mainly of pointers to kernel mbufs [97]. The state maintained for each packet includes the packet sequence number, its size, the number of attempted local retransmissions, and a flag set if the packet was thought to have been retransmitted by the sender. For each connection, we also maintain some other associated information, as depicted in Figure 4.7. Since this soft state takes up only 54 bytes per active connection, most of the used memory is due to the packets themselves. In general, the size of the cache at a forwarding base station needs to be large enough to handle the maximum transmission window size.

1. We could have maintained independent garbage collection timers for each direction, but decided against it because we wanted to minimize the number of timers and timer interrupts in the protocol.

```
typedef struct wi_conn_state { /* FH --> MH */
        u_short  wi_state;      /* current state of snoop (wired sender) */
        u_short  wl_state;      /* current state of snoop (wireless sender) */
        u_long   addr;          /* address of FH (wired side) */
        u_long   wladdr;        /* address of MH (wless side) */
        u_short  port;          /* wired side port */
        u_short  wlport;        /* wireless side port */
        tcp_seq  last_seen;   /* first byte of last packet buffered */
        tcp_seq  last_ack;    /* last byte recd. by MH for sure */
        u_short  last_win;      /* last flow control window size from receiver */
        tcp_seq  expected_next_ack;  /* expected next ACK on connection */
        short    expected_dacks;        /* expected number of duplicate ACKs */
        tcp_seq  iss;           /* initial seq # for this connection (used for debugging) */
        long     srtt;          /* smoothed rtt estimate */
        long     rttdev;        /* smoothed rtt linear deviation */
        short    bufhead;       /* next pkt goes here in ring buffer */
        short    buftail;       /* first unack'd pkt in ring buffer */
        short timeout_pending;          /* timeout pending on this connection */
        packet_t *pkts[];       /* ring buffer of unack'd packets (snoop cache) */
} wi_conn_state_t;

typedef struct wl_conn_state { /* MH --> FH */
        tcp_seq  wl_last_seen;    /* last seq  number seen from wireless side (MH) */
        tcp_seq  wl_last_ack;     /* last ACK seen */
        short    wl_bufhead;       /* head of seq_t ring buffer */
        short    wl_buftail;       /* tail of ring buffer */
        seq_t    *wlseqs[];        /* ringbuf of seq #s and sizes */
} wl_conn_state_t;
```

Figure 4.7 Per-connection data structures used in the implementation of the Snoop protocol. For data transfer from a FH, the space used is 54 bytes. Thus, most of the space is used by the TCP segments themselves. For a transfer from a MH, typical space consumption is 16 bytes, plus 8 bytes per hole to store the hole's sequence number and size.

Data processing: The implementation of this phase consists of a caching step and a forwarding one. Several studies have shown that one of the predominant implementation costs of TCP arises due to the copying of data [34, 73]. We use the reference counting mechanism in kernel mbufs to avoid data copying in the snoop agent. Thus, we do not incur any extra overhead associated with copying at the base station. Thus, the overhead due to the protocol implementation is negligible in the common case when segments are arriving in increasing order: an incoming segment is added to the cache without copying it, and it is forwarded on to the mobile host. A small number of state variables (the last sequence number seen and a flag indicating that the connection is still active) are updated.

To make it easy to clean acknowledged data from the cache and also perform local retransmissions, the cache is maintained in increasing sequence order. Thus, when out-of-sequence data arrives at the snoop agent, we may need to reorganize the ring buffer. If the arriving segment is smaller in sequence than the lowest segment currently cached, then we prepend it to the cache and forward the segment on. If not, we traverse the cache and locate the position of the current segment, shift all the earlier segments to the left, and insert the current one at that position. Finally, the cache might be full, in which case we eliminate the oldest segment and cache the current recent one. Continuing to save old segments is inappropriate in general because the mobile host may have moved out of range of the current base station and be sending (and receiving) its packets via another BS.

ACK processing(): When a new ACK arrives at the base station, we forward it on to the fixed host and clean the snoop cache by freeing the packets corresponding to packets already acknowledged by the mobile. We also update the estimate of the wireless link's round-trip time and linear deviation, and restart the pending retransmission timer. When a duplicate ACK arrives, we perform the algorithm described in Section 4.3.2.

Retransmissions: In addition to retransmitting packets depending on the number and type of ACKs received as described in Section 4.3.2, the snoop agent also performs retransmissions triggered by timeouts. The retransmission timer is based on the estimate of the smoothed round-trip time (srtt) of the last link. We compute this using the standard exponential weighted moving average technique, srtt = (1-α)*old_srtt + α*curr_rtt, with α set to 1/8. We also estimate the mean linear deviation of the round trip time similar to TCP (Chapter 2) and retransmit a packet if we do not receive an ACK in a time equal to the smoothed estimate plus four times the linear deviation. To limit the amount of time spent processing timer interrupts, we do not timeout more frequently than a minimum threshold time, set to 200 ms in our implementation.

On both data-driven and timer-driven retransmissions, the segment retransmission function ensures that the number of retransmissions is less than a configurable maximum threshold (whose default value is 8). If so, then it sets the IP type-of-service (TOS) field to IPTOS_LOWDELAY in the IP header of the packet and forwards the packet on to the destination. This is an indicator to the lower level interface queues to transmit this packet at a higher priority than currently enqueued packets.

4.6.3 Data Transfer from a Mobile Host

To achieve good TCP performance in this case, we need to make modifications to both the BS and the MH, which is the TCP sender. Fixed hosts in the rest of the Internet are unchanged.

Data structures: At the BS, the snoop agent detects holes in the data transmission from the MH. However, unlike in the previous case, it does not have to cache any data segments, which makes its implementation simpler. The basic data structure used for this purpose is a list of blocks, maximal contiguous segments of data specified by a starting sequence number and size. Gaps between blocks correspond to missing segments in the transmission sequence. At the MH, a new variable is added to the connection's TCP control block to keep track of the last ELN-triggered retransmission.

Data processing: When new data arrives from the MH at the BS, the agent attempts to coalesce it with an existing block and checks if it bridges a previous hole. A new data segment that is out of the normal increasing sequence creates a hole, because segments in between have been lost. The agent ensures that every hole was the consequence of a corruption-induced loss, and not due to congestion. Congestion-induced losses can occur in two ways—due to the base station's input queue overflowing, or due to the mobile host's output queue overflowing. Piggybacked on each packet from the MH is its instantaneous output queue length, which the base station can use to classify each loss it detects. When a new, out-of-sequence packet arrives at the BS, the snoop agent checks the size of its input queue and gets the size of the MH's output queue from the packet. If either of the instantaneous queue sizes is above a certain threshold of the maximum (set to 75% in our implementation), then this loss is not considered as one due to corruption. Because the agent should only flag ELN information for those losses that were unrelated to congestion, the implicitly maintained list of holes should only include packets lost due to corruption.

ACK processing: Whenever an ACK arrives at the BS, the agent updates the list of blocks by cleaning up all blocks (or parts of blocks) that have been acknowledged so far. It also checks if the ACK corresponds to a hole, i.e., if the next segment in sequence was lost due to corruption on the wireless link. It does so by comparing the ACK to the sequence number of the earliest block currently in its list for that connection. If the value of the ACK is smaller, it sets the ELN bit in the TCP header, modifies the TCP checksum, and forwards the ACK on to the MH. Because there is

currently no specific bit in the TCP header for ELN, we use one of the unreserved bits for this purpose. Note that while ELN marking happens most often for duplicate ACKs, it can (and does) also happen for new ones. However, it does not happen when the list of blocks is empty, because there is no way the agent at the BS can know if any further data has been transmitted by the mobile host.

When the MH receives an ACK with the ELN bit set, it retransmits the missing packet and updates a variable (eln_last_rxmit) that keeps track of the last ELN-induced retransmission. This ensures that the MH does not retransmit the same packet on every ACK with ELN that it receives via the base station; thus, the retransmission policy is left to the end-host, and the base station only conveys relevant information to it. The TCP stack at the MH uses a configurable variable, eln_rexmt_thresh, to determine when to retransmit a segment upon receiving an ACK with ELN set. Upon receiving eln_rexmt_thresh ACKs with ELN, it retransmits the missing segment. In our implementation and experiments, we set its value to 2.

When the MH retransmits a segment based on ELN information, it does not reduce its congestion window and perform congestion control. It also bypasses the fast recovery code that inflates the congestion window for every duplicate ACK. Finally, we note that these actions are taken only for duplicate ACKs with ELN, and not for other duplicate ACKs that signify network congestion.

4.7 Implementation Performance

We performed several experiments with the Snoop protocol in our WaveLAN-based wireless testbed and compared the resulting performance with TCP Reno. We present the results of these experiments in this section. In the absence of packet losses, the maximum throughput achieved by a TCP connection was about 1.5 Mbps.

4.7.1 Wireless Bit-Errors

Figure 4.8 shows sequence traces of TCP transfers over an error-free link, as well as TCP Reno and the Snoop protocol over error-prone links. The middle curve shows the progress of a transfer using the Snoop protocol, which is a curve close to the ideal, error-free case. The performance improvement in this transfer over TCP Reno is a factor of four. This is typical of the degree of improvement we obtain in our WaveLAN-based wireless testbed for reliable data transfers.

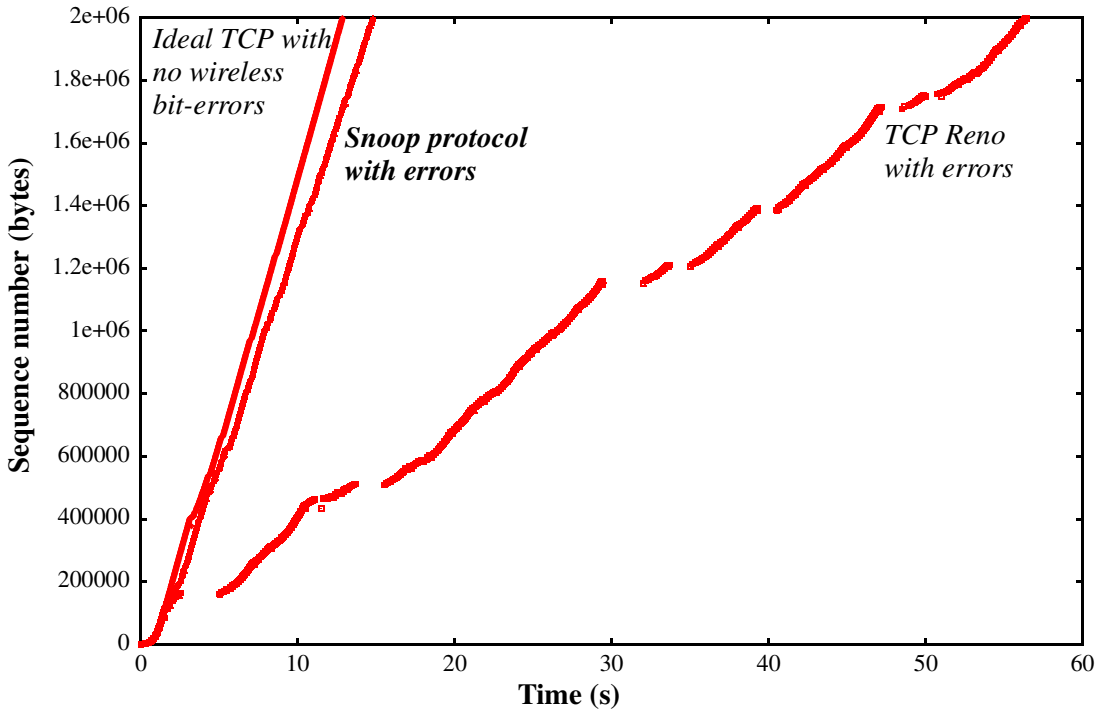

Figure 4.8 Sequence traces of wide-area TCP transfers over a last-hop WaveLAN wireless link, using an ideal error-free link, as well as TCP Reno and the Snoop protocol over error-prone links. The middle curve shows the performance improvement over TCP Reno and contrasts it to Figure 4.2.

We now discuss the results of measurements under controlled error conditions. We first present the results of data transfer from a fixed sender to a mobile receiver. The maximum possible window size for the connection was 64 KBytes and the maximum TCP segment size was 1440 bytes.

To measure the performance of the implementation under controlled conditions, we used an exponentially-distributed bit-error model. We emulated bit-errors using an exponential distribution for each bit-error rate. The emulator changed the IP checksum of the packet at the receiver if the error generator determined that the packet should be dropped. This was done on both sides of the wireless link, so both TCP data and ACKs were corrupted. The choice of exponential distribution was not intended to model reality, but to characterize the relative improvement in performance at each controlled bit-error rate. In Section 4.8, we discuss the performance of the Snoop protocol using traces of errors collected from a real testbed. Each run involved a 10 MByte transfer that was repeated twenty times at each error rate.

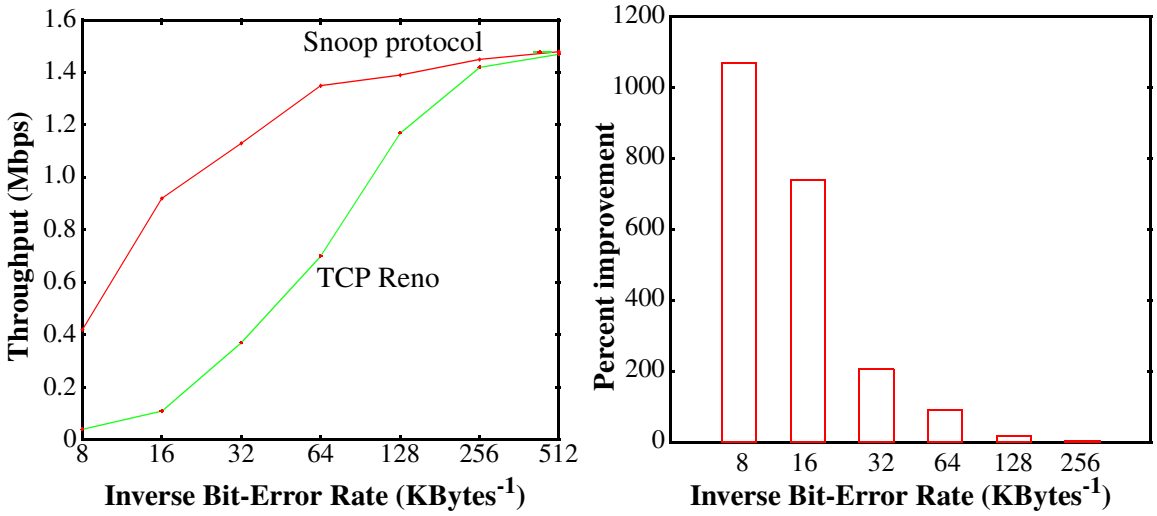

Figure 4.9 Performance of TCP Reno and the Snoop protocol for a bulk transfer across a range of bit-error rates. The figure on the left shows the throughput, while the one on the right shows the percentage improvement due to the Snoop protocol.

Figure 4.9 compares the throughput of a connection using the Snoop protocol with that of a connection using TCP Reno, for various exponentially-distributed bit-error rates shown on a log scale. We see that for bit-error rates of over $1x10^{-6}$ (close to the 64 KB point on the horizontal axis of the graph) the Snoop protocol performs significantly better than unmodified TCP, achieving a throughput improvement factor of 2 to 20 depending on the bit error rate. In fact, the Snoop protocol was robust and completed the run even when every other packet was being dropped over the last link, whereas the regular TCP connection did not make any significant progress. Under conditions of very low bit error rates ($< 5x10^{-7}$), there is little difference between the Snoop protocol and unmodified TCP. At such low bit errors there is typically much less than one error per transmitted window and unmodified TCP is quite robust at combating these error rates. At these error rates, the Snoop protocol behaves as is it were not present and incurs no overhead.

A more detailed picture of the behavior of the protocols can be seen can be seen in Figure 4.10, which plots the packet sequence trace for one of the experiments. The figure shows the comparison of sequence number progression in a connection using the Snoop protocol and a connection using TCP Reno for an exponentially-distributed bit-error rate of $3.9x10^{-6}$ (a bit error every 256 Kbits on average). We see that the Snoop protocol maintains a consistent level of progress, resulting in good throughput, while TCP Reno unnecessarily invokes congestion control procedures in response to

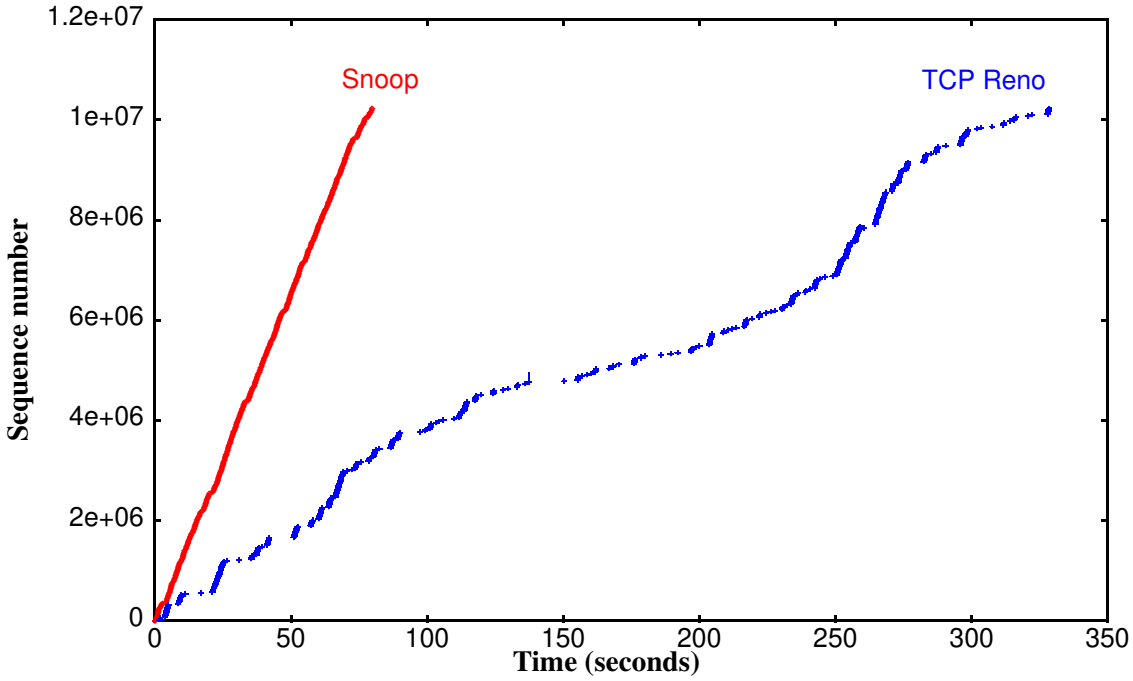

Figure 4.10 Sequence trace for transfer to MH over a channel with 3.8×10^{-6}(1/256 Kbits) BER.

bit-error losses several times during the course of the connection. This phenomenon leads to time-outs that appear as the flat and empty regions of the curve, significantly degrading the resulting throughput. For this particular transfer, the aggregate bandwidth with the Snoop protocol was about 1 Mbps, while it was only about 0.25 Mbps for TCP Reno.

We also performed several experiments to measure the performance of data transfer from the MH to the FH. These results, measured across a range of exponentially-distributed bit-error rates, are shown in Figure 4.11. As before, there are significant performance benefits of using the Snoop protocol coupled with ELN in this situation. These measurements were made for wide-area transfers between UC Berkeley and IBM Watson Research Center (NY), across one wireless WaveLAN hop and 16 Internet hops. At medium to high error rates, the performance improvement due to ELN is roughly 100%. At lower error rates, TCP Reno performs quite well as expected, and the benefits of ELN are not as pronounced. The main advantage of ELN is that it helps maintain a large TCP congestion window even when wireless error rates are high, reacting only to congestion.

While the relative performance of ELN is about 100% better than Reno at high error rates, it is not as high as the relative improvement for the FH to MH case using the Snoop protocol. The reason

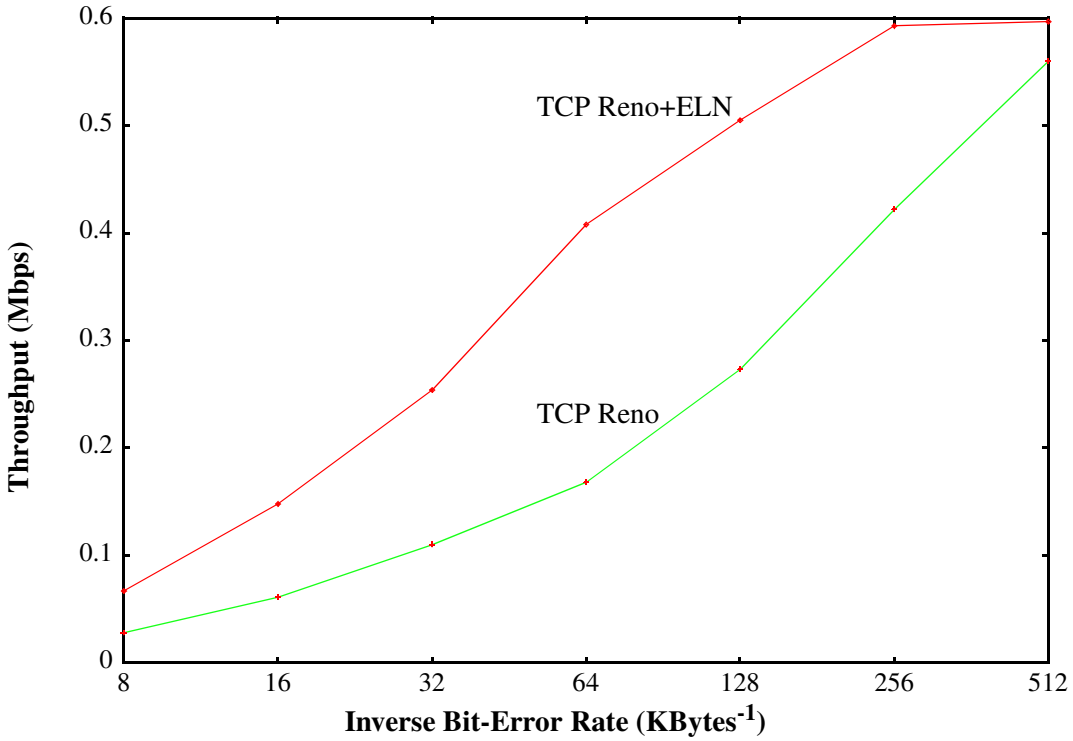

Figure 4.11 Throughput of TCP Reno and TCP Reno enhanced with ELN across a range of exponentially distributed bit-error rates, for transfers from a mobile host.

for this is that the Snoop protocol performs quick local recovery in addition to shielding the sender from congestion. On the other hand, loss recovery with ELN is due to TCP retransmissions alone. When multiple losses occur in a window TCP with ELN incurs a coarse timeout, leading to "only" a 100% improvement in throughput. A more detailed quantitative evaluation of this and enhancements to further improve performance are in Chapter 5.

4.7.2 Mobility

To isolate and measure the impact of handoffs on TCP performance, we examined the performance of a TCP connection to the mobile host with regularly spaced handoffs between two base stations. As mentioned in Section , each base station sends out a beacon signal once per second. The presence of these beacons reduces the peak TCP throughput in the absence of errors and handoffs to 1.45 Mbps. In this experiment, the arrival of a beacon at the mobile host does not trigger an analysis of the signal strengths of the different base stations; instead, the mobile host uses the time elapsed since the last handoff to determine if a handoff should occur. We performed several tests to stress the performance impact of the handoff scheme, varying the time between handoffs from 1 to

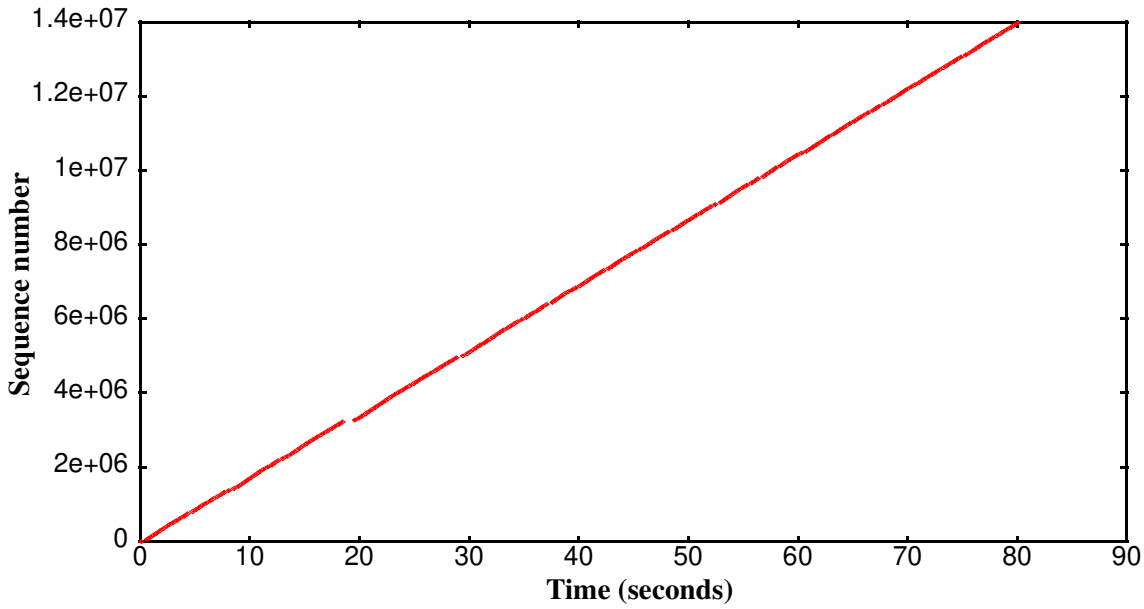

Figure 4.12 Sequence numbers for transfer to mobile host over channel with handoffs every 10 seconds.

10 seconds. The results for different handoff frequencies is shown in Table 4.1. These measure-

Time Between Handoffs (sec)	Throughput (Mbps)	Standard Deviation (Mbps)
1	1.42	.011
2	1.43	.016
3	1.43	.012
5	1.43	.014
8	1.44	.012
10	1.43	.012
Inf.	1.45	.011

Table 4.1 TCP Throughput received by the mobile host at different handoff frequencies. The last row, labeled "Inf." corresponds to the "no-handoff" case.

ments show that even frequent handoffs have very little impact on performance. In real environments, handoffs are usually likely to occur much less frequently than once per second.

The behavior of a connection experiencing handoff is shown in Figure 4.12. The figure plots the sequence numbers of a TCP connection to a mobile host with handoffs occurring every 10 sec-

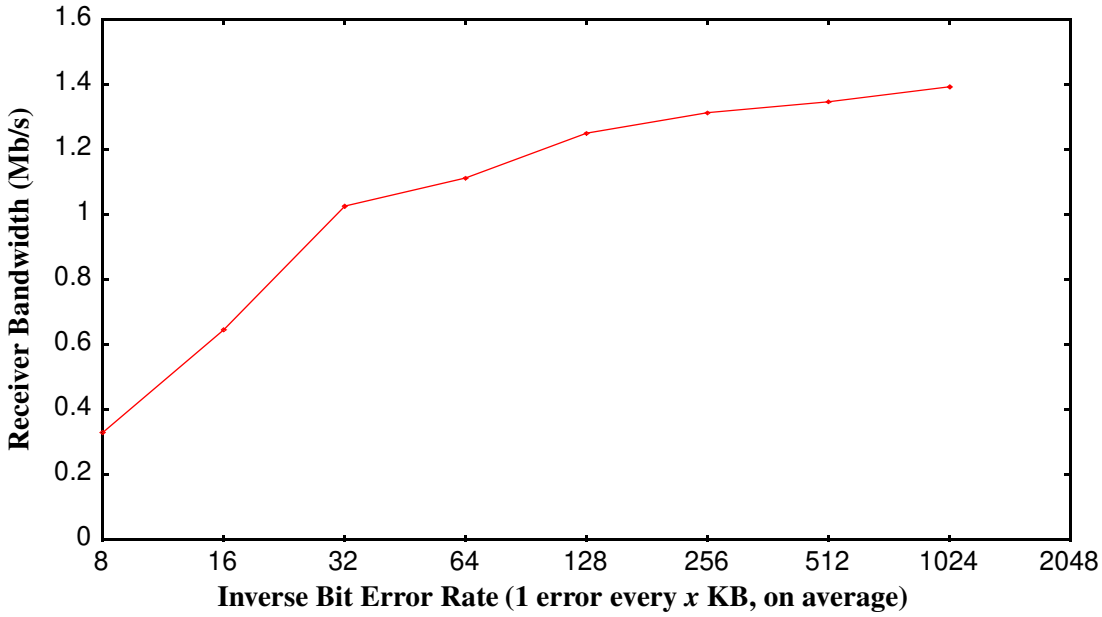

Figure 4.13 Throughput received by the mobile host at different bit-error rates ($\log_2$ scale). Handoffs occur every 5 seconds.

onds. We see that the data transfer progresses without any significant interruptions despite the presence of the handoffs. The throughput during this transfer was about 1.4 Mbps.

Figure 4.13 shows the performance obtained from the combination of the Snoop and handoff protocols. The figure plots the receiver throughput seen over a 10 Mbyte transfer. The connections experienced different error rates, shown on the x-axis, and handoffs at a fixed rate, once every 5 seconds. At low bit-error rates, $< 1\text{x}10^{-6}$ (one error every 1 Mbits of data), we obtain close to the peak performance available in the presence of beacons, 1.45 Mbps. At higher bit-error rates, we see performance comparable to transfers through Snoop to a mobile host not experiencing handoff. This indicates that the handoffs do not adversely affect the snoop processing and that the Snoop agent state is effectively mirrored at the buffering base stations.

4.8 Detailed Simulations

To thoroughly evaluate performance under different bandwidths, latencies, error rates and workloads, and to quantify the scaling behavior of the Snoop agent, we performed a number of other experiments in *ns* (Section 3.1). We now discuss these results.

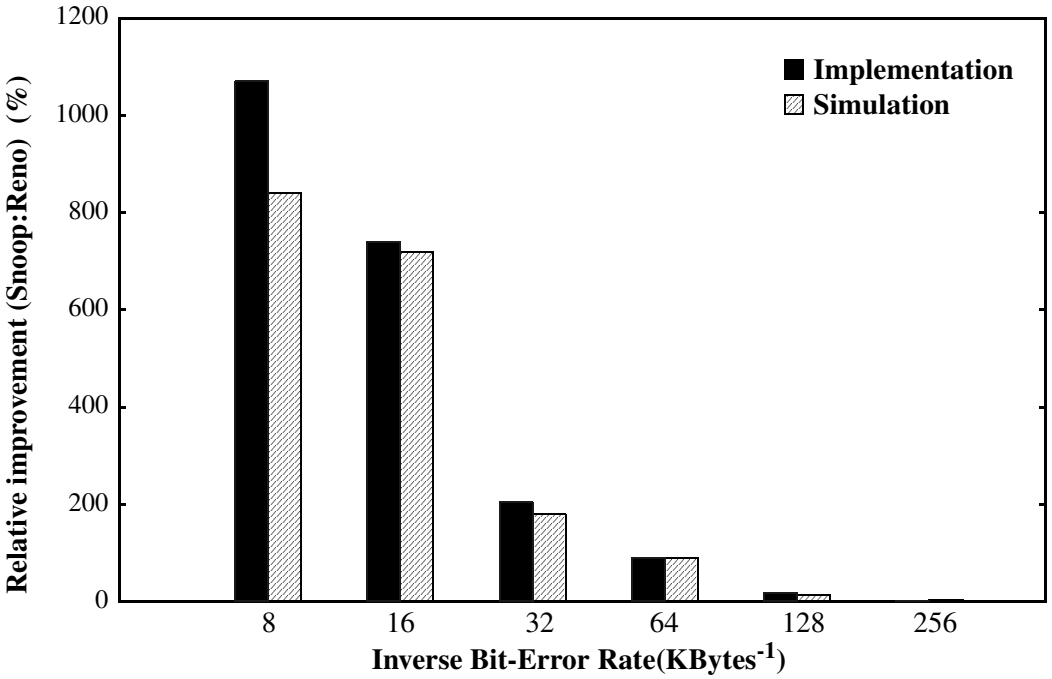

Figure 4.14 Comparison of implementation and simulation performance of the Snoop protocol relative to TCP Reno at different bit-error rates. The simulation performance closely tracks implementation measurements.

We first present some results that validate the simulation of the Snoop protocol and TCP in *ns*. Then, we present several results of bulk and Web transfers under different topologies, and a quantitative analysis of the scaling behavior of the Snoop protocol as a function of the memory consumed at the base station by the Snoop agent. In most of these experiments we use the empirical channel error model derived in Section 3.1.2.

We start by presenting a set of measurements to validate our simulation environment with real measurements. We do this by comparing the performance of a bulk TCP transfer in both environments as a function of an exponentially distributed bit-error distribution similar to the one used in the previous sections. This comparison is shown in Figure 4.14, which compares the percentage improvement of the Snoop protocol relative to TCP Reno for each error rate, in both the real implementation and the simulator. This comparison shows that the simulations track the real implementation quite closely. At high error rates, the performance of TCP Reno is especially dependent on the number and duration of timeouts that occur when multiple losses occur in a window, because losses are relatively frequent and window sizes are small. This explains why the rel-

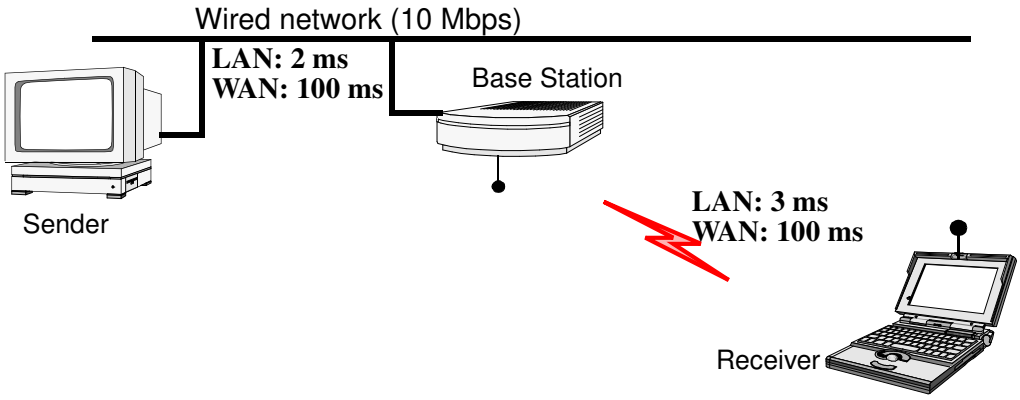

Figure 4.15 Network topologies used in simulation experiments. Wired and wireless link delays for the LAN and WAN cases are shown near the corresponding links.

ative improvement of the real implementation is larger than that in simulation in this realm—timers in the implementation are more conservative than in our simulations.

In the rest of this section, we use the empirical error models derived from traces collected in the Reinas outdoor wireless network. Section 3.2.1 describes this in more detail.

4.8.1 Bulk Transfers

We performed four kinds of experiments modeling different combinations of LAN and WAN scenarios to evaluate the Snoop protocol. These topologies are shown in Figure 4.15. We experimented with wireless link bandwidths from 128 Kbps to 4 Mbps, increasing in powers of two, in both LAN and WAN situations.

Each individual experiment, for a given combination of wired and wireless parameters, consisted of a 50-second TCP bulk transfer with 1000-byte TCP packets in the presence of a small amount of background cross-traffic. Errors on the wireless link were generated according to the empirical error model shown in Figure 3.3 and Figure 3.4. At the end of 50 seconds, we calculate the throughput achieved by the transfer; for each combination of parameters, we report the mean of 50 distinct experiments.

Figures 4.16 and 4.17 shows the results of these simulations for the wireless LAN and wireless WAN respectively. While the Snoop protocol achieves between 15% and 25% of the maximum possible link bandwidth even in the presence of significant error rates, both TCP Reno and TCP SACK achieve only between 3% and 10% of the maximum link speeds. Furthermore, the benefits

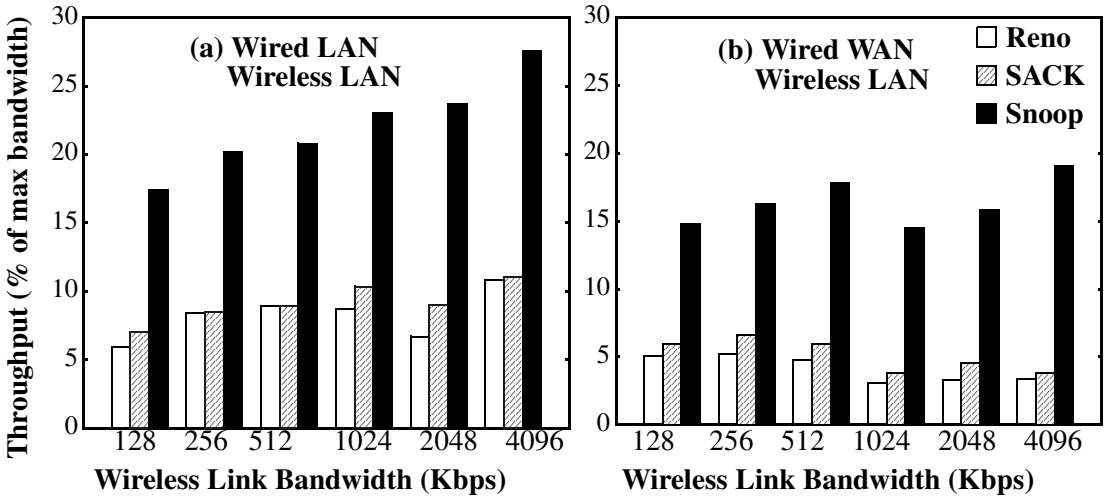

Figure 4.16 Simulated performance of TCP Reno, TCP SACK and the Snoop protocol as a function of link bandwidth of the wireless LAN. (a) shows the results of the TCP sender on the same LAN as the BS, while in (b), the sender is 100ms away from the BS.

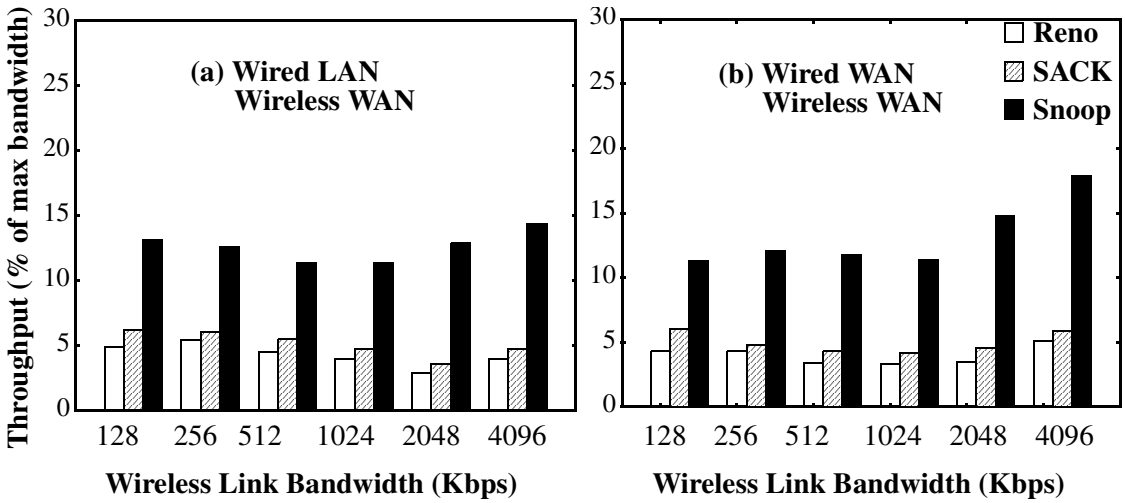

Figure 4.17 Simulated performance of TCP Reno, TCP SACK and the Snoop protocol as a function of link bandwidth of the wireless LAN. (a) shows the results of the TCP sender on the same LAN as the BS, while in (b), the sender is 100ms away from the BS.

of the Snoop protocol relative to TCP are more pronounced in the (wired or wireless) WAN case. In this case, the bandwidth-delay product of the connection is larger, making it more important for the sender to maintain as large a congestion window as possible. While the predominant benefits of the Snoop protocol in the wireless LAN case are due to its local recovery mechanisms and prevention of sender timeouts, the fact that TCP and TCP SACK frequently reduce their congestion window is not particularly problematic because of the inherently small bandwidth-delay product in this case. But this causes significant problems and results in worse performance in the WAN case.

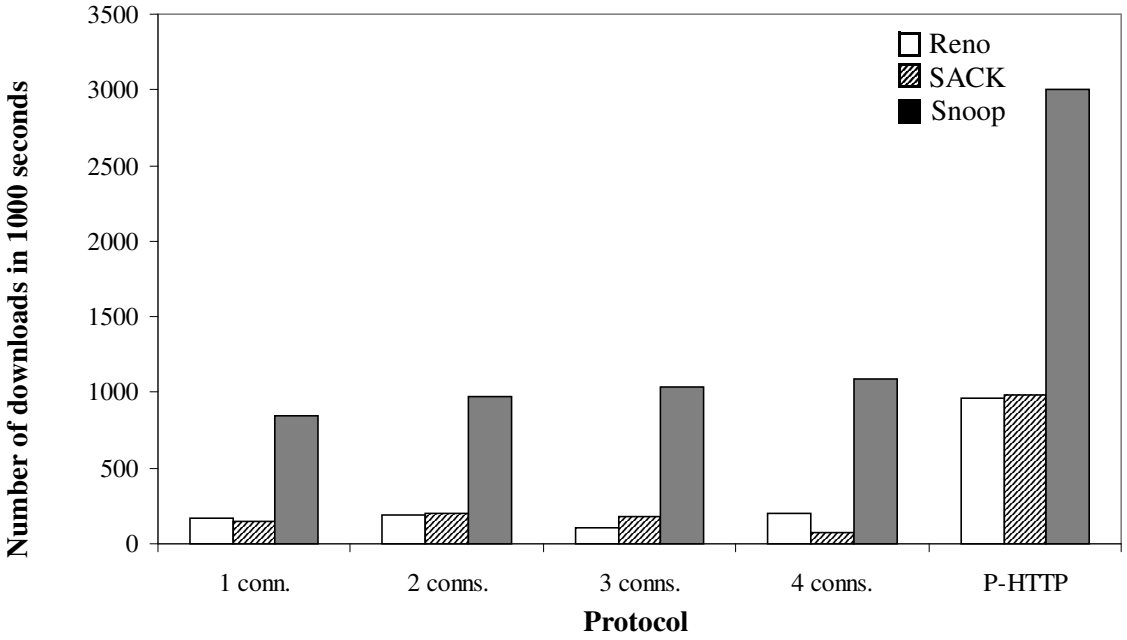

Figure 4.18 Simulated performance of the TCP Reno, TCP SACK, and the Snoop protocol on a Web workload in different protocol configurations. The graphs show the number of separate downloads (connections) using 1 to 4 concurrent connections per client, as well as the performance of persistent-HTTP [44] for the three protocols.

The same observations explain the better relative performance of the Snoop protocol at higher wireless link bandwidths.

4.8.2 Web Transfers

In the previous sections, we saw that the Snoop protocol significantly improved over TCP Reno and TCP SACK across a wide range of error rates and network topologies, for bulk transfer workloads. To investigate performance on shorter transfers, we performed a set of experiments with a Web-like access workload. We use Mah's empirical Web workload mentioned in Section 3.3 [88].

We conducted experiments using TCP Reno, TCP SACK and the Snoop protocol using the Web workload, varying the number of concurrent TCP connections from 1 to 4, as well as using persistent-HTTP [44]. The results of these experiments are shown in Figure 4.18, which depicts the number of successfully completed individual downloads (connections) in 1000 seconds. We see that the performance of the Snoop protocol is between three and six times better than the other protocols. We also note that the P-HTTP performance does not account for user think times between

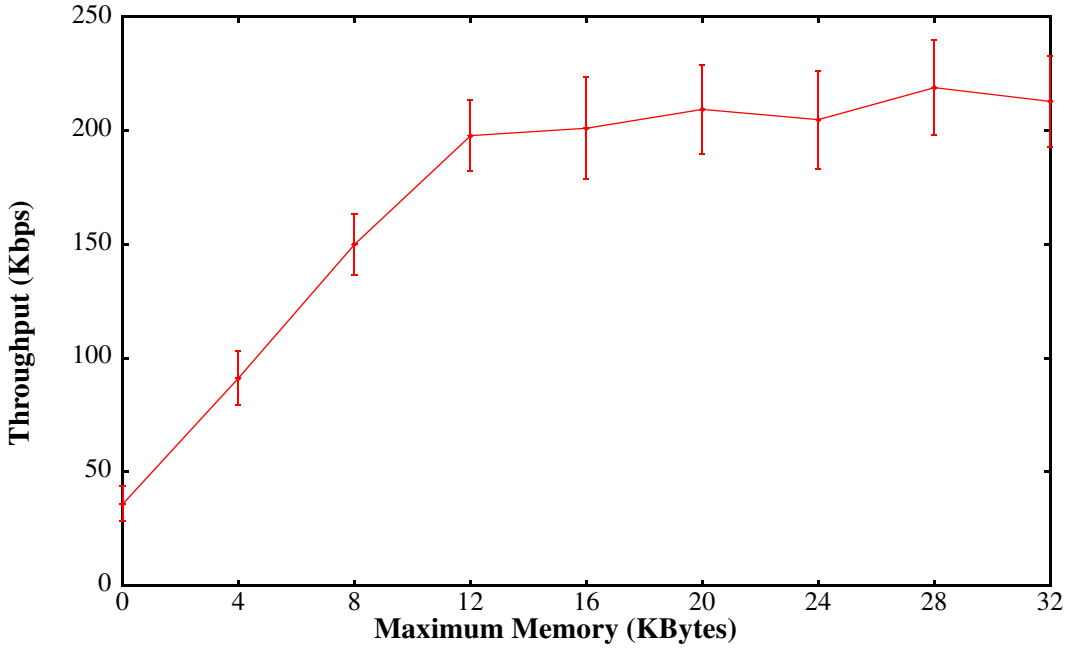

Figure 4.19 TCP connection throughput as a function of the maximum memory allocated to the snoop cache. The maximum window size of the connection is 32 KBytes. This graph shows the worst-case degradation because the bandwidth-delay product is large (50 KB).

successive downloads, and is hence an optimistic upper bound on the performance achievable in practice.

These results show that not only does the Snoop protocol perform well for large bulk transfers, but that it also results in significant performance improvements for short transfers (combined with occasional long ones) that characterize Web workloads today. The performance improvement is between a factor of three and six for this empirical Web workload under realistic wireless error conditions.

4.8.3 Scaling Behavior

In this section, we discuss the results of experiments to quantify the scaling behavior of the Snoop protocol as the number of connections grows. There are two potential sources of overhead associated with the Snoop protocol at the base station: processing overhead and memory consumption. We first show that the processing overhead is negligible and then evaluate the memory consumed by the snoop agent.

The per-packet processing required at the BS when no packet losses occur is not significantly more than without the Snoop protocol for transfers in either direction. For each incoming packet, the protocol performs a lookup in a hash table that maintains all active TCP connections keyed by source and destination address/port tuples and extract a pointer to the connection state. Then, depending on the direction, the packet is either added to a cache (without any extra memory copies) and forwarded on, or else forwarded on after a check is made for any holes in the transmission sequence. In both cases, this takes a negligible number of instructions compared to the (background) cost of routing and forwarding packets. Most of the protocol processing complexity occurs when losses occur and duplicate ACKs arrive, but our measurements show this to be a negligible component of the forwarding code time.

We now consider the question of memory consumption at the base station. For transfers originating from a mobile host, the amount of memory consumed is dependent on the number of holes in the transmission sequence and is largely independent of the transmission window size. It is also independent of the packet size, since no caching occurs in this case. For transfers to a mobile host, memory consumption is dominated by the per-packet storage cost, which we quantify below.

As the number of connections increases, the state maintained by the snoop agent increases roughly linearly. This state has two components—a small amount of control state and state required to store unacknowledged packets. The maximum amount of cache memory consumed by the snoop agent per connection is bounded by the maximum transmission window size of the connection. In our implementation, this space is not statically allocated, but is dynamic, reflecting the actual window size changes in the connection.

Thus, the key scaling question is: as the total amount of cache memory in the base station varies, how does the number of sustainable connections vary, at a given error rate? Here, the term “sustainable” is used to imply that all connections must have a noticeable performance improvement by having the Snoop protocol run at the base station, since we can always sustain as many connections as we want, limited only by the wireless link bandwidth, with the snoop agents providing no performance benefit.

We answer this question by varying the amount of per-connection cache memory at the base station, and by measuring the performance of the Snoop protocol for bulk TCP transfers as a function

of the amount of cache memory. The results of this experiment, performed on 1KB sized packets, is shown in Figure 4.19. We see that the performance linearly increases with cache memory until it plateaus off at the maximum value when the cache size reaches a certain threshold value. This value is equal to the "effective" window size for the connection. We note that this curve highlights the worst-case performance degradation because the bandwidth-delay product of the connection is large (50 KB).

This linear performance degradation when cache memory is small is expected, as the following analysis shows. Clearly, the existence of the cache does not influence packets that were successfully received by the receiver. For any lost packet needing retransmission, let p be the probability of it existing in the Snoop cache. If W is the effective window size for the connection and m the available cache memory, then $p = m/W$. Now, for each lost packet in the cache, the corresponding connection throughput is B_h (cache hit contribution). Similarly, the throughput for a cache miss is $B_m < B_h$. Thus, the effective connection throughput is $pB_h + (1-p)B_m$. Since $p = m/W$, the performance is linear in m, and is constant for $m >= W$.

For the error rates observed in our experiments, performance was close to the best possible for snoop cache sizes over 12 KBytes per connection, corresponding to the effective window size. Given current technology trends and the inexpensiveness of memory, we do not expect base station memory to be a practical limitation to schemes like the Snoop protocol, especially considering the performance benefits obtained. A nice property of the soft state in the Snoop protocol is that the performance degradation is linear in the amount of memory used.

4.9 Summary

In this chapter, we presented the design and implementation of the Snoop protocol, a protocol that significantly improves the performance of TCP in single-hop cellular wireless networks, providing performance improvements up to 20 times, across a wide range of error rates.

For data transfer *to* a mobile host, the Snoop protocol improves TCP performance by deploying an agent at the base station, called the Snoop agent. This agent aids in the quick, local recovery of data loss by taking advantage of the information conveyed in TCP ACKs from the receiver. It retransmits packets locally based on duplicate ACKs and local timeouts, and suppresses certain

duplicate ACKs from the sender to shield wireless losses from it. The internal state it maintains is soft, in the sense that it is not absolutely necessary for the correct end-to-end functioning of the protocol, and is very easy to regenerate and is periodically refreshed. For data transfer *from* a mobile host, the protocol uses the *Explicit Loss Notification* mechanism. These notifications are generated by the base station and sent to the mobile sender as part of the ACK stream, which reacts to them in an appropriate manner. While the Snoop protocol in the first case (transfer to a mobile host) is an example of a *transport-aware link protocol*, in the second case (transfer from a mobile host) it enhances the transport protocol to make it *link-aware*. We have also integrated the Snoop protocol with a multicast-based low-latency handoff protocol, achieving significant improvements in performance in the face of both wireless bit-errors and user mobility between cells.

In the next chapter, we present a detailed performance evaluation of the Snoop protocol and quantitatively compare it to many other alternatives. We discuss the relative benefits of several techniques used in this and other protocols, and argue that exploiting cross-layer protocol optimizations as the Snoop protocol does, is key to achieving good performance in error-prone wireless environments. We also discuss some other enhancements to the Snoop protocol in that chapter to handle wireless transit links and exploit selective acknowledgments for better loss recovery when losses occur in bursts. Finally, we discuss some key lessons we have learned from our experience with reliable data transport over error-prone wireless networks.

Everything's got a moral, if only you can find it.

— Alice in Wonderland, Chapter 9, Lewis Carroll

Chapter 5

Cross-Layer Protocol Optimizations

In this chapter, we compare the performance of the Snoop protocol to conventional link-layer protocols and to split-connection approaches. Our goal is to understand the precise reasons for performance improvement in each scheme. In the process, we demonstrate the performance benefits of cross-layer protocol optimizations and transport-awareness in the Snoop protocol. We also show that the split-connection architecture is not the reason for good performance in certain split approaches.

We then discuss enhancements to the Snoop protocol to improve performance in wireless transit networks using the ELN mechanism introduced in the previous chapter. We also describe how the Snoop protocol can be enhanced using selective acknowledgment (SACK) information from the receiver to perform better recovery from burst losses. We conclude this chapter with a discussion of some key lessons learned from our experience with improving reliable transport over error-prone wireless networks.

5.1 Introduction

There are two different approaches to improving TCP performance in error-prone wireless networks. The first approach hides any non-congestion-related losses from the TCP sender and there-

fore requires no changes to existing sender implementations. Protocols that adopt this approach attempt to make the error-prone link appear as a higher quality link with a reduced effective bandwidth. As a result, most of the losses seen by the TCP sender are caused by congestion. Examples of this approach include wireless links with reliable link-layer protocols (Section 2.5.2), split connection approaches (Section 2.5.3), and TCP-aware link-layer schemes such as the Snoop protocol (Chapter 4). The second approach attempts to make the sender aware of the existence of wireless hops and realize that some packet losses are not due to congestion. The sender can then avoid invoking congestion control algorithms when non-congestion-related losses occur. Examples of this approach include the Explicit Loss Notification (ELN) scheme described in Chapter 4 and approaches that inform the TCP sender of on-going handoffs [90]. Of course, it is possible for a wireless-aware transport protocol to coexist with link-layer schemes.

Each approach admits a number of different protocols. We classify the many protocols into three basic groups, based on their fundamental philosophy: *link-layer protocols, end-to-end protocols* and *split-connection protocols.*

We investigate ARQ-based link-layer protocols whose local retransmissions are tuned to the characteristics of the wireless link to provide significant improvements in performance. However, since the end-to-end TCP connection passes through the error-prone link, the TCP sender may not be fully shielded from wireless losses. Furthermore, if the link-layer protocol operates independently of the end-to-end transport, adverse interactions could occur between the two that degrade performance, as described in Section 2.5.2.

The end-to-end protocols we investigate attempt to make the TCP sender handle losses through the use of two techniques. First, they use some form of SACK to allow the sender to recover from multiple packet losses in a window without incurring a coarse timeout. Second, they attempt to have the sender distinguish between congestion and other forms of losses using the ELN mechanism.

Finally, split-connection protocols completely hide the wireless link from the sender by terminating the TCP connection at the base station. Such schemes use a separate reliable connection between the base station and the destination host. The second connection can use techniques optimized for wireless such as negative or selective ACKs, rather than just TCP Reno.

In this chapter, we evaluate the performance of several link-layer, end-to-end and split-connection protocols using end-to-end throughput and goodput as performance metrics, in both LAN and WAN configurations. In particular, we seek to answer the following specific questions:

1. What combination of mechanisms results in best performance for each class of protocols?
2. How important is it for link-layer schemes to be aware of TCP algorithms to achieve good end-to-end throughput? What is the quantitative benefit of TCP-awareness in the Snoop protocol?
3. How useful are selective ACKs in dealing with error-prone links, especially when losses occur in bursts?
4. Is splitting the end-to-end connection fundamentally important to achieving good wireless TCP performance?

We answer these questions by implementing and evaluating the various protocols in a WaveLAN-based cellular wireless testbed. For each protocol, we measure the end-to-end throughput and goodputs for the wired and wireless paths. For any path (or link), goodput is defined as the ratio of the actual transfer size to the total number of bytes transmitted over that path (Section 3.3). In general, the wired and wireless goodputs differ because of wireless losses and local retransmissions over the wireless link. These metrics allow us to determine the end-to-end performance as well as the transmission efficiency across the network.

We reinforce the results of Chapter 4, showing that transport-aware link protocols like the Snoop protocol lead to very good performance. Our experiments indicate that shielding the TCP sender from duplicate ACKs caused by wireless losses and being aware of TCP messaging improves throughput by at least 10-30% compared to a conventional reliable link-layer protocol in both LAN and WAN topologies. We show that split connections are not fundamental to achieving good performance. Split-connection approaches can be tuned to perform well, but their good performance is not due to the split *architecture*. Rather, their good performance derives from specialized techniques over the wireless hop, and the same techniques can be used without splitting the end-to-end connection. We also demonstrate that a simple SACK scheme and ELN lead to significant performance improvements. In particular, our ELN scheme improves end-to-end throughput by more than 100% compared to TCP Reno, with comparable goodput values.

The rest of this chapter is organized as follows. Section 5.2 presents a taxonomy of the protocols we study. Section 5.3 describes the implementation details of the different protocols in our wireless testbed. Section 5.4 presents the results and analysis of several experiments and discusses the reasons for observed performance in each case. In Section 5.5, we present enhancements to the Snoop protocol to handle wireless transit links and burst losses. We discuss some key lessons learned about protocol design for error-prone networks in Section 5.6 and conclude with a summary in Section 5.7.

5.2 Taxonomy

Table 5.1 summarizes the key ideas in each protocol and the main differences between them. Each protocol reacts to these losses and recovers from them in different ways.

Name	Category	Special Mechanisms
E2E	end-to-end	Standard TCP-Reno
E2E-NEWRENO	end-to-end	TCP Newreno
E2E-SMART	end-to-end	SMART-based selective ACKs
E2E-IETF-SACK	end-to-end	IETF selective ACKs
E2E-ELN	end-to-end	Explicit Loss Notification (ELN)
E2E-ELN-RXMT	end-to-end	ELN with retransmit on first dup ACK
LL	link-layer	Local retransmissions
LL-TCP-AWARE	link-layer	Local retransmissions + dup ACK suppression
LL-SMART	link-layer	SMART-based selective ACKs
LL-SMART-TCP-AWARE	link-layer	SMART and dup ACK suppression
SPLIT	split-connection	none
SPLIT-SMART	split-connection	SMART-based wireless connection

Table 5.1 Summary of protocols investigated in this chapter.

5.2.1 Link-Layer Schemes

Unlike TCP for the transport layer, there is no de facto standard for reliable link-layer protocols. Our base link-layer protocol, called LL, uses cumulative ACKs to detect lost packets and then retransmit them from the base station to the mobile host. To minimize overhead, our implementation of LL uses TCP ACKs instead of generating its own. Timer-driven retransmissions are done

based on a smoothed round-trip time estimate, with a minimum timeout granularity of 200 ms to limit the overhead of processing timer events. This permits LL to retransmit packets several times before a typical TCP Reno sender times out. LL is logically equivalent to the Snoop agent that does not suppress any duplicate ACKs and does not attempt in-order delivery of packets across the link when packets are lost.

While the use of TCP ACKs by our LL protocol renders it atypical of traditional ARQ protocols, it still preserves the key feature of such protocols: the ability to retransmit packets locally, independently of, and on a faster time scale than TCP. In practice, we find that LL is effective in performing quick local retransmissions of lost segments and preventing timeouts at the sender. Therefore, we expect the qualitative aspects of our results to be applicable to general ARQ-based link-layer protocols.

We also investigate a more sophisticated link-layer protocol (LL-SMART) that uses selective retransmissions to improve performance. The LL-SMART protocol uses a SMART-based selective ACK scheme (Section 2.3.1) at the link layer. Like the LL protocol, LL-SMART uses TCP ACKs instead of generating its own and limits its minimum timeout to 200 ms. LL-SMART is equivalent to the Snoop agent performing retransmissions based on SACKs but not suppressing duplicate ACKs at the base station.

We added TCP awareness to both the LL and LL-SMART protocols, resulting in the LL-TCP-AWARE and LL-SMART-TCP-AWARE schemes. The LL-TCP-AWARE protocol is identical to the Snoop protocol, while the LL-SMART-TCP-AWARE protocol uses SMART-based techniques for further optimization using selective retransmissions. LL-SMART-TCP-AWARE is the best link-layer protocol in our experiments (and in fact the best-performing protocol amongst those we considered): it performs local retransmissions based on selective ACKs and shields the sender from duplicate ACKs caused by wireless losses.

5.2.2 End-To-End Schemes

Although a wide variety of TCP variants are used in the Internet, TCP Reno is the current de facto standard for TCP implementations (Section 2.2). We call this the E2E protocol and use it as the standard basis for performance comparison.

The E2E-NEWRENO [55] protocol improves the performance of TCP-Reno after multiple packet losses in a window by remaining in fast recovery mode if the first new ACK received after a fast retransmission is "partial," i.e, is less than the value of the last byte transmitted when the fast retransmission was done. This protocol is described in Section 2.3.2.

The E2E-SMART and E2E-IETF-SACK protocols add SMART-based and IETF SACKs (Section 2.3.1) respectively to TCP Reno. This allows the sender to handle multiple losses within a window of outstanding data more efficiently than TCP Reno. However, the sender still assumes that losses are a result of congestion and invokes congestion control procedures, shrinking its congestion window size. This allows us to identify what fraction of the end-to-end performance degradation is associated with standard TCP's error detection and retransmission.

In SMART [76], the receiver sends information about which data segment caused the duplicate cumulative ACK to be generated. The sender uses this information to construct a map of which segments in the window are missing and perform the appropriate retransmissions. We used the SMART-based scheme only for the LAN experiments. This scheme is well-suited to situations where there is little reordering of packets, which is true for one-hop wireless systems such as ours. Unlike the scheme proposed in [76], we do not use any special techniques to detect the loss of a retransmission. The sender retransmits a packet when it receives a SMART ACK only if the same packet was not retransmitted within the last round-trip time. If no further SMART ACKs arrive, the sender reverts to the coarse timeout mechanism to recover from the loss. We experimented with the IETF SACK scheme in both LAN and WAN settings. Our implementation is based on the RFC and takes appropriate congestion control actions upon receiving SACK information [11].

The E2E-ELN protocol adds the Explicit Loss Notification (ELN) option to TCP ACKs. When a packet is dropped on the wireless link, future cumulative ACKs corresponding to the lost packet are marked to identify that a non-congestion related loss has occurred. Upon receiving this information with duplicate ACKs, the sender retransmits data without invoking the associated congestion control procedures. This protocol allows us to identify what fraction of the end-to-end performance degradation is associated with TCP's incorrect invocation of congestion control algorithms when it performs a fast retransmission of a packet lost on the wireless hop. The E2E-ELN-RXMT protocol is an enhancement of E2E-ELN, where the sender retransmits the packet on receiving the *first* duplicate ACK with the ELN option set (as opposed to the third duplicate ACK

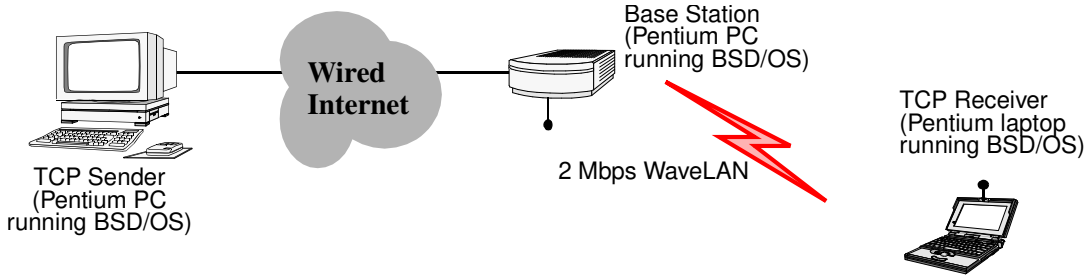

Figure 5.1 Experimental topology. A 10 Mbps Ethernet connected the sender to the base station in the LAN experiments, while there were 16 wired Internet hops between them in the WAN experiments.

in the case of TCP Reno), in addition to not shrinking its window size in response to wireless losses. At the TCP sender, this is similar to the protocol described in Section 4.4.2.

5.2.3 Split-Connection Schemes

We experiment with two split-connection schemes: SPLIT and SPLIT-SMART. The SPLIT scheme uses the base station to divide an end-to-end TCP connection into two separate TCP connections (Section 2.5.3). Our efficient implementation avoids data copying at the base station by passing the pointers to the same buffer between the two TCP connections. The SPLIT-SMART scheme is optimized for good performance over the wireless hop. It uses a SMART-based SACK scheme on the wireless connection to perform selective retransmissions. There is little chance of reordering of packets over the wireless connection since the base station is only one hop away from the final destination; thus all duplicate ACKs signify losses rather than reordering.

5.3 Experimental Methodology

We performed several experiments to compare the performance and efficiency of each of the protocols. The protocols were implemented as a set of modifications to the BSD/OS TCP/IP (Reno) network stack. To ensure a fair basis for comparison, none of the protocols implementations introduce any additional data copying at intermediate points from sender to receiver.

The network topology for our experiments is shown in Figure 5.1. The peak throughput for TCP bulk transfers is 1.5 Mbps in the local area testbed and 1.35 Mbps in the wide area testbed in the absence of congestion or wireless losses. These testbed topologies represent typical scenarios for

mobile hosts connected over cellular wireless networks. We focus on data transfer to the mobile host, which is the common case for mobile applications (e.g., Web accesses).

To measure the performance of the protocols under controlled conditions, we emulate errors on the wireless link using an exponentially distributed bit-error model. The receiving entity on the error-prone link generates an exponential distribution for each bit-error rate and changes the IP header checksum of the packet if the error generator determines that the packet should be dropped. Losses are generated in both directions of the wireless channel, so TCP ACKs are dropped too. The TCP data packet size in our experiments is 1400 bytes. We first measure and analyze the performance of the various protocols at an average error rate of one every 64 KB (this corresponds to a base bit-error rate of about 1.9×10^{-6}). Because the exponential distribution has a standard deviation equal to its mean, there are several occasions when multiple packets are lost in close succession in the same window. We then investigate the performance of many of these protocols across a range of error rates from one every 16 KB to one every 256 KB.

The choice of the exponentially distributed error model is motivated by our desire to understand the precise dynamics of each protocol in response to a wireless loss, and is not an attempt to empirically model a wireless channel. While the actual performance numbers will be a function of the exact error model, the relative performance is dependent on how the protocol behaves after one or more losses in a single TCP window. Thus, we expect our overall conclusions to be applicable under other patterns of wireless loss as well. Finally, we believe that though wireless errors are generated artificially in our experiments, the use of a real testbed is valuable because it introduces realistic effects such as wireless bandwidth limitation, media access contention, protocol processing delays, etc., that are hard to model realistically in simulation.

In our experiments, we ensure that losses are only due to wireless bit-errors and not due to congestion. This allows us to focus on the effectiveness of the mechanisms in handling such losses. In particular, the WAN experiments are performed across 16 Internet hops with minimal wired congestion[1] in order to study the impact of large delay-bandwidth products on performance.

1.WAN experiments across the United States were performed between 10 pm and 4 am (PST) and we verified that no congestion losses occurred in the runs reported.

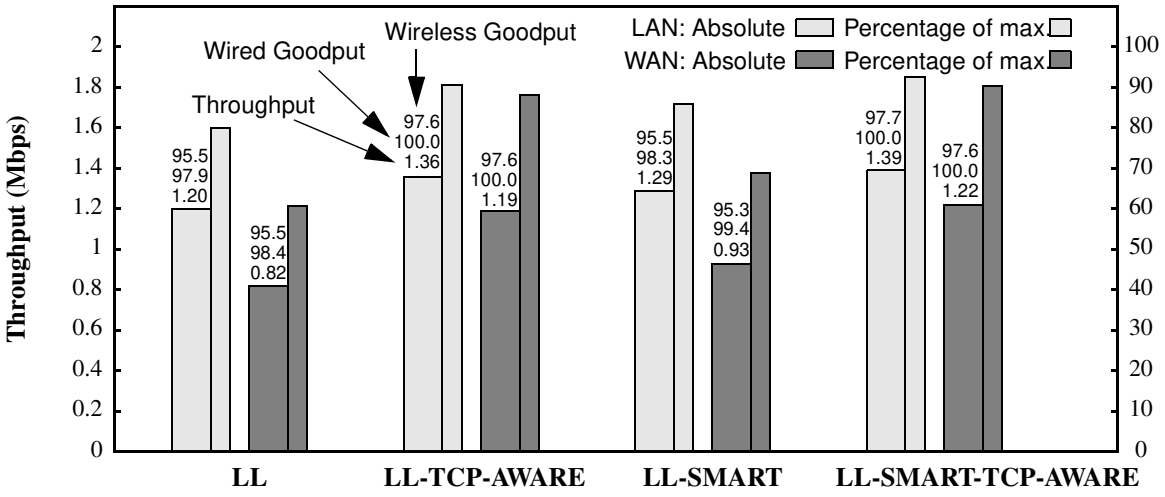

Figure 5.2 Performance of link-layer protocols: bit-error rate = 1.9×10^{-6} (1 error/64 KB), socket buffer size = 32 KB. For each case there are two bars: the thick one corresponds to the scale on the left and denotes the throughput in Mbps; the thin one corresponds to the scale on the right and shows the throughput as a percentage of the maximum, i.e. in the absence of wireless errors (1.5 Mbps in the LAN environment and 1.35 Mbps in the WAN environment).

Each run in the experiment consists of an 8 MByte transfer from the source to receiver across the wired net and the WaveLAN™ link. We chose this long transfer size to limit the impact of transient behavior at the start of a TCP connection. During each run, we measure the throughput at the receiver in Mbps and the wired and wireless goodputs as percentages. In addition, all packet transmissions on the wired network and WaveLAN are recorded for analysis using `tcpdump` [94], and the sender's TCP code instrumented to record events such as coarse timeouts, retransmission times, duplicate ACK arrivals, congestion window size changes, round-trip estimates, etc.

5.4 Results

In this section, we describe the results of our experiments. We focus on understanding and explaining the key reasons for measured performance in each case.

5.4.1 Link-Layer Protocols

Traditional link-layer protocols operate independently of the higher-layer protocol and consequently do not necessarily shield the sender from the error-prone link. In spite of local retransmis-

sions, TCP performance is often sub-optimal due to adverse interactions between independently designed link and transport layers. Section 2.5.2 describes these interactions.

Unlike the analysis in [38], timer interactions are not the dominant effect when link-layer schemes such as LL are used with TCP Reno and its variants. Most TCP implementations have coarse retransmission timeout granularities that are typically multiples of 500 ms, while link-layer protocols typically have much finer timeout granularities. The real problem is that when packets are lost, link-layer protocols that do not attempt in-order delivery across the link cause packets to reach the TCP receiver out-of-order. In response, the TCP receiver generates duplicate ACKs, which causes the sender to invoke fast retransmission and recovery. The result is degraded throughput and goodput, especially if the connection's delay-bandwidth product is larger than a few segments.

	LL	**LL-TCP-AWARE**	**LL-SMART**	**LL-SMART-TCP-AWARE**
LAN (8 KB)	1.20 (95.6%,97.9%)	1.29 (97.6%,100%)	1.29 (96.1%,98.9%)	1.37 (97.6%,100%)
LAN (32 KB)	1.20 (95.5%,97.9%)	1.36 (97.6%,100%)	1.29 (95.5%,98.3%)	1.39 (97.7%,100%)
WAN (32 KB)	0.82 (95.5%,98.4%)	1.19 (97.6%,100%)	0.93 (95.3%,99.4%)	1.22 (97.6%,100%)

Table 5.2 This table summarizes the results for the link-layer schemes for an average error rate of one every 64 KB of data. Each entry is of the form: throughput (wireless goodput,wired goodput). Throughput is expressed in Mbps.

Our results substantiate this claim, as can be seen by comparing the LL and LL-TCP-AWARE results in Figure 5.2 and Table 5.2. For a packet size of 1400 bytes, a bit error rate of 1.9×10^{-6} (1/64 KB) translates to a packet error rate of between 2.2% and 2.3%. Therefore, an optimal link-layer protocol that recovers from errors locally and does not compete with TCP retransmissions should have a wireless goodput of 97.7% and a wired goodput of 100% in the absence of congestion. In the LAN experiments, the throughput difference between LL and LL-TCP-AWARE is about 10%. However, the LL wireless goodput is only 95.5%, significantly less than LL-TCP-AWARE's wireless goodput of 97.6%, which is close to the maximum achievable goodput. When a loss occurs, the LL protocol performs a local retransmission relatively quickly. However, enough packets are typically in transit to create more than three duplicate ACKs. These duplicates eventu-

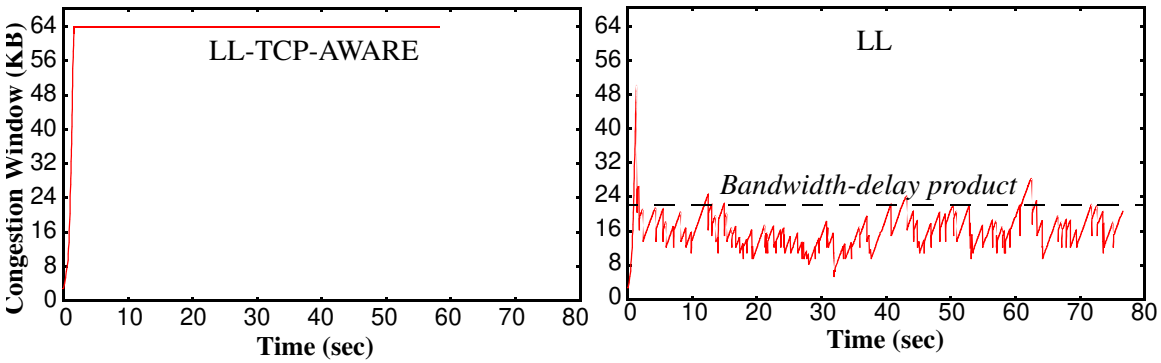

Figure 5.3 Congestion window size for link-layer protocols in the WAN experiments. The horizontal dashed line in the LL graph shows the 23 KB WAN bandwidth-delay product.

ally propagate to the sender and trigger a fast retransmission and the associated congestion control mechanisms. These redundant fast retransmissions degrade goodput; about 90% of the lost packets are retransmitted by both the TCP sender and the base station.

The effects of this interaction are much more pronounced in the wide-area experiments: the end-to-end difference in throughput between LL and LL-TCPAWARE is now about 30%. This pronounced deterioration compared to the LAN case is due to the higher bandwidth-delay product of the wide-area connection. The LL scheme often causes the sender to invoke congestion control due to duplicate ACKs and causes the average window size of the transmitter to be lower than for LL-TCP-AWARE. This is shown in Figure 5.3, which compares the congestion window size of LL and LL-TCP-AWARE as a function of time. Note that the number of outstanding data bytes in the network is the minimum of the congestion window and the receiver advertised window, bounded by the receiver's socket buffer size. In the congestion window graphs for each protocol, the receiver socket buffer is 32KB.

In the wide area, the bandwidth-delay product is about 23 KB (1.35 Mbps * 135 ms), and the congestion window drops below this value several times during each TCP transfer, and its time-averaged mean value is smaller than 23 KB. On the other hand, the LAN experiments do not suffer from such a large throughput degradation because LL's lower congestion-window size is usually still larger than the connection's delay-bandwidth product of about 1900 bytes (1.5 Mbps * 10 ms). Therefore, the LL scheme can maintain a nearly full "data pipe" between the sender and receiver in the local connection but not in the wide area one. The 10% LAN degradation is due to the excessive retransmissions over the wireless link and to the smaller average congestion window size

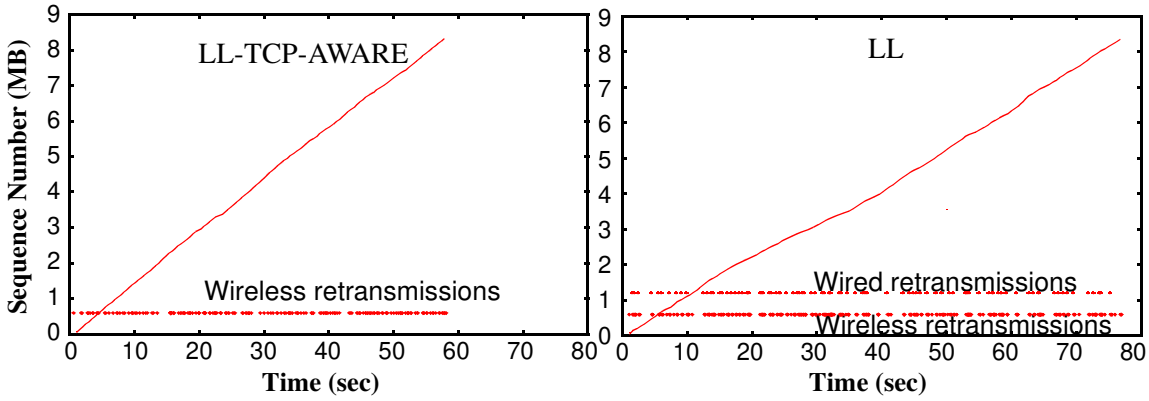

Figure 5.4 Packet sequence traces for LL-TCP-AWARE and LL. No coarse timeouts occur in either case. For LL-TCP-AWARE, the horizontal row of dots shows the times of wireless link retransmissions. For LL, the top row shows sender fast retransmissions and the bottom row shows both local wireless and sender retransmissions.

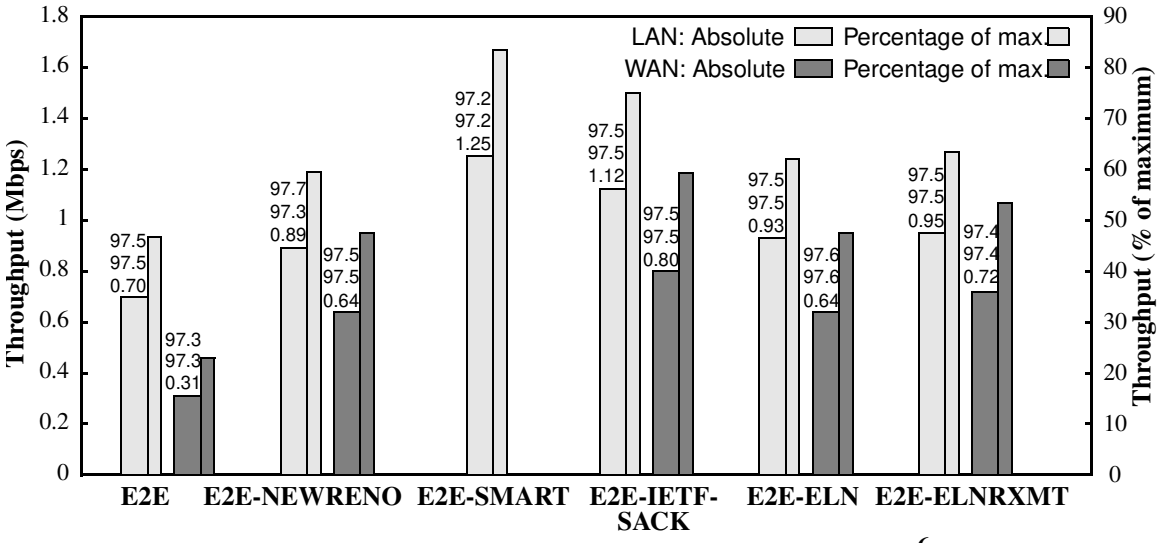

Figure 5.5 Performance of end-to-end protocols: bit error rate = 1.9×10^{-6} (1 error/64KB).

compared to LL-TCP-AWARE. Another important point to note is that LL achieves good local loss recovery and prevents coarse timeouts from occurring at the sender. Figure 5.4 shows the sequence traces of TCP transfers for LL-TCP-AWARE and LL; note that LL-TCP-AWARE has no wired retransmissions.

In summary, our results indicate that link-layer retransmission protocols that are designed independently of the transport layer are prone to adverse interactions with higher-layer mechanisms such as TCP fast retransmissions. These interactions degrade end-to-end performance. A transport-aware link-layer scheme that uses knowledge of TCP semantics to suppress duplicate ACKs

caused by wireless losses and locally retransmits packets achieves significantly better performance. This combination of local loss recovery based on TCP ACK monitoring and duplicate ACK suppression is the main reason for the good performance of the Snoop protocol.

5.4.2 End-To-End Protocols

The performance of the various end-to-end protocols is summarized in Figure 5.5 and Table 5.3.

	E2E	**E2E-NEWRENO**	**E2E-SMART**	**E2E-IETF-SACK**	**E2E-ELN**	**E2E-ELN-RXMT**
LAN (8 KB)	0.55 (97.0,96.0)	0.66 (97.3,97.3)	1.12 (97.6,97.6)	0.68 (97.3,97.3)	0.69 (97.3,97.2)	0.86 (97.4,97.3)
LAN (32 KB)	0.70 (97.5,97.5)	0.89 (97.7,97.3)	1.25 (97.2,97.2)	1.12 (97.5,97.5)	0.93 (97.5,97.5)	0.95 (97.5,97.5)
WAN (32 KB)	0.31 (97.3,97.3)	0.64 (97.5,97.5)	N.A.	0.80 (97.5,97.5)	0.64 (97.6,97.6)	0.72 (97.4,97.4)

Table 5.3 This table summarizes the results for the end-to-end schemes for an average error rate of one every 64 KB of data. The numbers in the cells follow the same convention as in Table 5.2.

The performance of TCP Reno, the baseline E2E protocol, highlights the problems with TCP over error-prone links. At a 2.3% exponentially-distributed packet loss rate (as explained in Section 5.4.1), E2E achieves a throughput of less than 50% of the maximum (i.e., 50% of the throughput in the absence of wireless losses) in the LAN and less than 25% of the maximum in the WAN experiments. However, all the end-to-end protocols achieve goodputs close to the optimal value of 97.7%. The primary reason for the low throughput is the large number of timeouts that occur during the transfer (Figure 5.6, E2E graph). The resulting average window size during the transfer is small, preventing the "data pipe" from being kept full and greatly reducing the effectiveness of the fast retransmission mechanism (Figure 5.7, E2E graph). In turn, these small congestion windows increase the likelihood of timeouts.

The various end-to-end enhancements improve throughput by retransmitting packets known to have been lost on the wireless hop earlier than they would have been by the baseline E2E protocol, and by reducing the fluctuations in window size. The E2E-NEWRENO, E2E-ELN, E2E-SMART and E2E-IETF-SACK protocols each use new TCP options and more sophisticated ACK processing techniques to improve the speed and accuracy of identifying and retransmitting lost packets, as

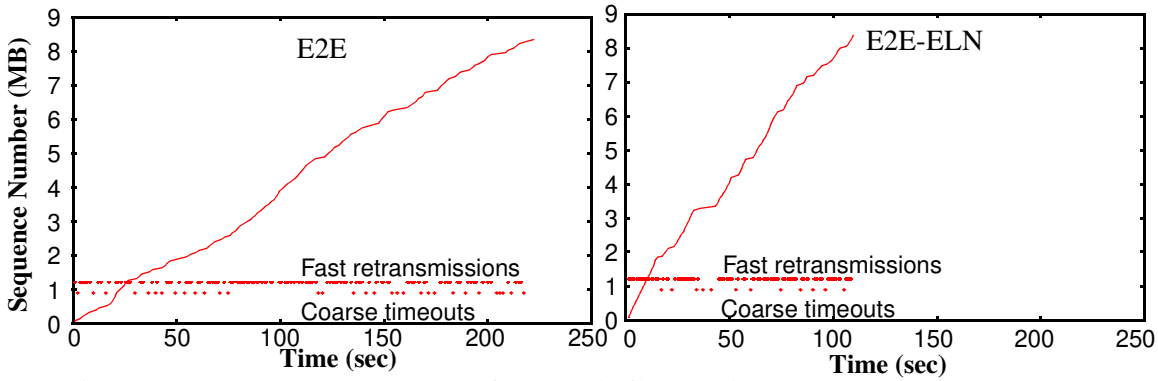

Figure 5.6 Packet sequence traces for E2E (TCP Reno) and E2E-ELN. The top row of horizontal dots shows the times when fast retransmissions occur; the bottom row shows the coarse timeouts.

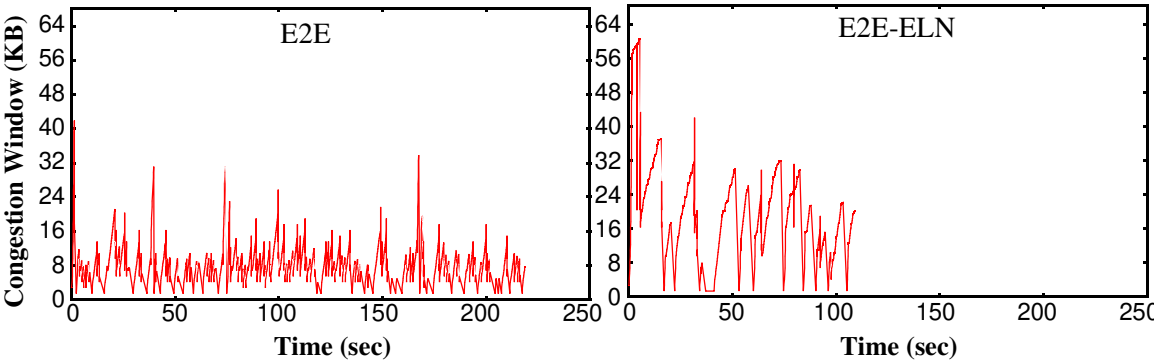

Figure 5.7 Congestion window size as a function of time for E2E (TCP Reno) and E2E-ELN. This figure clearly shows the utility of ELN in preventing rapid fluctuations, thereby maintaining a larger average congestion window size.

well as by recovering from multiple losses in a single transmission window without timing out. The remainder of this section discusses and quantifies the benefits of three techniques: *partial acknowledgments, explicit loss notifications* and *selective acknowledgments.*

Partial ACKs: E2E-NEWRENO, which uses partial ACK information to recover from multiple losses in a window at the rate of one segment per round-trip time, performs between 10% and 25% better than E2E over a LAN and about 100% better than E2E in the WAN experiments. Remaining in fast recovery when a partial ACK arrives reduces the occurrence of timeouts and directly translates to improved performance. The performance improvement is a function of the socket buffer size—the larger the buffer size, the better the relative performance. This is because in situations that E2E suffers a coarse timeout for a loss, the probability that E2E-NEWRENO does not increases with the number of outstanding packets in the network.

ELN: One way of reducing the number of coarse timeouts is to maintain as large a window size as possible. E2E-NEWRENO remains in fast recovery if the new ACK is only partial, but also performs congestion control and reduces the window size to half its original value upon the arrival of the first new ACK. The E2E-ELN and E2E-ELN-RXMT protocols use ELN information to prevent the sender from reducing the size of the congestion window in response to a wireless loss. Both these schemes perform better than E2E-NEWRENO and over a factor of two better than E2E. This is because the sender's explicit awareness of the wireless link reduces the number of coarse timeouts (Figure 5.6) and maintains a larger average window (Figure 5.7).

The E2E-ELN-RXMT protocol performs only slightly better than E2E-ELN when the socket buffer size is 32 KB. This is because there is usually enough data in the pipe to trigger a fast retransmission for E2E-ELN. The performance benefits of E2E-ELN-RXMT are more pronounced when the socket buffer size is smaller, as the results for the 8 KB socket buffer size indicate (Table 5.3). This is because E2E-ELN-RXMT does not wait for three duplicate ACKs before retransmitting a packet, if it has ELN information for it. The maximum socket buffer size of 8 KB limits the number of unacknowledged segments (i.e., the window) to a small number at any point in time, which reduces the probability of three duplicate ACKs arriving after a loss and triggering a fast retransmission, and increases the likelihood of a timeout.

Despite explicit awareness of wireless losses, timeouts sometimes occur in the ELN-based protocols. This is a result of our implementation of the ELN protocol, which does not convey information about *multiple* wireless-related losses in the same window to the sender. Since it is coupled with only cumulative ACKs, the sender is unaware of the occurrence of multiple wireless-related losses in a window.

Selective ACKs: We experimented with two different SACK schemes. In the LAN case, we used a simple SACK scheme based on SMART (Section 5.2.2). This protocol was the best of the end-to-end protocols in this situation, achieving a throughput of 1.25 Mbps (in contrast, the best link-layer scheme, LL-SMART-TCP-AWARE, obtained a throughput of 1.39 Mbps).

In the WAN case, our SACK implementation [11] was based on RFC 2018. For the exponentially-distributed loss pattern we used, the throughput was about 0.8 Mbps, significantly higher than the 0.31 Mbps throughput of TCP Reno. However, this is still about 35% worse than LL-OPT. Even

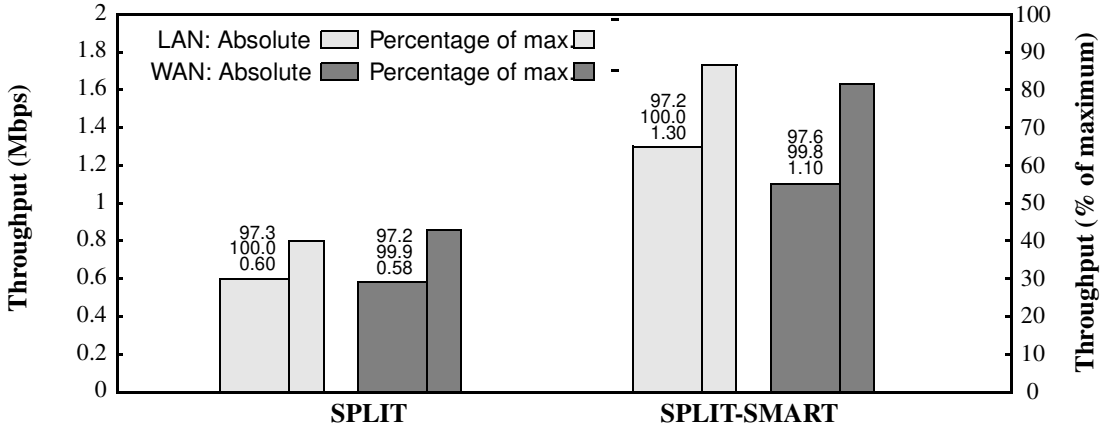

Figure 5.8 Performance of split-connection protocols: bit error rate = 1.9×10^{-6} (1 error/ 64 KB).

though SACKs allow the sender to often recover from multiple losses without timing out, the sender's congestion window decreases every time there is a packet dropped on the wireless link, causing it to remain small. The benefits of SACK are significant when windows are large and much smaller when they are small. SMART performs better because it assumes that no packet reordering occurs and retransmits segments early. We investigate in detail the problems to efficient loss recovery caused by small windows in Chapter 7.

In summary, E2E-NEWRENO is better than E2E, especially for large socket buffer sizes. Adding ELN to TCP significantly improves throughput by successfully preventing unnecessary fluctuations in the transmission window. Finally, SACKs provide significant improvement over TCP Reno, but perform about 10-15% worse than the best link-layer schemes in the LAN experiments, and over 35% worse in the WAN experiments for the particular error rate we used (1/64 KB).

5.4.3 Split-Connection Protocols

Split-connection protocols attempt to shield the sender from the vagaries of the wireless link by splitting the end-to-end connection in two at the base station. The hope is that the TCP sender (the wired connection) will be shielded by the wireless connection emanating from the base station.

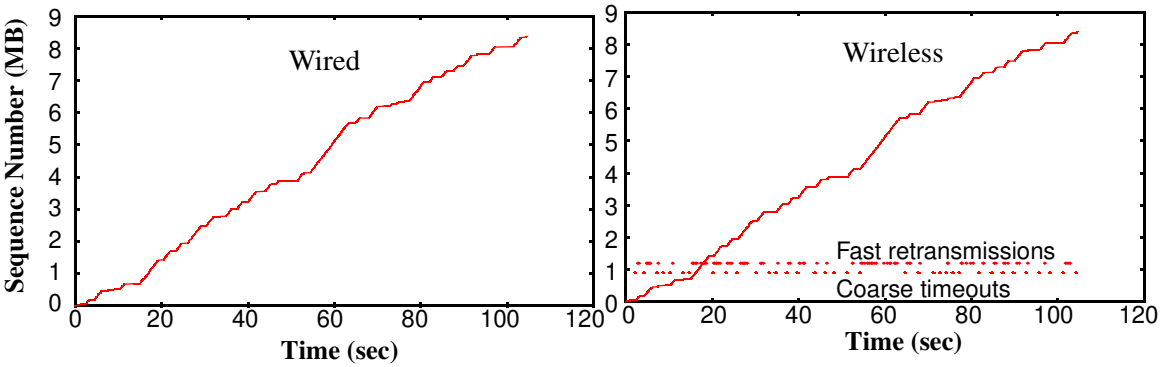

Figure 5.9 Packet sequence trace for the wired and wireless parts of the SPLIT protocol. The wireless part has two rows of horizontal dots: the top one shows the times of fast retransmissions and the bottom one the times of the timeout-based ones. The connection proceeds in fits and starts because the receiver-advertised window size from the base station to the TCP sender is often zero.

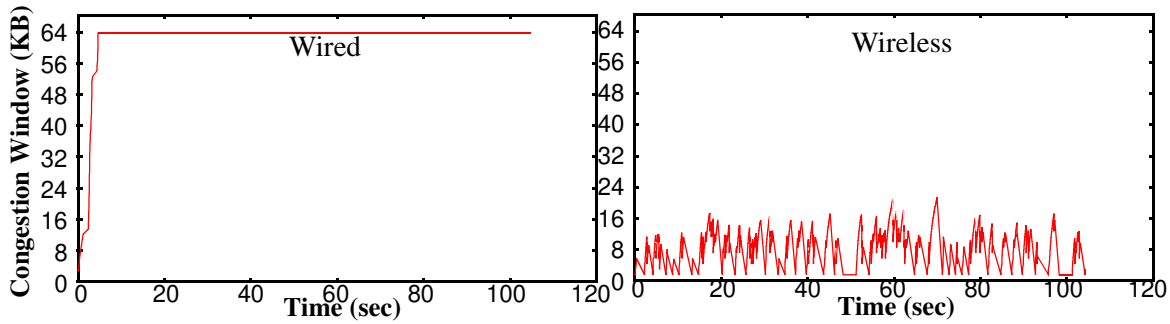

Figure 5.10 Congestion window sizes as a function of time for the wired and wireless parts of the split TCP connection. The wired sender sees no losses and maintains a 64 KB congestion window. However, the wireless TCP connection's congestion window fluctuates rapidly.

The sender of the wireless connection performs all the retransmissions in response to wireless losses.

Topology	SPLIT	SPLIT-SMART
LAN (8 KB)	0.54 (97.4%,100%)	1.30 (97.6%,100%)
LAN (32 KB)	0.60 (97.3%,100%)	1.30 (97.2%,100%)
WAN (32 KB)	0.58 (97.2%,100%)	1.10 (97.6%,100%)

Table 5.4 Summary of results for the split-connection schemes at an exponentially distributed error rate of 1 every 64 KB, on average.

Figure 5.8 and Table 5.4 show the throughput and goodput for the split-connection protocols in the LAN and WAN environments. We report the results for two cases: when the wireless connection uses TCP Reno (labeled SPLIT) and when it uses the SMART-based SACK scheme (labeled

SPLIT-SMART). In the LAN case, we see that the throughput achieved by SPLIT is quite low (0.6 Mbps), about the same as that for end-to-end TCP Reno (labeled E2E in Figure 5.5). The reason for this is apparent from Figures 5.9 and 5.10, which show the progress of the data transfer and the size of the congestion window for the wired and wireless connections. We see that the wired connection neither has any retransmissions nor any timeouts, resulting in a wired goodput of 100%. However, it periodically stalls whenever the sender of the wireless connection experiences a timeout, since the amount of buffer space at the base station (64 KB in our experiments) is bounded. Furthermore, a larger buffer at the base station will not improve steady-state performance for two reasons: (1) we measure performance in terms of receiver throughput, which is limited by the small congestion window size of the wireless connection, and (2) a long enough transfer will fill the buffer anyway. Thus, the performance of SPLIT is indistinguishable from TCP Reno, demonstrating that the split architecture per se does not yield any performance benefits.

In the WAN case the throughput of the SPLIT approach is about 0.58 Mbps, which is better than the 0.31 Mbps that the E2E approach achieves (Figure 5.5), but far inferior to the other protocols described earlier. The large congestion window size of the wired sender in SPLIT enables a higher bandwidth utilization over the wired network, compared to an end-to-end TCP connection where the congestion window size fluctuates rapidly.

As expected, the throughput for the SPLIT-SMART scheme is much higher. It is about 1.3 Mbps in the LAN case and about 1.1 Mbps in the WAN case. The SMART-based selective ACK scheme operating over the wireless link performs very well, especially since no reordering of packets occurs over this hop. However, there are a few times when both the original transmission and the first retransmission of a packet get lost, which sometimes results in a coarse timeout. This explains the 10-15% difference in throughput between the SPLIT-SMART scheme and the LL-SMART-TCP-AWARE scheme (Figure 5.2).

In summary, while the split-connection approach results in good throughput if the wireless connection uses special mechanisms, the performance is worse than that of a TCP-aware reliable link-layer protocol (LL-TCP-AWARE or LL-SMART-TCP-AWARE). The performance of the base SPLIT protocol is not significantly different compared to TCP Reno (E2E). Moreover, the link-layer protocols preserve the end-to-end semantics of TCP ACKs and do not incur the architectural

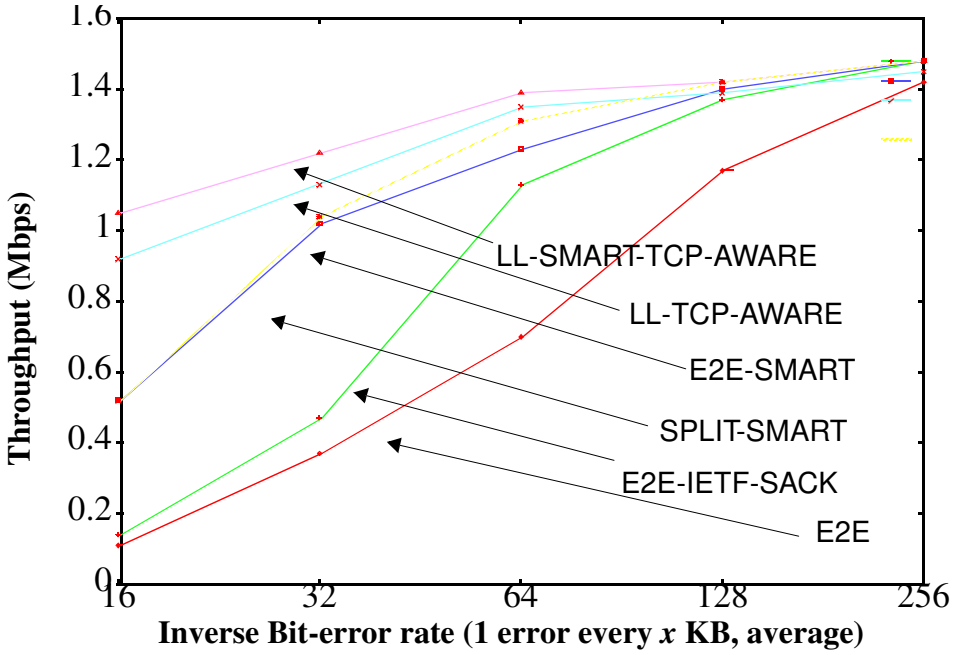

Figure 5.11 Performance of six protocols (LAN case) across a range of bit-error rates, ranging from 1 error every 16 KB to 1 every 256 KB shown on a log-scale.

drawbacks of the split-connection approaches. This demonstrates that the end-to-end connection need not be split at the base station to achieve good performance.

5.4.4 Performance at Different Error Rates

In this section, we present the results of several experiments performed across a range of bit-error rates for some of the protocols described earlier: E2E (the baseline case), LL-TCP-AWARE, LL-SMART-TCP-AWARE, E2E-SMART, E2E-IETF-SACK, and SPLIT-SMART. We choose the best performing protocols from each category, as well as some other protocols (e.g., E2E-IETF-SACK) to illustrate some interesting effects.

Figure 5.11 shows the performance of these protocols for an 8 MByte end-to-end transfer in a LAN environment for exponentially distributed error rates ranging from one error every 16 KB to one error every 256 KB, in increasing powers of two. We note that the overall qualitative results and conclusions are similar to those presented earlier for the 64 KB error rate. At low error rates (128 KB and 256 KB points in the graph), all the protocols shown perform almost equally well in improving TCP performance. At higher error rates, the performance of the TCP-aware link-layer

schemes is about 1.75-2 times better than E2E-SMART and about 9 times better than TCP Reno. These graphs clearly show the benefits of TCP-aware link protocols like the Snoop protocol and the Snoop protocol enhanced with SACKs.

Another interesting point to note is the relative performance of E2E-IETF-SACK and E2E-SMART, especially at the high error rates. The congestion window does not grow larger than a few packets in the steady state at these error rates where there are multiple losses in many windows. E2E-IETF-SACK does not retransmit any packet using SACK information unless it receives three duplicate ACKs, to guard against potential packet reordering of packets on the path. This implies that no fast retransmissions are triggered if the number of segments in the window is less than four. The sender's congestion window is often smaller than this, leading to timeouts and degraded performance. In contrast, our implementation of E2E-SMART assumes no reordering of packets (which is justified on a LAN) and retransmits the lost packet when the first duplicate ACK arrives. This reduces the number of timeouts and leads to better end-to-end performance. We note that this approach only points out the benefits of avoiding timeouts and is not the final solution to this problem, because it unrealistically assumes that no packet reordering can occur in the Internet. In Chapter 7 we present a robust solution to this problem caused by small transmission windows.

5.5 Enhancements to the Snoop Protocol

In this section, we discuss two enhancements to the Snoop protocol. The first improves performance when there are cellular wireless transit links on the path that bridge wired clouds. The second exploits selective ACK information to perform better loss recovery when wireless losses occur in bursts.

5.5.1 Wireless Transit Links

The ELN scheme presented in Section 4.4.2 for improving end-to-end performance from a mobile TCP sender also generalizes to cellular wireless transit links where the TCP ACKs traverse the same base stations as the data packets on the forward path [12]. Consider a connection over one cellular wireless transit link and other wired links, as shown in Figure 5.12. Suppose a loss happened due to corruption over the wireless link. In this case, the agent at base station A would have seen the packet, while B would not, so the packet gets added to B's list of holes using the same

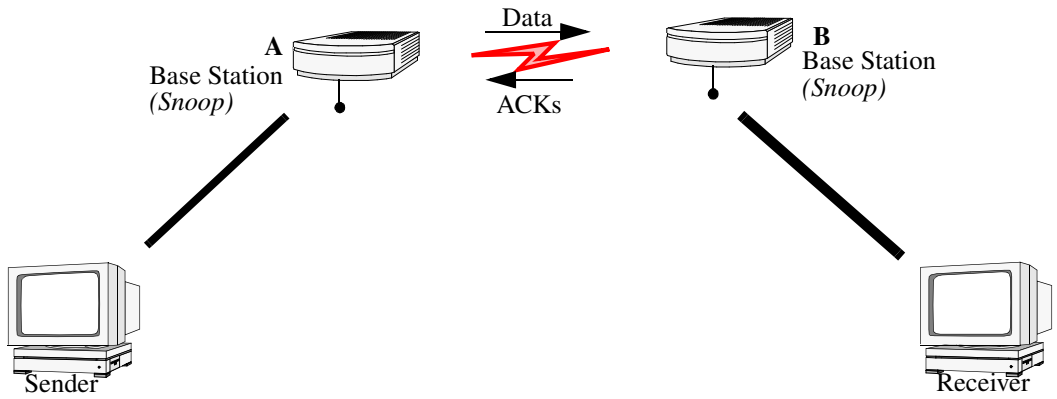

Figure 5.12 Topology for data transfer over a cellular wireless transit link. Both congestion and corruption could occur, necessitating active involvement by the base stations in demarcating and distinguishing the two types of losses to the TCP sender.

algorithm as in Section 4.4.2. When duplicate ACKs arrive for the missing packet at B, its agent sets the ELN information bit and propagates it towards A. Since A originally saw the packet (it is not in its list of holes), it infers that the packet was lost over the wireless link and lets the ACK go through to the data sender with the ELN information set on it, for the sender to perform a retransmission without invoking congestion control. Of course, it is possible to deploy a Snoop agent that caches and locally retransmits packets at A in this topology. In this case, the agent also suppresses duplicate ACKs for corrupted packets, as described earlier. It is important to note that local retransmissions from the Snoop agent at A now happen only upon duplicate ACKs with ELN set.

Now suppose that the packet was lost due to *congestion* elsewhere on the path. If it was lost before A, then the agent at A, upon seeing an ELN ACK via B, *resets* this bit because this packet did not reach A to start with. The sender reacts to this loss in the conventional way and performs congestion control. Similarly, if the packet was lost due to congestion after B, no agent in the network sets the ELN information on any duplicate ACK (because both A and B would have seen the packet), leading to correct congestion control. It is easy to see how this approach generalizes to the case when there is more than one single-hop wireless link on the path. Thus, we use ELN to differentiate between congestion and corruption in these topologies.

Our performance experiments show that in the first case (TCP sender retransmissions), observed performance is identical to the ELN case shown in Figure 4.9. In the second case (Snoop agent at A), performance closely tracks that of the last-hop Snoop agent, presented in Section 4.7.

5.5.2 Handling Burst Losses

In this section, we report the results of experiments that illustrate the benefit of selective ACKs in handling burst losses. We consider two of the best performing local protocols: LL-TCP-AWARE (Snoop) and LL-SMART-TCP-AWARE (Snoop with SMART-based selective ACKs). LL-TCP-AWARE recovers from a single loss by retransmitting the lost packet when a duplicate ACK arrives for it. It also keeps track of the number of expected duplicate ACKs and the next expected new ACK after this local retransmission. If this loss is part of a burst, the first new ACK to arrive after the duplicates will be less than the next expected new one, which causes an immediate retransmission of the lost segment. Thus, LL-TCP-AWARE attempts to recover from burst losses in a prompt manner, but resorts to a timeout if it is unsuccessful. LL-SMART-TCP-AWARE uses the additional useful information provided by the SMART-based selective ACKs—the sequence number of the segment that caused the duplicate ACK—to accurately determine losses and recover from them.

Burst Length	LL-TCP-AWARE (Mbps)	LL-SMART-TCP-AWARE (Mbps)
2	1.25	1.28
4	1.02	1.20
6	0.84	1.10

Table 5.5 Throughputs of Snoop and LL-SMART-TCP-AWARE at different burst lengths. This illustrates the benefits of SACKs, even for a high-performance, TCP-aware link protocol.

Table 5.5 shows the performance of the two protocols for bursts of lengths 2, 4, and 6 packets. These errors are generated at an average rate of one every 64 KB of data, and 2, 4, or 6 packets are destroyed in each case. Selective ACKs improve the performance of LL-SMART-TCP-AWARE over LL-TCP-AWARE by up to 30% in the presence of burst errors. While this is a simplistic burst-error model, it does illustrate the effectiveness of the TCP-aware schemes and selective ACKs in recovering from the loss of multiple packets in close succession.

5.6 Discussion

This section reflects on some key lessons we have learned from our experience with reliable transport over error-prone wireless networks.

- **Cross-layer protocol optimizations:** For data transfer to a mobile host, the Snoop protocol uses a *transport-aware* link protocol between the base station and mobile host to achieve good end-to-end performance for bulk and Web-like transfers. For data transfer from a mobile host, it uses ELN to make the transport protocol *link-aware*. This approach violates the conventional layered model used in network protocol design, where there is no explicit control information transferred between higher and lower layers of the protocol stack. The benefits of our solutions in both simulation and real deployments show that better protocol adaptation is enabled by making different layers of the protocol stack more aware of what the other layers are doing and also of what the underlying channel is doing. This approach achieves superior end-to-end performance because it *explicitly* prevents adverse interactions that could occur between independent protocols at different layers of the stack.

- **The end-to-end argument:** The design of the Snoop agent and protocol has been strongly influenced and guided by the end-to-end argument and design principle propounded by Saltzer, Reed and Clark [127]. The end-to-end argument is a design principle that guides the placement of services in a system. It states that a service should be implemented in a subsystem only if the service can be completely implemented in that subsystem, or if by partially implementing the service in the subsystem, it substantially improves end-to-end performance. This is precisely the philosophy that guided the design of the Snoop protocol and its approach toward local recovery. In contrast to split-connection approaches, the Snoop protocol does not violate the end-to-end semantics of TCP and is not vulnerable to base station failures. The good performance of some of the split connection approaches derives not from their spoofing ACKs and breaking the connection, but from their running a wireless-optimized protocol over the noisy link [14]. And, it is possible to obtain the same or higher benefit using the Snoop approach, without any change in end-to-end semantics.

- **Soft-state agents in the network infrastructure:** The performance benefits of deploying a soft-state Snoop agent at the base station are significant. In general, we believe that this is an efficacious way in which network heterogeneity and associated performance problems can be solved, using agents in the network infrastructure. The Snoop agent state is soft, because it is not crucial to the correct end-to-end functioning of the system and because it is periodically and automatically refreshed upon the arrival of data and ACKs. Any errors in the state do not compromise the correctness of the connection; incorrect state is automatically rectified when new

messages arrive. This soft-state approach also enables low handoff latencies with little connection disruption, since the contents of the Snoop cache can be mirrored using multicast from the home agent (Section 4.5.1).

- **Data-driven retransmissions:** Reliable transport protocols can recover from losses in two ways: using data-driven retransmissions, which infer that an earlier packet is missing from the receipt of later data, and timeouts, which are triggered when the sender does not receive an ACK within a certain period of time. Data-driven retransmissions recover from losses with little disruption in the stream, while timeouts necessarily have to be conservative to ensure that data that is delayed but still in transit does not get prematurely retransmitted [150]. In the Snoop protocol, we reduce the dependence on local timers by performing data-driven retransmissions as often as possible.

- **Higher priority local retransmissions:** Retransmitting packets at a higher priority improves performance at all error rates. This enables retransmitted packets to reach the mobile host sooner, reducing the number of duplicate ACKs and leading to improved throughput. In the common case when the wireless link is the connection's bandwidth bottleneck, the first new ACK generated after a higher-priority retransmission is usually less than what it would be without such prioritization. This implies that the number of segments transmitted by the sender in a burst is smaller in the former case, because the sliding window shifts by a smaller amount. This in turn reduces the degree of congestion in the network by reducing burst transmissions, leading to improved overall performance.

  Higher priority local retransmissions also allow us to avoid unnecessary local timeouts in the following way. When a retransmission is scheduled, the Snoop agent precisely knows what the last in-sequence transmission was. Hence, it knows what the next expected new ACK should be, assuming no further losses happen. Now, if the next new ACK is smaller than the next expected one, a retransmission occurs without a timeout. This helps in recovering from multiple losses in a single window without a timeout.

- **Ineffectiveness of end-to-end TCP SACK:** An interesting observation concerns the lack of effectiveness of purely end-to-end TCP SACK in the wireless environment, despite the sender's ability to recover quickly from multiple losses in a window when SACK information is available. Initially, this was surprising to us because of SACK's benefits in burst loss situations.

However, upon closer analysis [14], we found that even with SACK, the sender incurred numerous coarse timeouts. With a large enough window, burst losses over the wireless link are indeed recovered quickly, but the sender still performs congestion control and reduces its window. The end result is a small average window size, with only a small number of outstanding segments. As a result, there is not enough information for the sender to distinguish between reordered and lost packets, necessitating coarse timeouts. We present a robust solution to this problem based on enhancing TCP's loss recovery algorithm in Chapter 7.

- **Sharing loss recovery information:** When there are multiple connections outstanding to the same mobile host, the Snoop agent can perform loss recovery in an integrated way by effectively sharing ACK information between them. We do this by treating the different connections as one integrated stream and by keeping track of the relative transmission order between packets belonging to different connections. Then, when an ACK arrives on a connection that acknowledges a data item that was transmitted *after* a currently unacknowledged segment belonging to another connection, we treat this as a *later* ACK indication for the latter connection. When the number of *later* ACKs reaches a threshold, we retransmit the earliest missing segment, independent of the connection(s) that generated this later ACK information. Results of an end-to-end integrated loss recovery scheme in the context of TCP are presented in [15, 107].

- **End-to-end security:** An important issue concerning the viability of approaches like the Snoop protocol and split connection approaches is their interaction with end-to-end IP Security. If a connection has end-to-end security enabled, its entire IP payload, including the transport header, would be encrypted and therefore unreadable by network agents like the Snoop agent. One possible solution is for the source to not encrypt part of the TCP header—this would not compromise the security model provided by IP Security and would overcome the problem. Another (less attractive) possibility is to include the base station as part of the mobile host's trust infrastructure and allow it to decrypt packets destined for the mobile host. Developing a standardized way to solve this problem without compromising end-to-end security is currently gaining momentum in the Internet Engineering Task Force (IETF).

5.7 Summary

In this chapter, we presented a comparative analysis of several techniques to improve the end-to-end performance of TCP over error-prone wireless links. We categorized these as end-to-end, link-layer or split-connection schemes. We used end-to-end throughput and wired and wireless goodputs as metrics for comparison.

Our results lead to the following conclusions:

1. A reliable link-layer protocol that uses knowledge of TCP (LL-TCP-AWARE) to shield the sender from duplicate ACKs arising from wireless losses gives a 10-30% higher throughput than one (LL) that operates independently of TCP and does not attempt in-order delivery of packets. Also, the former avoids redundant retransmissions by both the sender and the base station, resulting in a higher goodput. Of the schemes we investigated, the TCP-aware link-layer protocol with SACKs performs the best. Thus, our conclusion is that such cross-layer optimizations as in the Snoop protocol are beneficial in obtaining good performance in the face of wireless bit-errors.

2. End-to-end schemes, while not as effective as local techniques in handling wireless losses, are promising since significant performance gains can be achieved without extensive support from intermediate nodes in the network. The ELN scheme resulted in a throughput improvement of more than a factor of two over TCP-Reno with comparable goodput. This is the link-aware transport algorithm used by the Snoop protocol for transfers from the mobile host.

3. The split-connection approach with TCP Reno used over the wireless link does not perform well. The sender periodically stalls due to timeouts on the wireless connection, leading to poor end-to-end throughput. A wireless-optimized SMART-based SACK mechanism for the wireless hop yields better throughput. However, it is still less than that of a transport-aware link-layer protocol that does not split the connection. This demonstrates that splitting the end-to-end connection is not essential for good performance in wireless systems and that the good performance of SPLIT-SACK is not a consequence of the split-connection architecture.

4. The SMART-based SACK scheme is quite effective at handling the high packet loss rate when deployed over the wireless hop or by the sender in a LAN environment. In the WAN experiments, the SACK scheme based on RFC 2018 improves end-to-end performance, although its

performance is not as good as the link-layer schemes. From our results we conclude that SACKs are very useful in the presence of error-prone links, especially when losses occur in bursts. However, the performance benefits of *end-to-end* SACK protocols are not impressive because average windows remain small, and the sender does not have sufficient information to distinguish genuine losses from potentially reordered packets.

5. Using ELN and soft-state network agents, we developed an enhancement to the Snoop protocol that performs well when error-prone wireless transit links are part of the connection.

In summary, our results show that cross-layer protocol optimizations significantly improve transport performance over error-prone wireless links in a variety of topologies. We demonstrated this using a transport-aware link protocol and a link-aware transport protocol that uses the ELN mechanism to distinguish congestion from corruption.

The next chapter discusses the problems caused by asymmetric effects on TCP performance and presents our solutions to them.

"What immortal hand or eye
Could frame that fearful symmetry?"

— William Blake, 'The Tiger'

Chapter 6

Effects of Asymmetry

This chapter investigates the problems to TCP performance that arise because of asymmetric effects. These problems arise in several wireless networks including bandwidth-asymmetric networks and latency-asymmetric packet radio networks, for different underlying reasons. However, the effect on TCP performance is the same in both cases: performance degrades significantly because of imperfections and variabilities in the acknowledgment (ACK) feedback from the receiver to the sender. We investigate these problems in both bandwidth-asymmetric and packet radio networks and solve them. Our solutions to these problems use a combination of link-layer and end-to-end algorithms in a two-pronged approach: first, we devise techniques to reduce the frequency of TCP ACKs; then, we develop techniques to handle reduced ACK frequency.

6.1 Motivation

Our work is motivated by two very different network technologies: bandwidth-asymmetric networks like wireless cable modem networks and direct broadcast satellite networks on the one hand, and multihop wireless packet radio networks on the other. In the first case, the reverse path used by TCP ACKs is constrained in bandwidth, with slow or loss-prone ACK feedback degrading TCP performance in the forward direction. In packet radio networks, traffic flowing simultaneously in

different directions adversely affects performance. This is because most packet radio networks use half-duplex radio units, which cannot transmit and receive data frames at the same time. The interactions between TCP and the media-access protocols in these networks cause significant ACK queues to build up, leading to highly variable communication latencies and round-trip times. TCP does not adapt well to this variability.

Despite the technological differences between asymmetric-bandwidth and packet radio networks, TCP performance suffers in both these networks because of slow, imperfect and variable ACK feedback. We address this root problem and solve it in general. The end result is a set of solutions that improve performance in different asymmetric situations, independent of the specific technology in question.

Asymmetry is inherent in the design of many wireless networks. For example, many systems have a central transmitter (or base station) transmitting at high power to receiving mobile units. However, to reduce power consumption, the portable units transmit to the base station at much lower power levels. In addition, they often have to contend with other mobile units to gain access to the channel. Thus, while packets from the base station reach the mobile hosts promptly, the converse is not always true.

Asymmetric network technologies are common in mobile ad hoc environments too. For example, consider an ad hoc network of mobile hosts communicating with each other internally via a packet radio network, whose external link to the Internet is over an asymmetric-bandwidth wireless cable modem network. Motivated by this scenario, we combine these two wireless access technologies in our study—wireless cable and packet radio—to investigate the problems to transport performance that then arise.

We generalize the various phenomena and examples described above to the following general, technology-independent definition of asymmetry: *a network is said to exhibit asymmetry with respect to TCP performance, if the throughput achieved is not solely a function of the link and traffic characteristics of the forward direction, but also depends significantly on those of the reverse direction*. Here, *forward* refers to the direction from the sender to the receiver and *reverse* the opposite. This general definition immediately permits a classification of different types of asym-

metry. In addition to *bandwidth asymmetry* described above, this definition extends to *latency*, *media-access* and *error-rate* asymmetries.

In this chapter, we study bandwidth and latency asymmetry, both in isolation and in combination, using Hybrid's wireless cable modem network and Metricom's Ricochet packet radio network as motivating technologies. We use measurements on a real testbed as well as simulations experiments with different topologies and workloads to identify performance problems. Based on these results, we design and evaluate several techniques to improve performance.

Fundamentally, network asymmetry affects the performance of reliable transport protocols such as TCP because these protocols rely on feedback in the form of ACKs from the receiver to ensure reliability and smooth transmissions. Because TCP transmissions are ACK-clocked (Section 2.2.3), any disruption in the feedback process can impair the performance of the forward data transfer. For example, a low bandwidth ACK path could significantly impede the growth of the TCP sender window during slow start, *independent* of the link bandwidth in the direction of data transfer. A second example occurs in packet radio networks, where variable latencies in the presence of bidirectional traffic (caused by ACKs flowing in a direction opposite to data packets) causes the sender's round-trip time estimate to be highly variable. This inflates TCP's retransmission timeout value, significantly impairing prompt loss recovery.

We investigate the problems caused by asymmetry on TCP performance and design solutions to overcome these problems. These include both end-to-end and link-layer techniques such as *ACK filtering*, *TCP sender adaptation,* and *ACK reconstruction*. These solutions improve individual connection throughputs, distribute throughputs fairly amongst connections and improve the scaling properties of asymmetric systems by being able to support more connections for a fixed available bandwidth.

6.2 Asymmetric Bandwidth Networks

In this section, we discuss the problems that degrade unidirectional transfer performance in bandwidth asymmetric networks, shown in Figure 6.1. Depending on the characteristics of the reverse path, two types of situations arise for unidirectional traffic over asymmetric-bandwidth networks:

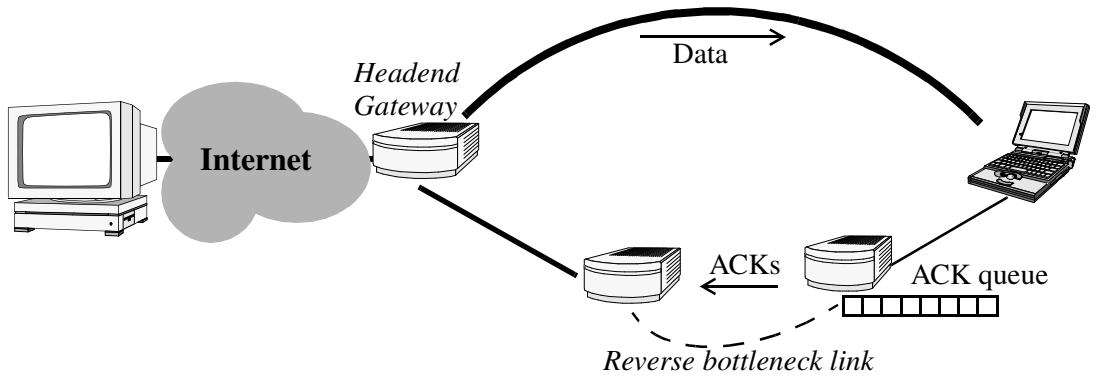

Figure 6.1 Schematic representation of a bandwidth-asymmetric network. If the reverse bottleneck queue is large, ACKs are queued there and reach the sender slowly, which reduces the forward throughput.

when the reverse bottleneck link has sufficient queueing to prevent packet (ACK) losses, and when the reverse bottleneck link has a small buffer. We consider each situation in turn.

6.2.1 Deep Reverse Queues

If the reverse bottleneck link has deep queues (Figure 6.1) so that ACKs do not get dropped on the reverse path, then performance is a strong function of the *normalized bandwidth ratio*, k, defined by Lakshman and Madhow [78]. k is the ratio of the raw bandwidths divided by the ratio of the packet sizes used in the two directions. For example, for a 10 Mbps forward channel and a 50 Kbps reverse channel, the raw bandwidth ratio is 200. With 1000-byte data packets and 40-byte ACKs, the ratio of the packet sizes is 25. This implies that k is 200/25 = 8. Thus, if the receiver acknowledges more frequently than one ACK every $k = 8$ data packets, the reverse bottleneck link will get saturated before the forward bottleneck link does, limiting the throughput in the forward direction.

If $k > 1$ and ACKs are not delayed or dropped, TCP ACK-clocking breaks down. Consider two data packets transmitted by the sender in quick succession. En route to the receiver, these packets get spaced apart according to the bottleneck link bandwidth in the forward direction. The principle of ACK clocking is that the ACKs generated in response to these packets preserve this temporal spacing all the way back to the sender, enabling it to transmit new data packets that maintain the same spacing. However, the limited reverse bandwidth and queuing at the reverse bottleneck router alters the inter-ACK spacing observed at the sender. When ACKs arrive at the bottleneck link in the reverse direction at a faster rate than the link can support, they get queued behind one another. The spacing between them when they emerge from the link is *dilated* with respect to their original

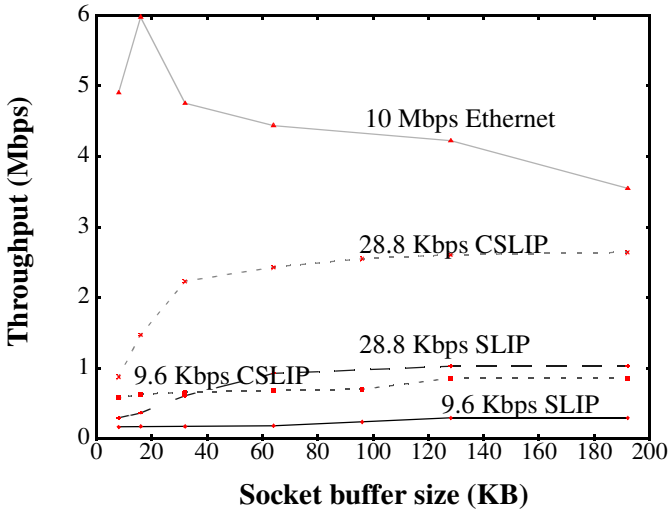

Figure 6.2 Measured performance of the Hybrid wireless cable network using different return channels, across a range of socket buffer sizes. Each run of the experiment involved the transfer of 1 MB of data in the forward direction between two BSD/OS hosts.

spacing, and is a function of the reverse bottleneck bandwidth. Thus the sender clocks out new data at a slower rate than if there had been no queuing of ACKs. No longer is the performance of the connection dependent on the forward bottleneck link alone; instead, it is throttled by the rate of arriving ACKs. As a side-effect, the sender's rate of congestion window growth slows down too.

Figure 6.2 shows the measured performance of a bulk TCP transfer over an asymmetric wireless cable networks using different reverse ACK channels, over reverse bottleneck links that have sufficient queueing. For each return bandwidth, SLIP header compression (CSLIP) [60] reduces the sizes of ACKs and reduces k, thus improving performance over the uncompressed case. For example, consider the case of a 10 Mbps forward and 28.8 Kbps reverse channel, with a data packet size of 1 KB. With the TCP timestamp option enabled, the ACK size is 52 bytes with SLIP and 18 bytes with CSLIP. So, k is 18.05 with uncompressed SLIP and 6.25 with CSLIP. With TCP delayed ACKs (one ACK for every two data packets), throughput is limited to 10*2/18.05 = 1.1 Mbps and 10*2/6.25 = 3.2 Mbps, with SLIP and CSLIP respectively. These numbers closely match the measured throughputs shown in Figure 6.2. Thus, for unidirectional traffic, CSLIP results in significant improvements in performance. In comparison, the performance with an Ethernet return channel is much better because of the absence of bandwidth asymmetry (k = 0.052 << 1) and a much smaller link delay than the dialup lines.

6.2.2 Shallow Reverse Queues

The second situation arises when the reverse bottleneck link has a relatively small amount of buffer space to accommodate ACKs. As the transmission window grows, this queue fills and ACKs are dropped. If the receiver acknowledges every packet, only one of every k ACKs get through to the sender, and the remaining *(k-1)* are dropped due to buffer overflow at the reverse channel buffer (here k is the normalized bandwidth ratio). In this case, the reverse bottleneck link capacity and slow ACK arrival are not *directly* responsible for any degraded performance. However, there are three important reasons for degraded performance in this case because ACKs are infrequent:

1. First, the sender transmits data in large bursts. If the sender receives only one ACK in k, it transmits data in bursts of k (or more) segments because each ACK shifts the sliding window by at least k (acknowledged) segments. This increases the likelihood of data loss along the forward path especially when k is large, because routers do not handle large bursts of packets well.
2. Second, TCP sender implementations increase their congestion window by counting the *number* of ACKs they receive and not on *how much* data is actually acknowledged by each ACK. Thus fewer ACKs implies a slower rate of growth of the congestion window, which degrades performance over long-delay connections.
3. Third, the sender's fast retransmission and recovery algorithms are less effective when ACKs are lost. The sender may not receive the threshold number of duplicate ACKs even if the receiver transmits more than the required number. Furthermore, the sender may not receive enough duplicate ACKs to adequately inflate its window during fast recovery.

In summary, bandwidth asymmetric networks suffer from degraded performance due to slow or imperfect ACK feedback.

6.3 Latency Asymmetry in Packet Radio Networks

We now look at the subtle effects that manifest themselves when TCP connections run over a packet radio network. While we use a network modeled after Metricom's Ricochet network in this section, our general results and conclusions are applicable to other packet radio networks as well. We start by describing the underlying packet radio network technology.

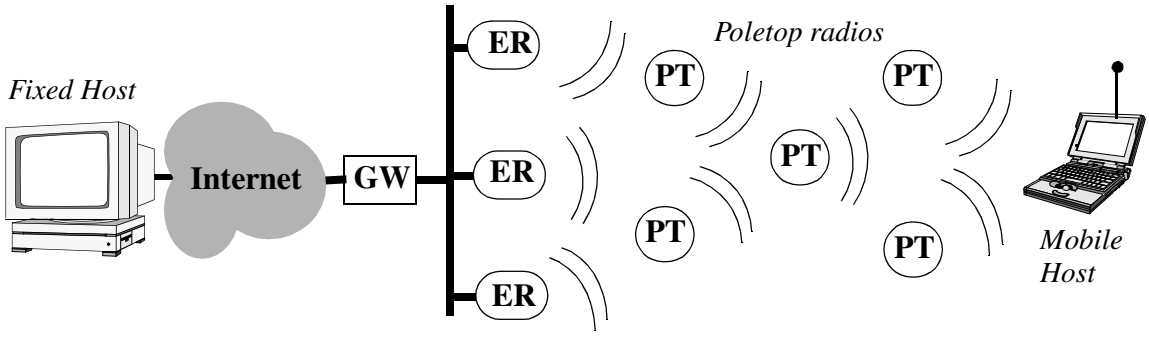

Figure 6.3 Topology of the Ricochet packet radio network. The Mobile Host (MH) has a modem attached to it, which communicates with a Fixed Host (FH) on the Internet through the Poletop Radios (PT) and Ethernet Radios (ER). The Gateway (GW) routes packets between the packet radio network and the Internet.

6.3.1 Technology

In this section, we describe the network topology, the characteristics of the radio units, and the media-access (MAC) and link-layer protocols in the packet radio network that displays latency asymmetry.

Topology: The network topology of the packet radio network is shown in Figure 6.3. The maximum link speed between any two nodes in the wireless cloud is 100 Kbps. Packets from a fixed host (FH) on the Internet are routed via a gateway (GW) and through the poletop radios (PT), to the end mobile host (MH). Similarly, packets from the MH are routed in the opposite direction. The Ethernet Radios (ER) interface between the wired Internet and the wireless links. Typically, there are between one and three wireless hops before the packet reaches the MH from the GW and vice versa.

Half-duplex radios: Because the power of a transmission drowns incoming receptions in the same frequency band, the radio units in the network cannot simultaneously transmit and receive data—i.e., they are *half-duplex.* After transmitting (receiving) a packet, a sender (receiver) wishing to receive (transmit) a packet needs to *turnaround* and change modes. Moving from transmitting to receiving mode takes a non-trivial amount of time, called the transmit-to-receive turn-around time, T_{TR}. Similarly, going from receiving to transmitting mode takes a time equal to the receive-to-transmit turnaround time, T_{RT}. Typical values of these times are shown in Table 6.2 on page 153.

Media-access: The radios in this network are frequency-hopping, spread-spectrum units operating in the 915 MHz ISM band. The details of the frequency-hopping protocol are not relevant to transport performance, since the predominant reason for poor performance is the interaction between the MAC protocol and TCP. The MAC protocol for contention resolution is based on a polling scheme, similar (but not identical) to the RTS/CTS ("Request-To-Send/Clear-To-Send") protocol used in the IEEE 802.11 standard. A station wishing to communicate with another (called the peer) first sends it an RTS message. If the peer is not currently communicating with any other station and is willing to receive this transmission, it sends a CTS message acknowledging the RTS. When this is received by the initiator, the data communication link is established. A data frame can then be sent to the peer. If the peer cannot currently communicate with the sender because it is communicating with another peer, it does not send a CTS, which causes the sender to backoff for a random amount of time and reschedule the transmission for later. It could also send a NACK-CTS to the sender, which achieves the same effect. In all this, care is taken by both stations to ensure that messages and data frames are not lost because the peer was in the wrong mode, by waiting enough time for the peer to change modes. To do this, each station maintains the value of the turnaround times of its peers in the network.

Error-control: The reliable link-layer protocol used in this network for error-control is a simple frame-by-frame protocol with a window size of one. When a frame is successfully received, the receiver sends a link-level ACK to the sender. If the frame is not received successfully, the sender retransmits it after a timeout interval. Such simple link-layer protocols are the norm in several packet radio networks (see, e.g., [68]).

6.3.2 Analysis of Problems

We now discuss the results of several measurements and simulations under various network topologies and traffic workloads. We start with the simplest case of a bulk TCP transfer across one wireless hop running the MAC and link-layer protocols described above.

Variable delays and ACK queueing: The need for the communicating peers to first synchronize via the RTS/CTS protocol before communication and the significant turn-around time for the radios result in a high per-packet overhead. Furthermore, since the RTS/CTS exchange needs to

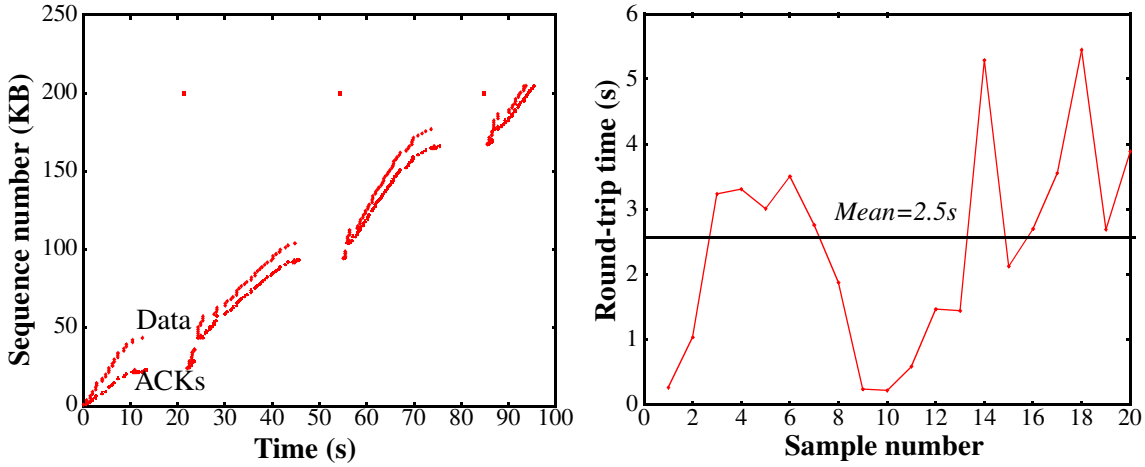

Figure 6.4 (a) Packet and ACK sequence trace of a 200 KB TCP bulk transfer measured over one wireless hop in the Ricochet network. The three pauses are sender timeouts, lasting between 9 and 12 seconds each. This happens because the retransmission timeout estimate is large. (b) Twenty successive round-trip time samples collected during this connection are shown. The samples have a mean of about 2.5 s and a standard deviation of about 1.5 s. The combination of the 2.5 s mean round-trip and large deviation leads to long timeouts.

back off when the polled radio is busy (for example, engaged in a conversation with a different peer), this overhead is variable. This leads to large and variable communication latencies in packet-radio networks. In addition, with an asymmetric workload with most data flowing in one direction to clients, ACKs tend to get queued in certain radio units (especially in the client modems), exacerbating the variable communication latencies.

These variable latencies and queueing of ACKs adversely affect smooth data flow. In particular, TCP ACK traffic interferes with the flow of data and increases the traffic load on the system. Figure 6.4(a) shows the packet sequence trace of a measured 200 KB TCP transfer over an unloaded wired path and one wireless hop in the Ricochet network. This clearly shows the effect of the radio turnarounds and increased variability affecting performance. The connection progresses well for the most part, except for the three timeouts that occur during the transfer. These timeouts last between 9 and 12 seconds each, keeping the connection idle for a total of 35% of the total transfer time! These timeouts clearly under-utilize the available bandwidth for the connection.

Why are these timeouts so long in duration? Ideally, the round-trip time estimate (*srtt*) of a TCP data transfer will be relatively constant (i.e., have a low linear deviation, *rttvar*). Then the TCP retransmission timeout, set to *srtt + 4*rttvar* (Section 2.2.2), will track the smoothed round-trip time estimate and respond well when multiple losses occur in a window. Unfortunately, this is not

true for connections in this network, as shown in Figure 6.4(b). This figure plots the individual round-trip time estimates for 20 successive samples during the same connection over the Ricochet network. The mean value of these samples is about 2.5 seconds and the standard deviation is large—about 1.5 seconds. Because of the high variation in the individual samples, the retransmission timer is on the order of 10 seconds, leading to the long idle timeout periods.

In general, it is correct for the retransmission timer to trigger a segment retransmission only after an amount of time dependent on both the round-trip time and the linear (or standard) deviation. If only the mean or median round-trip estimates were taken into account, the potential for spurious retransmissions of segments still in transit is large. Thus, our challenge is to devise solutions to this problem where the increased ACK flow and ACK queueing lead to variable latencies and long timeout periods.

An optimization to the link-layer error-control protocol that piggybacks link-layer ACKs with data reduces TCP round-trip variations to some extent. This scheme is motivated by the observation that the radios turn-around both for data frames as well as for link-layer ACKs. The presence of traffic in both directions, even when caused by TCP ACKs, already causes turnarounds to happen. So, if link-layer ACKs were piggybacked with data frames, some extra radio turnarounds can be eliminated. Recent releases of the radio software in the Ricochet network attempt this optimization whenever possible.

However, despite this optimization the fundamental problem of additional traffic and the underlying MAC protocols affecting round-trip time estimates and causing variabilities in performance still persists. Connections traversing multiple wireless hops are especially vulnerable to this effect, because it is now more likely that the radio units may already be engaged in conversation with other peers. Figure 6.5 shows this by plotting the round-trip samples and TCP estimates of the smoothed round-trip estimate and mean linear deviation during a bulk 1 MB TCP transfer in single-hop wireless network with this optimization (the best-case topology for the optimization). The average smoothed estimate maintained by TCP is about 2.2 seconds and the linear deviation is about 1 second, still rather large.

Our simulations and results (Section 6.6) of the multi-hop wireless network assume that the radio units perform this optimization and piggyback link-layer ACKs with data whenever possible. We

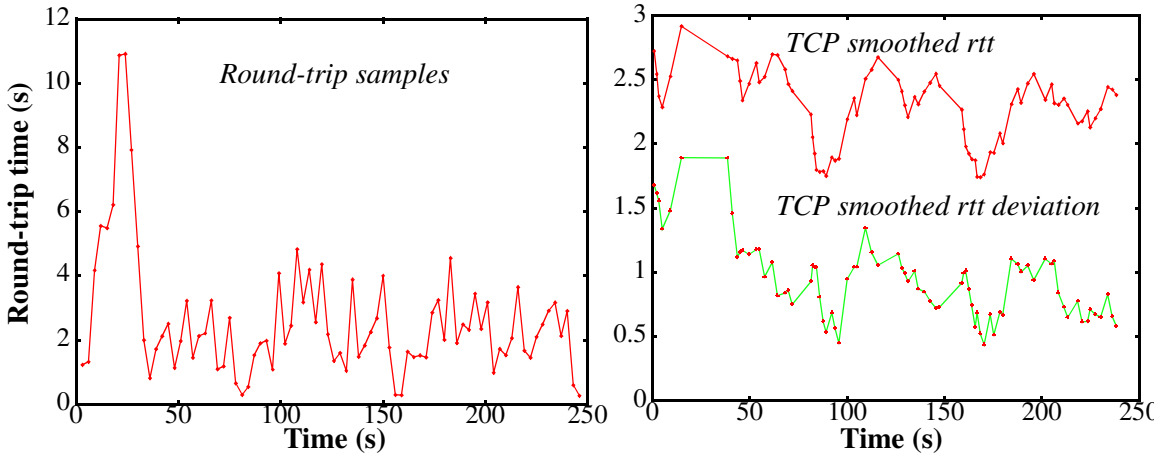

Figure 6.5 Round-trip time estimate for a 1 MB bulk transfer in a one-hop Ricochet wireless network using an optimized link-layer protocol that piggybacks link-layer ACKs with data. The figure on the left shows the raw round-trip samples, while the figure on the right shows the TCP smoothed round-trip estimate and mean linear deviation using the EWMA filters discussed in Section 2.2.2. The fundamental problems with large round-trip time variations still persist despite the piggybacking optimization.

also note that the basic reliable link-layer protocols in many packet radio networks do not piggyback ACKs with data.

6.4 Solutions

Our solutions to the problems described in the previous two sections are inspired by the following observation. In all these networks, regardless of technology, the root problems arise because of imperfections, slowness and variability in the ACK feedback. This immediately suggests the first prong of our two-pronged solution: *reduce the frequency of ACKs* because of the unfavorable interactions that arise when there are too many ACKs in the network. However, TCP does not handle infrequent ACKs well, as explained in Section 6.3.2. This motivates the second prong of our solution: devise techniques to *handle infrequent ACK feedback.* Together, these techniques lead to significant improvements in performance over different asymmetric networks.

We use a link-layer technique called *ACK filtering* to reduce ACK frequency. To handle infrequent ACKs, we develop two techniques: *sender adaptation,* an end-to-end scheme and *ACK reconstruction,* a link-layer scheme. In our experiments, we also evaluate the performance of ACK congestion control (ACC), described in Section 2.6.4.

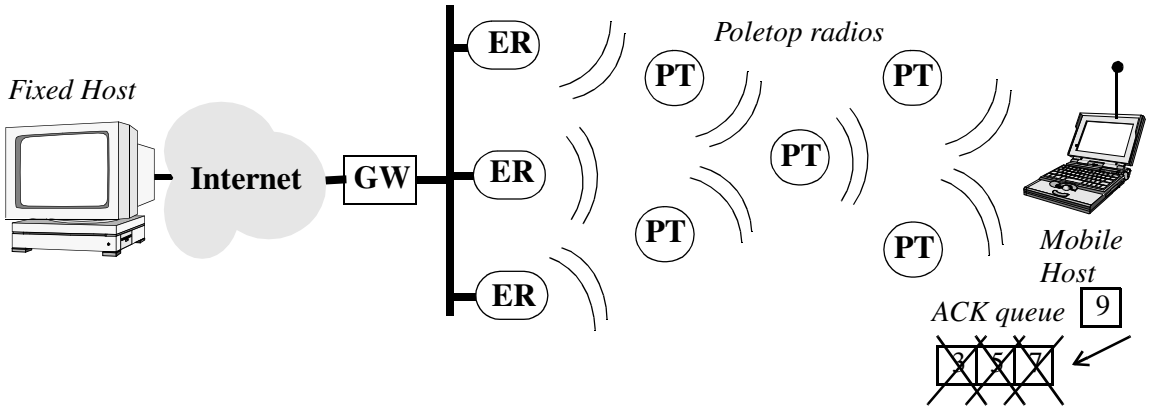

Figure 6.6 ACK filtering in the packet radio network. Both the MH's portable modem and the poletop radios implement AF. When cumulative ACK 9 is about to be enqueued, previously enqueued ACKs 3, 5 and 7 are purged without affecting the semantics of ACKs.

6.4.1 ACK Filtering (AF)

AF, shown in Figure 6.6, is a TCP-aware link-layer technique that reduces the number of TCP ACKs sent on the reverse channel. The challenge is to ensure that the sender does not stall waiting for ACKs, which can happen if ACKs are indiscriminately removed on the reverse path. AF removes only certain ACKs without starving the sender by taking advantage of the fact that TCP ACKs are cumulative. As far as the sender's error control mechanism is concerned, the information contained in an ACK with a later sequence number subsumes the information contained in any earlier ACK.

When an ACK from the receiver is about to be enqueued at a reverse direction router, the router or the end-host's link layer (if the host is directly connected to the constrained link) traverses its queues to check if any previous ACKs belonging to the same connection are already in the queue. If so, it removes them from the queue, reducing the number of ACKs that go back to the sender. The removal of these "redundant" ACKs frees up space for other data and ACK packets. In particular, it ensures that later ACKs do not get queued behind earlier ones waiting to be sent on the reverse path.

The policy that the filter uses to drop packets is configurable and can either be deterministic or random (similar to a random-drop gateway, but taking the semantics of the items in the queue into consideration). There is no need for any per-connection state to be maintained at the router—all

the information necessary to implement the drop policy is implicitly present in the packets in the queue.

In the experiments reported in this chapter, AF deterministically purges all preceding ACKs belonging to a connection whenever a new ACK for the same connection with a larger cumulative value enters the queue.

6.4.2 TCP Sender Adaptation (SA)

AF alleviates the problem of congestion on the reverse path by decreasing the frequency of ACKs, with each ACK potentially acknowledging several data packets. However, this is only one part of the complete solution: unilaterally reducing ACK frequency, while reducing the level of contention along the reverse path, actually leads to worse performance. In particular, as explained in Section 6.2, it causes three problems: the propensity of the sender to transmit large bursts of data, a slowdown in window growth, and reduced effectiveness of the fast retransmission algorithm. SA addresses these problems by modifying the TCP sender algorithm and making it adapt to infrequent ACKs.

First, it reduces the sender bursts by placing an upper bound on the number of packets the sender can transmit back-to-back, even if the window allows the transmission of more data. If necessary, it schedules transmissions of data for later points in time based on its estimate of the connection's data rate. It estimates the data rate as the ratio *win*/*srtt*, where *win* is the current TCP window size and *srtt* is the smoothed RTT estimate. Thus, large bursts of data are broken up into smaller bursts spread out evenly over time.

Second, SA takes into account the amount of data acknowledged by each ACK, rather than the number of ACKs while incrementing the congestion window, *cwnd*. So, if an ACK acknowledges *s* segments, the window grows as if *s* separate ACKs had arrived. This solves the window growth problem.

Third, an SA-enhanced sender does not just count the number of duplicate ACKs, to account for their removal by an ACK filter. With AF, when the reverse channel router filters some of the enqueued duplicate ACKs, sets a field in the TCP header of the remaining ACK informing the

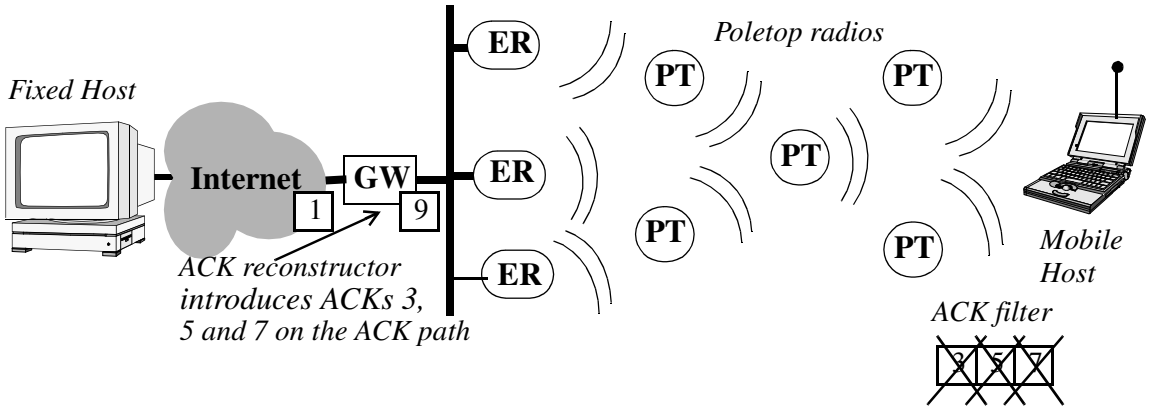

Figure 6.7 ACK reconstruction in the packet radio network, implemented at the Gateway node. When ACK 9 arrives, the previous ACK seen was 1. Rather than forward 9 at once, AR generates ACKs 3, 5 and 7 and sends them to the sender before forwarding 9.

sender of the number of purged duplicates. This helps the sender in the fast retransmission and recovery phases.

SA is generally applicable when the ACK path is loss-prone or if ACC (Section 2.6.4) is deployed in the system.

6.4.3 ACK Reconstruction (AR)

AR is a TCP-aware link-layer technique that reconstructs a sporadic ACK stream after it has traversed the reverse bottleneck. It prevents the reduced ACK frequency from adversely affecting the performance of standard TCP sender implementations on the Internet (i.e., those that do not implement SA). This enables us to use schemes such as AF (and ACC) without modifying TCP senders on the Internet. This solution can be easily deployed by Internet Service Providers (ISPs) of asymmetric access technologies in conjunction with AF, to achieve good performance without changing any TCP implementations. It is schematically shown for a packet radio network in Figure 6.7.

AR deploys a soft-state agent called the ACK reconstructor at the other end of the constrained ACK bottleneck. For example, it may be deployed at the gateway (GW in Figure 6.7) to the packet radio cloud. The reconstructor carefully fills in the gaps in the ACK sequence and introduces ACKs to smooth out the ACK stream seen by the sender. However, it does so without violating the end-to-end semantics of TCP ACKs, as explained below.

Suppose two ACKs, *a1* and *a2,* arrive at the reconstructor after traversing the constrained reverse link at times *t1* and *t2* respectively. Let $a2\text{-}a1 = \Delta a > 1$. If *a2* were to reach the sender soon after *a1* with no intervening ACKs, then at least Δa segments are sent by the sender in one burst (if the flow control window is large enough) and the congestion window increases by at most one, independent of Δa. AR remedies this problematic situation by interspersing ACKs to provide the sender with a larger number of ACKs at a consistent rate, which reduces bursts and allows the congestion window to increase faster.

How is this done? One of the configurable parameters of the reconstructor is δa, the ACK threshold, which determines the spacing between interspersed ACKs. Typically, we set δa to 2, which follows TCP's standard delayed-ACK policy. Thus, when successive ACKs separated by Δa arrive at the reconstructor, it interposes $ceil(\Delta a/\delta a) - 2$ ACKs, where *ceil()* is the ceiling operator. The other parameter needed by the reconstructor is δt, which determines the *temporal* spacing between the reconstructed ACKs. To do this, the reconstructor measures the rate at which ACKs arrive at its input. This rate depends on the output rate from the constrained reverse channel and on the presence of other traffic on that link. We use an exponentially weighted moving average estimator to monitor this rate; the output of the estimator is Δt, the average temporal spacing at which ACKs are arriving at the reconstructor (and the average rate at which ACKs would reach the sender if there were no further losses or delays).

If the reconstructor sets δt equal to Δt, then the connection would operate at a rate governed by the reverse bottleneck link, and the resulting performance would be determined by the rate at which *unfiltered* ACKs arrive out of the reverse bottleneck link. If sender adaptation were being done, then the sender behaves as if the rate at which ACKs arrive us $\Delta a/\Delta t$. Therefore, a good method of deciding the temporal spacing of reconstructed ACKs, δt, is to equate the rates at which *increments* in the ACK sequence happen in the two cases. That is, the reconstructor sets δt such that $\Delta a/\Delta t = \delta a/\delta t$, which implies that $\delta t = (\delta a/\Delta a)*\Delta t$. Therefore, the latest ACK in current sequence, *a2*, is held back for a time roughly equal to Δt, and $ceil(\Delta a/\delta a) - 2$ ACKs are evenly interposed in this time.

Thus, by carefully controlling the number and spacing between ACKs, standard TCP senders can be made to increase their congestion window at the right rate and avoid burst transmissions. AR can be implemented by maintaining only soft state at the reconstructor that can easily be regener-

ated. Note that no spurious ACKs are generated by the reconstructor and the end-to-end semantics of the ACKs are fully preserved. The trade-off in AR is between obtaining smoother data transmissions, a better rate of congestion window increase and a reduction in the round-trip variation, versus a modest increase in the round-trip time estimate at the sender. We believe that this is a good trade-off in the asymmetric environments we are concerned with.

6.5 Implementation

In this section, we discuss the implementation details of AF and SA in the BSD/OS 3.0 system.

6.5.1 ACK Filtering

The goal of AF is to remove all redundant ACKs from the queue of the reverse channel when a newer cumulative ACK arrives there. We describe below our experiences with implementing this scheme, and also discuss some practical problems in the context of the packet radio network modems.

Since both the reverse connection for the bandwidth asymmetry case and the network connectivity for the latency asymmetry case use Hayes-compatible modems running the Point-to-Point Protocol (PPP) [60], we implemented this scheme in the PPP layer of the BSD/OS 3.0 kernel. When a new ACK arrives, we traverse the queue of packets and remove all redundant ACKs for the connection, taking care not to remove any ACKs that have data associated with them. A connection is uniquely identified by the 4-tuple `<source address, destination address, source port, destination port>`.

In practice, packets on the PPP queue are often header-compressed using the algorithm described in [60], which makes it hard to offset into the header and obtain the necessary fields. There are several alternative approaches to solving this problem:

1. Decompress the compressed packets in the queue each time a new ACK arrives and compare fields. This requires maintaining extensive decompression state at the compressor (the location of the ACK filter), which makes it expensive and hard to implement.

2. Explicitly maintain two queues at the PPP layer, one for header-compressed packets and the other for their uncompressed TCP/IP headers alone. Then, upon every enqueue/dequeue to the packet queue, the corresponding header must be enqueued/dequeued on the header queue as well. We chose not to adopt this method because dequeueing of packets can be done from a variety of underlying disciplines (not just PPP), and more importantly, because we would need to constantly maintain the consistency between the two queue data structures.
3. Modify the PPP queue data structure to maintain not just pointers to kernel `mbufs` [97] (corresponding to the header-compressed packets), but a pointer to a structure containing the relevant fields of the packet header as well as the kernel `mbufs`. This is the simplest efficient alternative and is the one we implemented.

ACKs are purged if their sequence numbers are smaller than the newly arrived ACK, or if they are the same and the new ACK advertises a different window from the enqueued one (since later window updates subsume earlier ones). Currently, we do not filter out duplicate ACKs that could trigger a fast retransmission at the sender. We could do so by adding a TCP option that specified the number of purged duplicate ACKs and propagating that to the sender from the router and by modifying the sender adaptation implementation (Section 6.5.2) to recognize and react to this option.

When we implemented this mechanism and deployed it, we found that the queueing of ACKs occurs inside the modem and not in the kernel's PPP queue, with both the Ricochet and standard telephone modems. Thus, the effectiveness of the implementation in the PPP layer of the kernel is limited. We need to modify the modem firmware to implement AF or control queue lengths on the modem and cause queueing to happen in the kernel's PPP queue—we could not do this because we had no access to the modem's internal firmware.

Fortunately, since we have implemented accurate simulation models of the wireless cable and packet radio networks, we could perform detailed simulation experiments and obtain insights into the reasons for performance improvement with AF in these networks.

6.5.2 Sender Adaptation

The goal of SA is to enable the TCP sender to operate successfully even when the frequency of ACKs is reduced by AF or ACC. There are two aspects to sender adaptation:

1. Incrementing the congestion window based on the amount of data acknowledged, rather than on the count of ACKs.
2. Avoiding large bursts of data when each ACK acknowledges several data segments.

Our current implementation does not include the fast retransmission solution described in Section 6.4.2.

To satisfy the first requirement mentioned above, the sender computes `acked`, the number of bytes newly acknowledged by the most recent ACK. The quantity `acked` is scaled down by twice the maximum segment size, `2*mss`, to determine the number of window increase operations to be performed (either in the slow start phase or in the linear increase phase). Scaling down by `2*mss` makes it equivalent to TCP Reno with standard delayed ACKs, which involves one window increase operation for every two acknowledged data segments [20].

To avoid large bursts of data, the sender limits the maximum number of segment that can be transmitted in a single burst to a configurable parameter called `maxburst`. If more segments remain to be sent, a software timer is scheduled for a later time. The wait time for the timer is computed by dividing `maxburst` by the ratio of the current window size to the round trip time. The other segments are transmitted clocked by this software timer, one segment at a time. We set `maxburst` to three in our implementation.

To compute the rate at which the software timer must fire, we need to measure the round trip time more accurately than is done using the timestamp option in standard TCP, which has a coarse 500 ms timer granularity. While a coarse granularity is acceptable for the retransmission timer, it leads to highly conservative behavior when used to rate control transmissions. We solve this problem by having the sender insert the current time that has a microsecond granularity in the timestamp option [61], which is then echoed by the receiver. Thus, every time an ACK arrives, we know the accurate time at which it was originally sent and can calculate an accurate difference to update `t_exact_srtt`, the "exact" round-trip time estimate. This is a 4-byte option; we do not run into wrap-around problems as long as the RTT is under 4000 seconds (many other things would break in TCP before this if the true round-trip were indeed that large!). The quantity `t_exact_srtt` thus computed is used in the rate calculation described above.

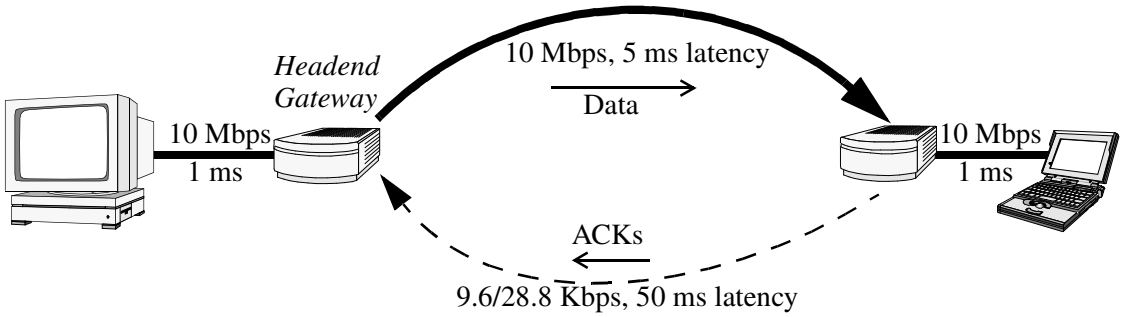

Figure 6.8 The simulation topology used to model a network with bandwidth asymmetry. The bandwidth and delay parameters have been chosen to closely model the Hybrid wireless cable modem network.

6.6 Simulation Results

In this section, we present and discuss the results of detailed *ns* simulations of bandwidth and latency asymmetry, modeled after the Hybrid wireless cable and Ricochet packet radio networks respectively.

6.6.1 Bandwidth Asymmetry

We first present the results of several simulations of unidirectional TCP transfers on a network that exhibits bandwidth asymmetry and discuss the reasons for observed performance.

The simulation topology we used to investigate the effects of bandwidth asymmetry is shown in Figure 6.8. The parameters of the forward channel are based on our measurements of the Hybrid wireless cable network. We experimented with reverse channels of different bandwidths, but fixed delay (50 ms). Although reverse channels ranging from slow to high speed dialup lines to ISDN have different latencies in reality, keeping the delay constant in the simulation experiments helps us focus on the bandwidth asymmetry alone.

We conducted two sets of experiments, each involving a 50-second transfer in the forward direction. There was no traffic in the reverse direction other than the ACKs for the forward transfer in both cases. In the first set of experiments (A), there was sufficient buffering to ensure that no data losses occurred in the forward direction in most cases. The second set of experiments (B) was designed to investigate how the different protocols performed when losses occurred in the forward

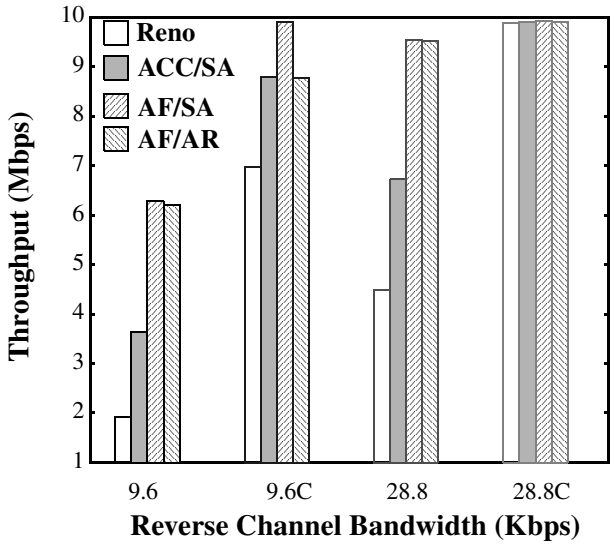

Figure 6.9 Throughputs from the simulation of a single one-way transfer with no data losses (experiment (A)). "C" indicates the use of SLIP header compression.

direction due to insufficient forward buffer capacity, and in particular to highlight the benefits of using SA or AR on a loss-prone forward path when the ACK stream is prone to losses.

6.6.1.1 Loss-free transfers

Figure 6.9 shows the throughputs obtained for four protocols—TCP Reno, Reno with AF/SA (i.e., AF with SA at the sender), Reno with AF/AR and Reno with ACC/SA—with different types of return channels with no data losses in the forward direction. The socket buffer size at the sender and receiver was set to 100 KB and each data packet was 1 KB in size. The buffer size at the reverse bottleneck router was set to 10 packets and at all other routers to 20 packets. The ACK size was set to 6 bytes and 40 bytes to model situations with and without header compression. The motivation is to understand how the bandwidth and queue length of the return channel affect forward throughput when there is no data loss.

Since the transfers are long, the reverse buffer fills early for TCP Reno. Beyond that point, only one ACK in k gets through on average, causing the sender to transmit bursts of k packets on average. As long as k does not exceed the bottleneck buffer size in the forward direction, burst transmissions from the sender do not lead to losses.

The normalized bandwidth ratio (Section 2.6.2 & Section 6.2.1) exceeds 20 for the cases of SLIP without header compression, resulting in bursts larger than the size of the buffers in the forward

direction. This explains the poor throughput of TCP Reno in those cases (1.93 and 4.48 Mbps) because losses do indeed occur on the forward path. SA or AR used in conjunction with AF breaks up potential bursts, thereby avoiding performance degradation suffered by TCP Reno.

For the 9.6 Kbps reverse channel with header compression, k is 6.25, which is less than 20. Still, the throughput obtained with TCP Reno (6.67 Mbps) is worse than that for the other schemes. This happens because the reverse channel buffer gets filled with ACKs (totalling 10*6 = 60 bytes), which adds a significant delay (60*8/9.6 = 50 ms) to the connections round-trip time (RTT). The same effect also explains why the performance with ACC is somewhat worse than that with AF for both the 9.6 Kbps and 28.8 Kbps uncompressed SLIP cases. The former only tries to ensure that the reverse channel queue does not get completely filled up. The latter ensures that there is not more than one ACK per connection in the queue, which minimizes the effect of queuing on the round-trip time.

The following are the key results from the set of experiments (A):

1. C-SLIP is a big gain, and in some cases (when there are no losses in the forward direction) it entirely eliminates the problems caused by bandwidth asymmetry.
2. AF and ACC lead to significant improvements when the reverse buffer is small, especially when k is large.
3. Large reverse buffers cause TCP Reno performance to severely degrade, and cause ACC to perform poorer than AF. This is because Reno now progresses at the rate at which ACKs leave the reverse queue, and ACC is benign until the number of reverse ACKs is a substantial fraction of the reverse queue.

6.6.1.2 Loss-prone transfers

We now discuss the results of the second set of experiments (B), designed to study the effects of forward losses on performance, and to investigate how Reno, AF and ACC alter the inter-ACK spacing and how SA and AR prevent burst transmissions. Table 6.1 shows the results of these experiments for the different protocols. The maximum window size was set to 120 packets (to ensure that the receiver window is not the performance bottleneck) and all queue sizes were set to 10 packets. We used a 28.8 Kbps header-compressed reverse channel.

Metric	Reno	ACC/SA	AF/SA	AF/AR	AF alone
Throughput (Mbps)	6.71	6.95	**7.82**	**8.57**	**5.16**
Average cwnd (pkts)	66.7	62	**65.3**	**104.6**	**43.8**
Average rtt (ms)	79	70	65	97	65

Table 6.1 Performance of different protocols in the presence of losses in the forward direction. The highlighted numbers show the benefits of SA and AR and demonstrate that AF alone is not enough.

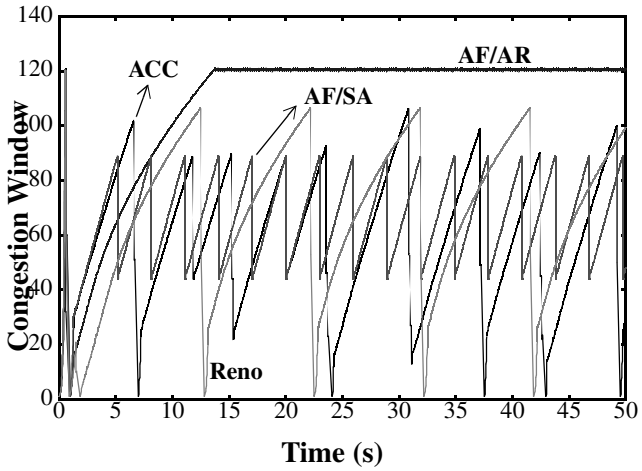

Figure 6.10 Congestion window evolution for the different schemes. AF/AR, AF/SA and ACC don't fluctuate as much as Reno, achieving better performance.

AF/AR and AF/SA perform the best, achieving throughputs between 15% and 21% better than Reno. The degree of burstiness reduces significantly and the reverse router queue is no longer perpetually full. This is shown in Figure 6.10, which depicts the time-evolution of congestion windows for the different protocols. Table 6.1 shows the time-averaged TCP congestion window and round-trip times for the different protocols. It is clear from these results that reducing the frequency of ACKs alone is not sufficient—AF alone performs even worse than Reno and techniques like SA or AR are essential to achieve good performance.

We note that AR results in a larger round-trip time than the other protocols, but this leads to the best performance for this setting of parameters because it reduces the number of losses (since packet transmissions from the source are now spread over a longer duration). For ACC/SA, we use a RED gateway to mark packets (ACKs) and drop packets when the queue is full (Section 2.6.4). We found that using a random drop policy is superior to dropping from the tail when an ACK has

to be dropped. This is because tail drops sometimes lead to long, consecutive sequences of ACKs being dropped, leading to increased sender burstiness.

In summary, our results show that SA or AR is important to overcome the burstiness that results from a loss-prone ACK stream.

6.6.2 Latency Asymmetry

Based on experimental measurements of the Ricochet network, we modeled the system in *ns* (Section 3.1). The simulation parameters used to obtain the results described in this section are shown in Table 6.2. We do not consider the impact of wireless bit errors in these simulations, to isolate the impact of variability due to the MAC protocol on performance. Link-layer retransmissions of corrupted packets only add to the latency variability of the network.

Parameter	**Value**
Link bandwidth	100 Kbps
Fixed link latency	10 ms
T_{TR}	11.125 ms
T_{RT}	13.25 ms
Radio queue size	10 packets
Receiver buffer size	32 KB

Table 6.2 Simulation parameters of the multi-hop packet radio network. The number of wireless hops varies between one and three.

Our first set of experiments are for single one-way transfers through a network including the wireless cloud, with the number of wireless hops varying between one and three, similar to the topology of the commercial Ricochet network. Then, we experiment with multiple simultaneous TCP transfers through the wireless cloud to investigate the issues of fairness and scale in this wireless network.

6.6.2.1 Single Bulk Transfers

The workload in these experiments consists of a 100-second TCP transfer, with no other competing traffic and a maximum receiver window size of 32 KB. The choice of long transfer time lets us investigate the steady state performance of the different schemes. Congestion losses occur as a result of buffer overflow and lead to sender timeouts if multiple packets are lost in a transmission window. We investigate the performance of TCP Reno, AF/SA and ACC/SA.

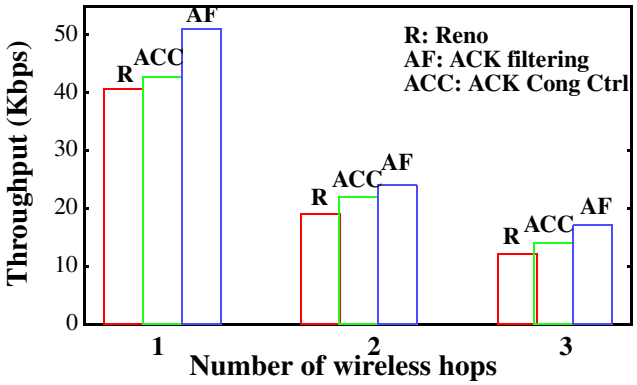

Figure 6.11 Simulated TCP throughputs of Reno, ACC and AF, as a function of the number of wireless hops.

Figure 6.11 shows the results of these experiments, as a function of the number of wireless hops. The performance of AF and ACC with SA are better than Reno, and AF/SA is better than ACC/SA. The performance improvement for AF/SA over Reno is shown in Table 6.3—the improvement in throughput varies from 25% (one wireless hop) to 41% (three wireless hops). Similar improvements are observed when AR is used at the gateway to the wireless cloud instead of SA at the sender.

# hops	Reno (Kbps)	ACC/SA (Kbps (%))	AF/SA (Kbps (%))
1	40.7	42.8 (5%)	51.0 (25%)
2	19.0	22.0 (16%)	24.0 (26%)
3	12.1	14.5 (20%)	17.1 (41%)

Table 6.3 Simulated performance of Reno, ACC/SA and AF/SA as a function of the number of wireless hops in the multihop wireless network. Throughputs are in Kbps; the numbers in parentheses are percentage improvements compared to Reno.

The main reasons for this improvement are the reduced number of packets and reduced round-trip variability of the two enhanced protocols compared to Reno. Figure 6.12 shows the round-trip times of simulations of a TCP Reno connection and a TCP connection with AF, over two wireless hops in a chain-like topology between sender and receiver. For Reno, the mean round-trip time is about 2.67 seconds and the standard deviation is about 1 second. AF reduces the mean round-trip time to 1.85 seconds and the corresponding standard deviation to only 0.6 seconds. The number of packets traversing each node also drops, reducing the amount of contention in the network. These factors lead to a 26% improvement in end-to-end throughput, from 19 Kbps (Reno) to 24 Kbps

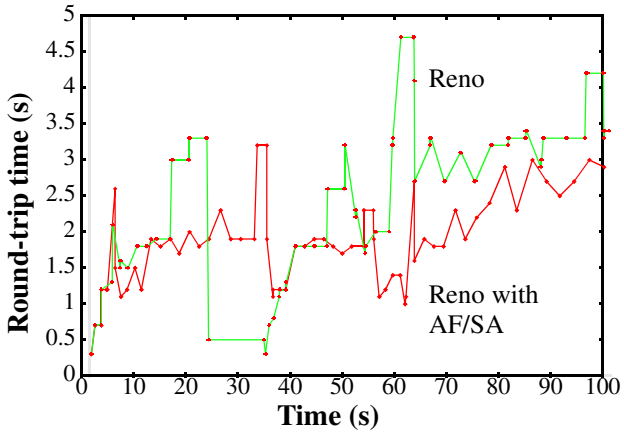

Figure 6.12 Round-trip times obtained from simulations of TCP Reno and TCP with AF/SA for a connection over two wireless hops. The round-trip times for the Reno connection are much more variable, with a mean of 2.67 s and a standard deviation of 1 s, whereas AF reduces the mean to 1.85 s and the standard deviation to 0.6 s.

over two wireless hops. Similar improvement in performance is seen for connections traversing three wireless hops: end-to-end throughput improves on average from 12.1 Kbps (Reno) to 17.0 Kbps (AF), an improvement of 41%. While the exact values of the round-trip time and the deviation are a function of the window size and the amount of competing traffic, similar improvements are observed in other configurations as well.

Finally, we note that AF outperforms ACC because the former completely eliminates all redundant ACKs and reduces the amount of interfering traffic caused by TCP ACKs to a larger extent than ACC does.

6.6.2.2 Multiple Simultaneous Transfers

We now experiment with multiple simultaneous transfers in the multihop wireless network to understand performance as a function of the number of connections traversing the network. These concurrent connections compete for resources in the network. We focus on simulations of connections over one wireless hop in this section for Reno+AF/SA, the best performing protocol, and compare it to Reno.

There are two important metrics to consider while studying the impact of increasing the number of connections in the network: *utilization* and *fairness,* as defined in Section 3.3. We are interested in

obtaining large values of both the network utilization as well as the fairness index for any configuration.

# simultaneous connections	Reno throughput (Kbps) [std-dev]	Reno+AF/SA (Kbps) [std-dev]
1	40.7 [-]	51.0 [-]
2	42.8 [13.1]	51.8 [10.0]
4	45.2 [24.9]	49.3 [8.9]
6	47.1 [32.5]	49.3 [20.2]
8	45.0 [37.6]	49.6 [28.0]
10	45.6 [42.2]	48.4 [32.4]
12	45.8 [48.0]	48.8 [36.4]

Table 6.4 Throughputs of TCP Reno and Reno with AF/SA as a function of the number of connections over one wireless hop. The numbers in brackets are the standard deviations of the throughputs calculated over the simultaneous connections.

Table 6.4 shows the aggregate throughput achieved by all the connections in the network as a function of the number of competing connections for Reno and AF/SA. The table also shows the standard deviation of the resulting performance, which is a measure of the degree of variation in the throughputs observed by the concurrent connections. The aggregate performance varies between 44 and 47 Kbps for TCP Reno and between 48 and 52 Kbps with AF, as the number of connections varies, implying that there is little change in utilization as a function of the number of connections for both protocols. However, the standard deviation of the throughputs of the concurrent connections is much higher for Reno than for AF/SA. The distribution of throughputs is more spread out and less equal across any set of connections, especially as the offered load on the network scales to larger numbers of connections.

We quantify this effect in Figure 6.13, where the fairness index (Section 3.3) of throughput is plotted as a function of the number of concurrent connections, n. TCP Reno is often grossly unfair in the distribution of throughput, reaching a value as low as 0.49 (n = 12) and not exceeding 0.85 (n = 2). In contrast, AF substantially improves the fairness index of the throughput distribution by over 25%, while also improving the overall utilization of the network. The reasons for the improvements in fairness are the reduced number of interfering and competing packets in the network. The duration of TCP timeouts are smaller because of the reduced variation in round-trip times. Another

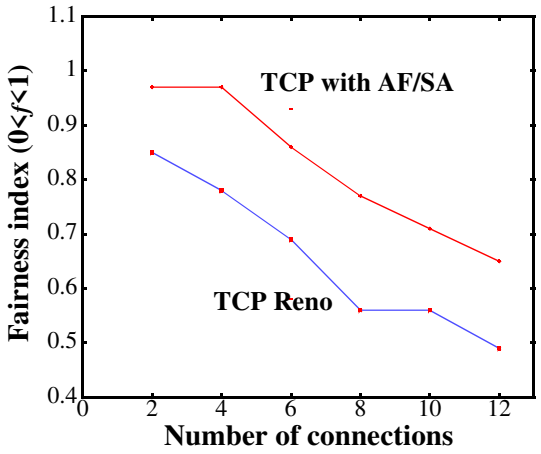

Figure 6.13 Simulation results for the fairness index of TCP Reno and TCP with AF, as a function of the number of connections traversing one hop of the packet radio network.

way of interpreting this result is that the performance of each TCP connection becomes more predictable.

Thus, we see that our enhancements to TCP and the link-layer algorithms help improve overall utilization and fairness as the number of connections increases. In the next section, we investigate some scaling issues by simulating a Web-like workload in a network with a high-bandwidth forward channel and a packet radio return channel.

6.6.3 Combining Bandwidth and Latency Asymmetry

In this section, we investigate the effects of combining different types of asymmetry on TCP performance. We focus on a network topology with a high-bandwidth forward path modeled after Hybrid's wireless cable channel, and a low-bandwidth, packet radio reverse path modeled after the Ricochet network. Such composite network topologies are important in several application scenarios. For example, a disaster relief vehicle or ambulance with a unidirectional high-bandwidth link would use a wide-area wireless network as its reverse channel.

Figure 6.14 shows the measured performance of 1 MB TCP transfers as a function of the receiver socket buffer size using a Hybrid Networks' wireless cable forward path and a one-hop wireless Ricochet return path. The error-bars show the standard deviation of measured performance (20 runs each). These controlled measurements were performed in the absence of any cross-traffic; the inherent variability of the return path manifests itself in the significant error-bars on the graph.

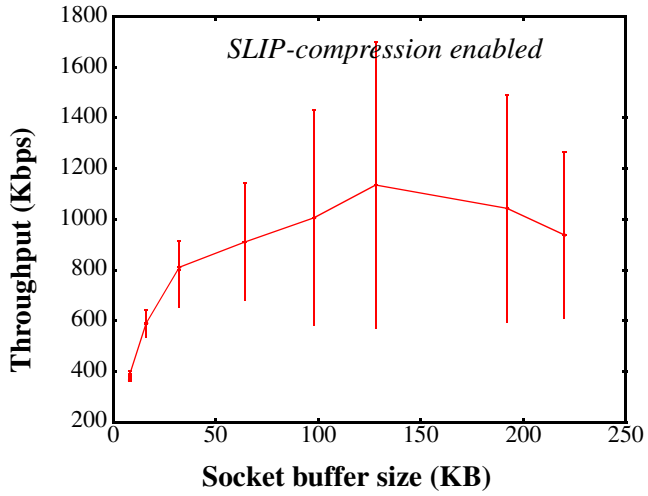

Figure 6.14 Measured performance of a 1 MB TCP transfer across a Hybrid™ wireless cable downlink and Ricochet return channel. The large errorbars (standard deviations) are a consequence of the high variability of the Ricochet return channel.

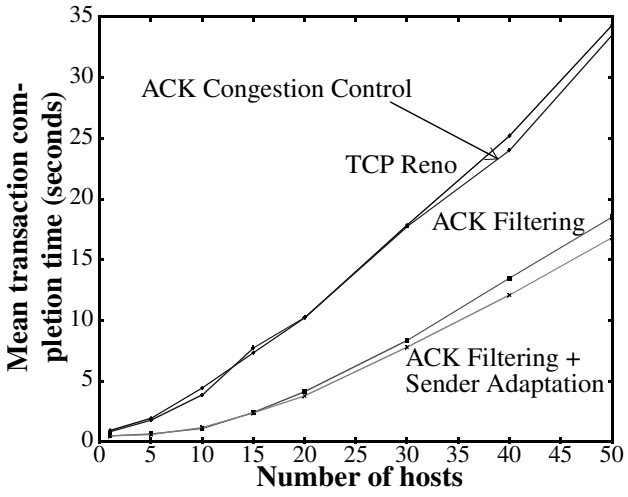

Figure 6.15 Simulated scaling behavior of *TCP Reno*, *AF*, and *ACC* as a function of the number of connections for a Web-like micro-benchmark over a packet radio return path. The graph shows average completion time vs. *n*, the number of hosts (4 concurrent connections per host). AF performs the best as *n* increases.

We focus on a Web-like benchmark in the following simulations and study the performance of this network as the number of hosts and connections increases. We investigate the various modifications to the transport and link-layer protocols to reduce the average completion time of a Web request in such networks.

We model the Web micro-benchmark as a 500-byte Web request followed by set of four concurrent TCP transfers of 10 KB each to the client. This is not intended to be an accurate model of reality, but to understand the effects of small and concurrent connections, as well as competing and interacting users, on performance. Since latency is a critical factor that affects performance in the packet radio network, we implicitly assume that the Web requests use pipelining [108], with one 500-byte request resulting in four downloads. We vary the number of hosts from 1 to 50 in the simulations. Hosts make requests independent of each other at a time uniformly distributed in [0,5] seconds. We measure the mean completion time for the entire Web transaction (i.e., all four concurrent connections).

Figure 6.15 shows the mean completion time for a transaction as the number of hosts varies for TCP Reno, Reno with AF and Reno with ACC. Two curves are shown for the AF case: with sender adaptation enabled and without.

These results indicate that AF significantly reduces the response time and improves the throughput of the network as the system scales. The reason ACC is not particularly beneficial here is the shorter transfer lengths in this workload compared to the bulk transfer experiments. No connection exceeds 10 packets in length, so the sender's window is never large. This limits the extent to which ACC can improve performance because it takes a longer duration to adapt to the state of congestion on the reverse path. The other reason is the larger number of ACKs traversing the network with ACC compared to AF. A critical factor in this network is the latency and variability associated with each packet on the wireless network. AF is a significant benefit because it purges *all* redundant ACKs from the queue independent of the state of congestion, thus reducing the number of packets traversing the wireless cloud.

The lack of sender adaptation does not significantly impair performance, since each connection is rather small. The maximum possible burst in the network is automatically limited by this short length and the slow start process starting from 1 segment. Finally, we note that with persistent-connection HTTP [108], recommended by HTTP/1.1 [44], connections will tend to be longer. This is likely to help ACC perform better than it currently does for this workload.

6.7 Summary

In this chapter, we discussed the adverse effects of asymmetry on TCP performance. These problems arise in several wireless networks, including bandwidth-asymmetric networks and packet radio networks for different underlying reasons. However, the net effect on TCP performance is the same in both cases: performance degrades significantly because of imperfections and variabilities in the ACK feedback from the receiver to the sender. Our solutions to these problems use a combination of link-layer and end-to-end algorithms in a two-pronged approach: we devise techniques to reduce the frequency of TCP ACKs, and develop techniques to handle this reduced ACK frequency. Our detailed simulations show that these techniques significantly improve end-to-end performance in both bandwidth-asymmetric wireless cable and latency-asymmetric packet radio networks. ACK filtering combined with TCP sender adaptation improves the scaling properties of a network that includes both bandwidth and latency asymmetry.

It has long been an axiom of mine that the little things are infinitely the most important.

— Sherlock Holmes, "A Case of Identity," by Sir Arthur Conan Doyle

Chapter 7

Enhanced Loss Recovery for Small Windows

In previous chapters, we addressed the problems caused by wireless bit-errors and asymmetric effects on TCP performance and solved them using a combination of link-layer and end-to-end techniques. In this chapter, we analyze the problems that arise due to small TCP transmission windows and solve them by enhancing TCP's data-driven loss recovery mechanism. Small transmission windows are the norm in many wireless networks that have inherently low channel bandwidths. Because the bandwidth-delay product of the connection is a measure of its sustainable transmission window, low channel bandwidths imply small windows. In addition, short transfers typical of today's Web workload add to the prevalence of small windows.

The result of small TCP windows is that data-driven loss recovery using fast retransmissions is not triggered often; most losses require expensive timeouts to recover from, keeping the connection idle for long periods of time. Our goal is to incur expensive timeouts only when persistent congestion occurs and not when transient losses occur in the network. By designing enhancements to TCP's loss recovery algorithms to make it more data-driven, we reduce the reliance on conservative retransmission timers.

The rest of this chapter is organized as follows. We start by analyzing packet-level traces of over 1.6 million TCP connections collected from the IBM-run official Web server of the 1996 Atlanta Summer Olympic Games [104] to understand the problems caused by small windows. Then, after discussing the limitations of TCP selective acknowledgments (SACK) [92] in solving these problems, we present the details of our loss recovery enhancements to TCP. We conclude the chapter by discussing simulation and implementation performance over wireless links and cross-country Internet transfers.

7.1 Analysis Overview

To understand the behavior of TCP connections in today's Internet, we analyze packet-level traces of connections to a busy Web server. In particular, we seek to answer the following questions:

1. *Loss recovery:* Do current TCP senders recover from packet losses using data-driven loss recovery mechanisms such as fast retransmission and fast recovery or do they tend to incur coarse timeouts when losses occur?
2. *Selective Acknowledgments (SACKs):* What are the potential benefits of using more sophisticated loss recovery mechanisms such as SACKs? How beneficial are SACKs likely to be in reducing the frequency of timeouts?
3. *Receiver bottleneck:* How often do receiver-advertised windows, used to perform flow control, restrict the network performance of a connection? Are the effective window sizes available to senders often limited by small receiver-advertised window sizes?

To answer these questions, we analyze a very large data set—packet-level traces of over 1.6 million TCP connections collected from the official Web server for the 1996 Atlanta Olympic games. Over a three-week period, the traces included approximately 1.5 billion TCP packets from 700,000 distinct hosts from all over the world.

Our analysis of the traces yield the following answers to the questions raised above:

1. *Existing loss recovery techniques are not effective in recovering from packet losses in a timely manner.* We find that over 50% of all losses are recovered only after a coarse timeout; fast retransmissions recover from less than 45% of all losses. This implies that better solutions are needed to perform efficient loss recovery.

2. *The deployment of SACKs is not likely to significantly reduce the occurrence of timeouts.* In these traces, only a little over 4% of the observed timeouts could have been avoided by the presence of SACKs.

3. *Receiver socket buffer sizes are sometimes too small, making the connection throughput limited by the receiver-advertised window.* Future network implementations should increase their default socket buffer size to avoid the receiver window from becoming a bottleneck on typical Internet paths. The socket buffer size limited the throughput of approximately 14% of all observed connections.

Table 7.1 summarizes the results of our analysis (details in Section 7.4).

Trace Statistic	**Value**	**%**
Total connections	*1,650,103*	*100*
With packet reordering	97,036	6
With receiver window as bottleneck	233,906	14
Total packets	*7,821,638*	*100*
During slow-start	6,662,050	85
# of slow-start pkts lost	354,566	5
During congestion avoidance	1,159,588	15
# of congestion avoidance pkts lost	82,181	7
Total retransmissions	*857,142*	*100*
Fast retransmissions	375,306	44
Slow-start retransmissions	59,811	7
Coarse timeouts retransmissions	422,025	49
Avoidable with SACKs	18,713	4
Avoidable with enhanced recovery	104,287	25

Table 7.1 Summary of analysis (percentages are relative to the category above it).

7.2 Data Collection and Processing

This section describes the details of the trace collection and post-processing we performed on the packet traces. We start with a brief description of the Web server location and configuration.

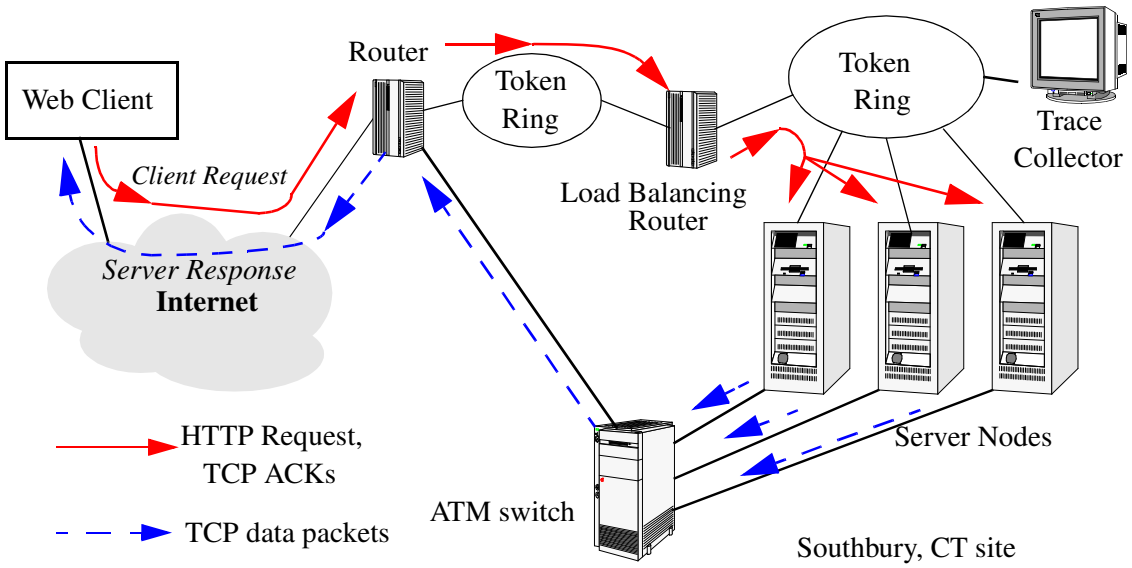

Figure 7.1 WWW server and monitoring setup.

7.2.1 Web Site

The Web traces used in this chapter come from the primary Web site (in Southbury, CT) that IBM ran for the Atlanta Olympic games [104]. Figure 7.1 shows the topology of the Web site's network and the hardware used at the site. During the Olympics, the site was connected via a T3 link to each of the four U.S. Network Access Points (NAPs), located in Chicago (Bellcore and Ameritech), the San Francisco Bay Area (Bellcore and Pacific Bell), New York (Sprint), and Washington, D.C. (MFS Datanet). In addition, there were mirror sites at Cornell (N.Y.), Keio (Japan), Karlsruhe (Germany) and Hursley (U.K.), for which we did not collect data. More details about the site structure and software are available from [147].

Requests for data from any client arrive at the Web site after being routed through the appropriate Internet NAP. These requests are passed to a load-balancing connection router that distributes them across several server nodes [3, 147]. These nodes retrieve the appropriate Web objects and transmit them across an internal ATM network and through the Internet to the clients.

7.2.2 Methodology

Stemm and Seshan monitored all the traffic coming into the site and obtained packet-level traces of this traffic by running the `tcpdump` [94] utility on a machine placed on the token ring connecting

the load-balancing router to the server nodes (Figure 7.1). This machine was an IBM 150 Mhz Pentium Pro PC running BSD/OS 2.1 from BSDI [23]. They extracted and stored the first 350 bytes of every packet destined to TCP port 80 (the HTTP port), compressing this data on-the-fly using the `gzip` utility. In addition, the server also transmitted other data (such as streaming audio) on other ports. They did not capture this data due to limited storage space of the setup and the high data packet capture rates. Even while restricting packet capture to traffic on port 80, they collected approximately 1 GB of compressed packet headers per hour during peak periods. Periodically, the packet trace files were dumped onto tape. While dumping the trace files to tape, however, there were occasional periods during which the traffic was not monitored.

Due to the location of the trace collection machine and the asymmetric path of request and response data, only packets coming from the clients *into* the Web server complex were captured. These packets include the initial TCP SYN packet from the client, the HTTP request packets, as well as all TCP ACKs for data sent from the server. In particular, this implies that no data packets from the server to the client were captured. However, this does not prevent us from determining the packet sequence trace as a function of time, since we can estimate this from the sequence of TCP ACKs. In addition, we modified the TCP stack on a subset of the servers. These modified server nodes transmitted the TCP/IP header, together with information on the current sender congestion window size, smoothed round-trip time estimate, and number of duplicate ACKs causing the retransmission of all retransmitted packets on the token ring interface to the trace collection machine. This retransmission information combined with the ACK traces and the knowledge of the sender's TCP algorithms at the server allow us to reconstruct the outgoing data stream with high accuracy.

Trace Statistic	Value
Packets captured	1,540,312,422
Packets dropped	7,677,854
Average packet drop percentage	0.498%
Distinct client addresses	721,417
Total Bytes Collected	~189 GB

Table 7.2 Raw trace statistics.

Table 7.2 describes the raw statistics about the trace data. More details about these traces and a study of the stability of wide-area network performance based on them can be found in [17, 15].

7.3 TCP Emulation Engine

We post-processed the packet traces to generate a one-record summary of the progress of each TCP connection, including the times and sequence numbers of ACKs as well as the server-side retransmission notifications. We then organized the connection summaries into a database that allowed us to cluster connections from a single client and compare the TCP performance of different clients at different times.

To faithfully reproduce the state of the sender-side TCP implementation, we wrote a *TCP emulation engine* that took the captured ACKs as input and reproduced the evolution of the sender-side state variables (e.g., congestion window, round-trip time estimate, maximum packet sequence transmitted, etc.). The captured ACKs are sufficient for this because the change in any sender-side TCP state variable is triggered by the reception of an ACK or the expiration of a timer.

In essence, the engine is an implementation of a subset of the functions of a TCP sender at user-level, driven by a sequence of ACK traces. If the evolution of the sender's state were completely driven by the arrival of ACKs (as is usually the case if a connection experiences no losses), it is possible to reconstruct it with complete accuracy. We handle the complications due to timer events by using a number of heuristics, in addition to estimating the round-trip time by faithfully following the implementation on the Web server's TCP stack.

We validated our engine by comparing the times at which it predicted a retransmission event with the actual times that retransmissions occurred, as reported by the modified server nodes to the trace collector. When completed, the engine predicted the number of coarse timeouts, fast retransmissions, and slow-start retransmissions to within 0.75% of the correct number (even assuming that no retransmission notifications were lost by the packet capture process). As a result, all the window size calculations are also accurate to within this bound plus the underlying loss rate in the packet capture process (about 0.5% on average).

In summary, the TCP emulation engine is a tool that emulates the evolution of a TCP sender's state with a high degree of accuracy. The emulation engine, combined with the collected traces, allow us to analyze how TCP connections as used by Web browsers behave in today's Internet.

7.4 Analysis Results

In this section, we present the results from our analysis of the real-world behavior of TCP connections. We focus on the behavior of individual connections, analyzing how well the different TCP loss recovery mechanisms work in practice and looking for the presence of receiver bottlenecks.

7.4.1 Retransmissions and Loss Recovery

We classify the retransmissions of lost segments in TCP Reno into three categories—*fast retransmissions*, *timeouts* (Section 2.2.2) and *slow-start retransmissions*, which are retransmissions performed by the sender immediately after a timeout for subsequent packets that were presumed lost in the window. There are two situations that lead to coarse timeouts. In TCP Reno, the loss of multiple segments in a window usually leads to a coarse timeout. Coarse timeouts also occur when the number of duplicate ACKs is insufficient to trigger a fast retransmission.

Running the TCP emulation engine on the observed ACK trace showed that timeouts and the subsequent slow-start retransmissions are the predominant mechanism for loss recovery in TCP Reno. Specifically, over a 3 hour trace involving 1,650,103 connections and 285,979 individual retransmission events, we found that 49.3% of all retransmissions were due to timeouts, 43.8% were the result of a TCP fast retransmission, and 6.9% were the result of slow start retransmissions. That is, 56.2% of all retransmissions occurred soon after a coarse timeout.

We also characterized the state of the sender at the time a loss occurred into slow-start periods and congestion avoidance periods, to determine if either mode shows inherently different behavior from the other. Both congestion avoidance and slow-start have similar frequencies of loss: 82,181 (7%) out of 1,159,588 packets were lost in congestion avoidance and 354,566 (5%) out of 6,662,050 packets were lost in slow start.

From this analysis, we see that existing loss recovery techniques are not effective in recovering from packet losses in a timely manner and that new techniques must be developed to handle them.

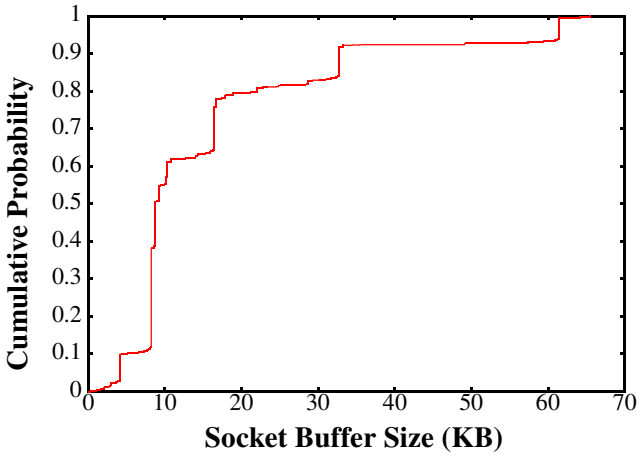

Figure 7.2 Cumulative distribution function of maximum receiver-advertised window sizes.

This result is disappointing because it implies that in practice, the more sophisticated data-driven loss recovery mechanisms are not sufficiently used and that there is a heavy dependence on timeouts for loss recovery. In Section 7.5, we discuss the effectiveness of TCP SACK in reducing the number of timeouts and describe an Enhanced Recovery (ER) technique to significantly improve performance.

7.4.2 Receiver-advertised window

Assuming that the source always has data to send, the amount of data transmitted by TCP at any time is primarily governed by the minimum of two parameters—the sender's congestion window, which tries to track the available bandwidth in the network, and the receiver's advertised window, which handles flow control. The maximum possible value for the latter parameter is equal to the socket buffer size chosen by the application when the connection is established. An excessively small value of this parameter could result in sub-optimal end-to-end performance if the receiver's advertised window is consistently smaller than the sender's congestion window, because the connection's bottleneck links could be under-utilized.

Figure 7.2 shows the cumulative distribution function of receiver socket buffer sizes for transfers to 32,000 hosts (over 1,650,103 connections) during a 3-hour period. The CDF shows distinct upswings at window sizes of 4 KB, 8 KB, 16 KB, etc.—values commonly used by receivers.

For each connection, we compared the receiver advertised window with the congestion window size as computed by the TCP engine. We found that in approximately 14% (233,906) of all connec-

tions the latter grew to be larger than the former multiple times during the connection; i.e., in 14% of all connections in this trace, the receiver advertised window limited the amount of data the TCP sender could have had outstanding. In these cases, the Web client could potentially have obtained a higher throughput had it employed a larger socket buffer and consequently have advertised a larger window.

From this analysis, we make the following recommendation*: future transport implementations should increase their default socket buffer size to avoid the receiver window from becoming a performance bottleneck.* In particular, default values of 4 KB used by some implementations are often too small.

7.5 Reducing the Occurrence of Timeouts

As discussed in Section 7.4.1, over 55% of all retransmissions in our connection trace of the Olympics Web server happened after one or more coarse timeouts. These timeouts keep the connection idle for periods from hundreds of milliseconds to seconds, stalling the steady flow of data to the client. Our analysis showed two main reasons for the occurrence of these timeouts:

1. *Timeout following a fast retransmission:* This happens when the TCP Reno sender is unable to recover from multiple losses within the same transmission window. Typically, the first loss is recovered by the fast retransmission mechanism, which is invoked after three duplicate ACKs reach the sender, but the other losses lead to timeouts.

2. *Insufficient duplicate ACKs:* Available bandwidths on many of today's Internet paths are low and many wireless links are characterized by inherently low channel bandwidths. Because transmission window sizes depend on the connection's bandwidth-delay product, low bandwidth paths keep windows small. When even a small number of losses occur in this small window, the number of arriving duplicate ACKs is not sufficient to trigger a fast retransmission. The result is a timeout-driven retransmission that keeps the link idle for long periods of time. This situation could also occur when most packets in a larger window are lost in the network.

Our solution to these problems has two components: Selective Acknowledgments (SACK) and Enhanced Recovery (ER). SACKs improve loss recovery and reduce the occurrence of timeouts when multiple losses occur in a large transmission window, while ER improves performance when

windows are small—the common case on many Internet and wireless paths. In addition, these two techniques are orthogonal to each other and can coexist for better overall end-to-end performance when losses occur.

7.5.1 Selective Acknowledgments (SACK)

In Chapter 2 (Section 2.3.1), we discussed the IETF SACK mechanism based on RFC 2018 [92]. Recall that this document standardizes only the receiver behavior and SACK header format, leaving the details of the use of SACK information to the sender. The standard allows for up to three out-of-order, maximally contiguous blocks of data to be communicated to the sender, in "most-recent-first" order. The sender behavior is unspecified, except that it should invoke congestion control in response to congestion losses that the arrival of SACKs usually signify.

We now discuss some important details and tasks that the TCP sender performs in our SACK protocol. These mechanisms lead to better congestion control and also improve wireless performance.

1. *Accurate window management:* When duplicate ACKs signifying packet loss arrive in TCP Reno, the sender increments its congestion window by one maximum-sized segment for each incoming duplicate ACK. In general, this is inaccurate because it is not tied to exactly how many data bytes have left the network. For instance, if some duplicate ACKs get lost, the increase is pessimistic; on the other hand, if the sender previously transmitted variable-sized (especially small) segments, the corresponding increase is optimistic. With SACKs, it is possible to avoid this and increment the congestion window precisely by the number of bytes that have reached. This information is available in the SACK blocks, and is relatively robust to ACK loss because information about each block is communicated in at least three different SACKs from the receiver. This is the same approach used in Mathis and Mahdavi's FACK scheme [91].

2. *Fast recovery modification:* In TCP Reno, all duplicate ACKs beyond the third one cause the sender to increment its congestion window by one segment. When more than half the window has been successfully acknowledged, the sender enters the fast recovery phase, transmitting a new segment for each arriving duplicate ACK (Section 2.2.2).

   While this approach does reduce the effective window during fast recovery to half its value at the time of loss, the *rate* at which new segments are transmitted is still approximately equal to the old rate. This is because a new segment is transmitted for each incoming duplicate ACK—

and the rate at which ACKs arrive is approximately equal to the rate at which segments were originally transmitted. If a router en route was incapable of sustaining the original data rate, then it is unlikely to sustain the same rate during fast recovery. This could lead to further losses in this period and require coarse-grained timeouts for recovery.

In contrast, our solution (also independently suggested by Hoe [55]), is to ensure that the transmission rate of new segments is halved during fast recovery. Rather than wait for half the window and then clock data out at the original rate, we transmit one segment for every other incoming ACK. This ensures that the transmission rate is halved, in accordance with congestion control principles. Once fast recovery is complete, the sender's congestion window is set to half the original value at the time of loss.

3. *Integration with Explicit Loss Notification (ELN):* In Chapter 5, we described the details of ELN and its benefits in distinguishing congestion from wireless corruption. By combining ELN messages with SACK information, efficient recovery from wireless losses is possible, especially over wireless networks with large bandwidth-delay products. To understand the benefits of this, consider the following example. Suppose that packets 1, 3 and 5 have been lost due to wireless corruption, all within the same transmission window. With only cumulative ACKs and ELN, each ELN indication triggered by duplicate ACKs indicates the corruption loss of the earliest packet, 1. Thus, it is not clear to the sender why packets 3 and 5 were lost. With SACK, it is possible to indicate exactly which SACK holes correspond to corruption losses, and for the sender to appropriately react to this information.

4. *Handling variable-sized segments:* In standard TCP without SACK, a retransmission usually sends out one MSS worth of data into the network. While this is appropriate for transfers where the sender originally segmented data into MSS-sized packets, it often leads to redundant retransmissions of data already at the receiver. Such variable-sized segments arise in several applications such as interactive terminal sessions (e.g., `telnet`); over low-bandwidth, high-latency wireless networks, these redundant retransmissions excessively reduce goodput and degrade interactive performance.

   It is possible, but not easy or elegant, to solve this problem without the use of SACKs, by maintaining additional state in the TCP sender's control block. However, SACKs provide an elegant solution to this problem, since the sender is now aware of exactly which bytes beyond the miss-

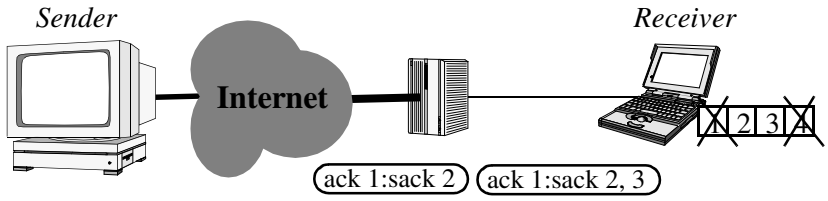

Figure 7.3 Illustration of how small windows affect data-driven loss recovery. Only two duplicate ACKs with SACK are generated, which is insufficient information for the sender to deduce that segment 1 is missing and not just reordered in the network.

ing segment are already at the receiver. Hence, rather than retransmit an MSS worth of data beginning at the first missing one, the sender only retransmits data up to the first selectively acknowledged byte beyond the cumulative ACK.

The performance benefits of SACK are significant for connections whose bandwidth-delay product is large, where multiple losses are likely to occur within a window [42]. Examples of such networks include satellite networks [53] and high-bandwidth terrestrial networks [109].

7.5.2 Enhanced Recovery (ER)

The use of TCP SACK has often been suggested as a technique to improve loss recovery and avoid timeouts unless there is persistent congestion in the network [42]. While this is true in networks with large bandwidth-delay products, the benefits of SACK in preventing timeouts over current (late 1990s) Internet paths and many wireless networks is questionable. If transmission windows are small, then even a small number of losses in the window are not recoverable without timing out, even if the SACK protocol is deployed. An example of such a situation is shown in Figure 7.3.

The problem arises because the sender needs to ensure that a segment is in fact lost and not just *reordered* in the network before initiating a retransmission. Unfortunately, packet reordering is an increasingly common phenomenon on many Internet paths [110], as well as in packet radio networks (e.g., Metricom's Ricochet network [77]). The TCP Reno sender waits for three duplicate ACKs before retransmitting the missing segment; this is a heuristic based on observations of how much reordering typical paths display. When windows are small, not enough duplicate ACKs arrive for the sender to make this distinction, necessitating a timeout.

Observe that in these situations, the information provided by SACKs is not helpful in deducing that reordering has not occurred and in avoiding timeouts (Figure 7.3). A detailed analysis of the trace data based on the output of the TCP emulation engine (Section 7.3) showed that out of the 422,025 coarse timeouts over a 3-hour period, SACKs could have avoided at most 18,713 (4.43%) of them. That is, situations where windows are large enough to have prevented unnecessary timeouts using SACKs are rather infrequent. It is therefore clear that an alternative technique is needed to recover from the bulk of losses, without incurring expensive timeouts.

Our solution to this problem uses the following observations and design principles:

1. *Entire windows are often not lost:* During the same 3-hour period, approximately 403,312 (95.6%) coarse timeouts occurred as a result of insufficient ACK information (i.e., an insufficient number of duplicate ACKs arrived to trigger a retransmission). Of these timeouts, no duplicate ACKs arrived at all for 70% of them. In these situations the network is most likely experiencing severe congestion, since the entire window (or all ACKs) are lost. The best solution for the sender to wait for a coarse timeout to occur before transmitting any packets. However, the remaining 30% of these timeouts occurred even though the entire window was *not* lost, and the fast retransmission mechanism was insufficient. Only 4.43% of these had windows large enough that the deployment of SACKs would have avoided the resulting timeout.
2. *Conservation of packets:* This is the guiding principle used in TCP's transmission algorithms. When an ACK (new or duplicate) arrives at the sender, the sender is assured that at least one segment has left the data pipe and is in the receiver's buffers. Therefore, it can transmit at least as many bytes as have left the pipe without significantly affecting the load it imposes on the network.

We make the following enhancements to the sender algorithm to greatly reduce the occurrence of timeouts. These apply when the window is small, less than six segments.

When the first duplicate ACK arrives, we know that one segment has cleared the pipe and is at the receiver. It is likely that the segment indicated by the duplicate ACK is lost, but the sender cannot be sure of this because of the possibility of packet reordering. Following the conservation of packets principle, the sender can now transmit one segment to the receiver. Because the retransmission of a segment could prove redundant, the sender transmits a *new* segment from beyond the "right

edge" of the current window. Note that this does not violate any congestion control principles because of packet conservation.

Now, if this new segment reaches the receiver, it will generate another duplicate ACK. As long as the number of duplicate ACKs is smaller than three (the threshold for fast retransmissions), the sender transmits new segments in this fashion, probing the network for sustained congestion or loss. Upon the arrival of the third duplicate ACK, the sender performs a retransmission as usual. We call this form of loss recovery using these new "probe" segments *enhanced recovery* (ER). Note that we have not violated standard congestion control procedures by doing this—we transmit a segment only when one has left the data pipe. In addition, if SACK information were also available, the arrival of an ACK with a SACK block indicating the reception of one of the newly transmitted segments is a strong indicator that the original segment was lost, independent of whether three duplicate ACKs arrive or not. Thus, this mechanism will improve performance even when the sender's window is small and losses occur and prevent the detrimental effects of long timeouts unless there is *sustained* network congestion.

In practice, we deploy ER together with Hoe's Newreno techniques, which are orthogonal to ER and complement ER well. Newreno, which keeps the sender in fast recovery if the new ACK that arrives after a fast retransmission is "partial," usually recovers from multiple losses in a large transmission window without incurring a timeout. In this respect, it achieves the same results as SACK, without any changes required at the receiver. However, the rate at which lost segments in the window are recovered is roughly one segment per round trip time, rather than multiple segments in each round-trip duration. ER and Newreno complement each other well because the former prevents unnecessary timeouts when windows are small, while the latter prevents timeouts when windows are large.

We now show that timeouts occur in our protocol only when congestion is persistent. There are three times when timeouts occur. First, when the entire transmission window, or all ACKs for a window, are lost in the network. Because the sender now has no ACK information to perform data-driven loss recovery, a timeout occurs. Second, when a retransmission of a lost segment is also lost. This is also a sign of sustained congestion since the loss situation has persisted for on the order of a round-trip time. Finally, with ER, timeouts may also occur if a newly sent "probe" segment from the right edge of the window was lost, or if the sender did not receive a duplicate ACK for it. As

before, this is also a likely sign of sustained congestion in the network. We note here that if any of these duplicate ACKs had ELN information on them (as explained in Chapter 4), then the corresponding losses occurred because of corruption; these trigger retransmissions *without* the associated congestion control actions at the sender.

7.6 Implementation

In this section, we discuss some details of our implementations of SACK and ER. Our implementation is in the BSD/OS operating system from BSDI [23]. The SACK implementation is conformant with RFC 2018.

7.6.1 SACK

SACK involves changes to both the sender and receiver TCP implementations. RFC 2018 describes the receiver mechanism in detail, leaving large parts of the sender algorithms to specific implementations. Thus, we will focus mostly on the sender details of our implementation and the points discussed in Section 7.5.1.

At the receiver, out-of-sequence segments cause entries to be updated (or newly added) to a list of blocks. Blocks are maximal contiguous regions of received data in the transmission sequence beyond the current cumulative ACK. Whenever a gap is detected in the transmission sequence, duplicate cumulative ACKs with up to three SACK blocks are sent to the TCP sender. The three chosen blocks are ones with most recent activity in them, which is a heuristic that is robust to the loss of some ACKs on the reverse path. These blocks are marshalled into the TCP header in a special SACK option.

At the sender, we maintain a list of holes, which correspond to contiguous gaps in the receiver's data sequence. The arrival of a SACK causes this list to be updated because each SACK usually contains information about the receipt of some data item, information about which was previously unknown to the sender. Each SACK arrival also results in a check to decide how many items in the list of holes are now eligible for retransmission. To distinguish between reordered segments and lost ones, a hole is marked eligible if we know from SACK information that a segment greater than or equal to at least three segments from the missing one has been received successfully.

A hole marked eligible for retransmission is not necessarily retransmitted the moment it is marked. Following the principle of packet conservation, we keep track of how many bytes have currently been acknowledged by successively arriving SACKs. Every time this number increases by twice the MSS, up to one MSS-worth of data may be transmitted. At this point, the sender traverses its list of holes in order, picking out one MSS-worth of data to be retransmitted. If there are no holes eligible for retransmission, *and* if a segment can be transmitted, then a new one is sent out. That is, for a bulk transfer, every other duplicate ACK with SACK information triggers the transmission of a segment, with lost segments (after accounting for possible reordering) sent implicitly at higher priority than new ones.

There are some marked differences between this algorithm and the one TCP Reno normally uses during fast retransmission and recovery. First, in TCP Reno, a retransmission automatically causes an MSS-worth of data starting from the cumulative ACK to be sent, whereas with SACK we retransmit one or more holes reducing the chances that no previously received bytes are resent. Second, in TCP Reno (and Newreno), the duplicate ACKs after the third one cause new segments to be transmitted, not older ones, because the sender does not know for sure which specific segments in the original window are missing. With SACK, typically every other duplicate ACK causes the transmission of an eligible lost segment; only if no segments are eligible is a new segment transmitted. Finally, with SACK, more precise congestion window management is possible because the number of bytes acknowledged by each duplicate ACK is more exact than the rough estimate of one MSS that duplicate ACKs alone indicate. This is especially beneficial in situations when the ACK path is loss-prone or when the original transmissions were not in multiples of the connection's MSS.

7.6.2 ER

One important advantage of the ER scheme is that it involves changes only to the TCP sender and not to the receiver, unlike SACK. This makes it easier to incrementally deploy ER in the Internet and reap its benefits.

If a duplicate ACK arrives when the sender's congestion window is less than or equal to six segments, it transmits one new segment. At this time, it also increments a variable in the control block, `er_sent`, which keeps track of the number of new segments transmitted in each recovery phase.

When the third duplicate ACK arrives and a fast retransmission occurs, the congestion window is decremented by the number of bytes sent in the ER phase. This is because the TCP Reno sender sets its congestion window after a fast retransmission as:

```
tp→snd_cwnd += tp→t_dupacks * tp→t_maxseg;
```

Because ER would have already sent out `tp→er_sent*tp→t_maxseg` bytes, we now adjust the congestion window as:

```
tp→snd_cwnd -= tp→er_sent * tp→t_maxseg;
```

If the sender and receiver have negotiated the SACK option, then ER is terminated upon the arrival of a SACK block acknowledging the receipt of the newly sent segment, even if three duplicate cumulative ACKs have not arrived. This is sufficient to guard against possible reordering in the network.

Finally, if the connection itself is short, there may not be any new segments to transmit in the ER phase when new segments are transmitted. There are two ways of handling this situation:

1. We could transmit only the header of a zero-byte segment corresponding to the last byte of the connection. This approach does not add significantly to the state of congestion if the unit of buffering in the routers is in terms of bytes, as opposed to packets. Then, upon three duplicate ACKs, a fast retransmission of the missing segment is possible as usual.
2. We could lower the fast retransmission threshold when the window is small *and* there are no new segments to transmit. Then if no new segment can be sent upon the arrival of the first (or second) duplicate ACK, the missing one is retransmitted. This enables quicker loss recovery than (1) above and does not introduce any unnecessary probe packets into the network. We adopt this approach in our implementation.

7.7 Performance

In this section, we present measurements of our SACK implementation and simulation and implementation results of ER.

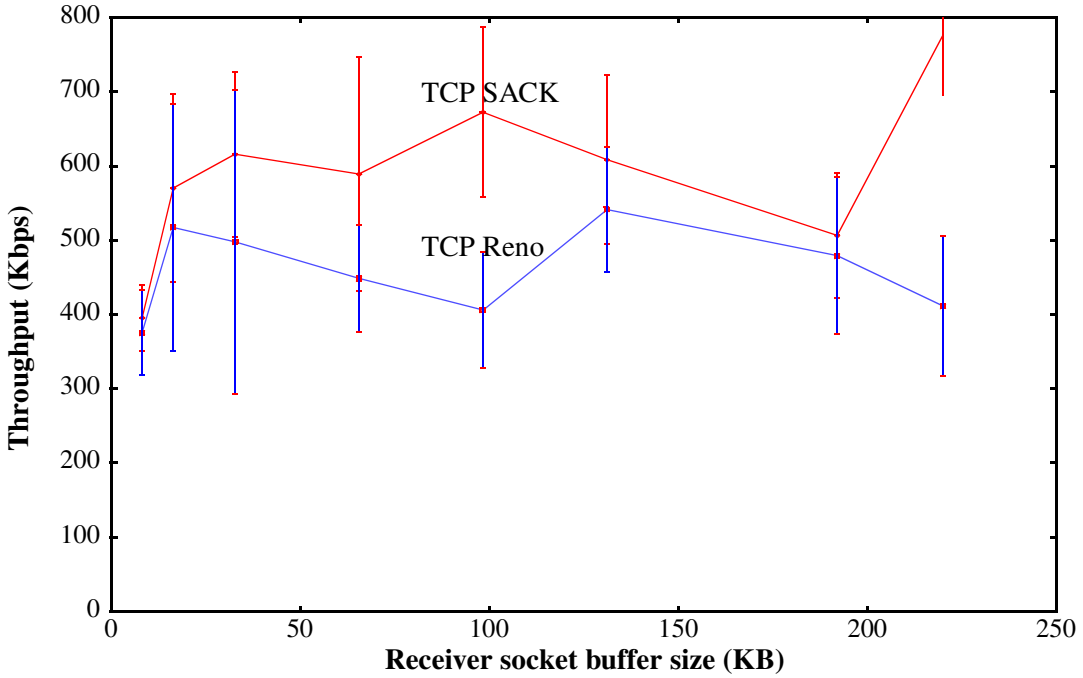

Figure 7.4 Measured throughputs for TCP Reno and SACK over a 16-hop Internet path as a function of the receiver socket buffer size. The performance improvement due to SACK on his particular path is between 5 and 30% compared to Reno. The vertical errorbars show the standard deviations of the measurements.

7.7.1 SACK

We measured the performance of our SACK implementation on a connection over a randomly chosen Internet path to verify the correctness of the implementation and to understand what the typical degree of performance improvement might be when it is deployed in the Internet. One of the hosts was at UC Berkeley (CA) and the other was at IBM Watson Research Center (NY), with 16 Internet hops between them. We performed these experiments during the day when congestion on the Internet caused losses. Each experiment consisted of a 2 MB TCP transfer, repeated ten times at each receiver socket buffer size. The round-trip time between the hosts varied between 90 and 200 ms with an average of 135 ms during the experiments; the maximum bandwidth between the two end hosts was about 1.35 Mbps.

The results of the experiments are shown in Figure 7.4, which plots the TCP throughput as a function of the socket buffer size for SACK and TCP Reno. We find that when socket buffers are small, there is little difference between the two protocols. This is because at small socket buffer sizes,

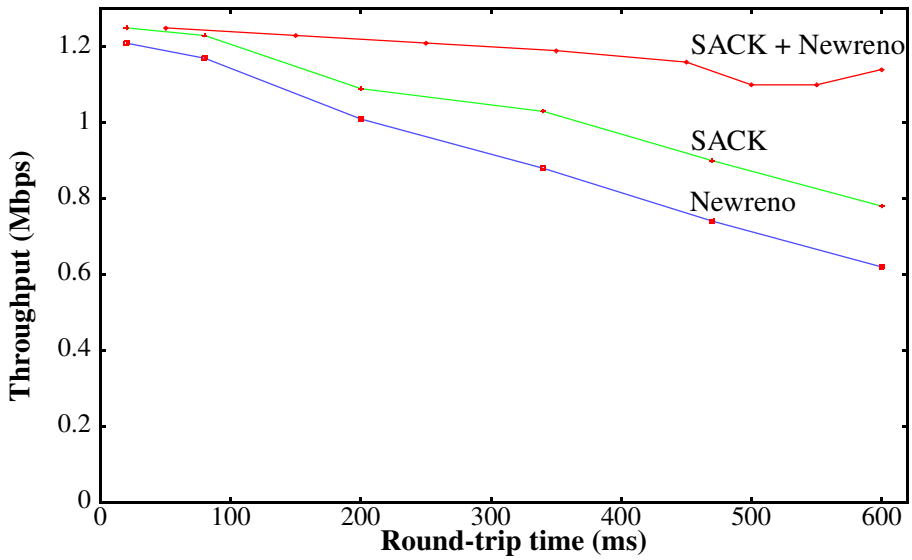

Figure 7.5 Performance of SACK+Newreno, SACK and Newreno as a function of connection round-trip time. (Measurements taken by Tom Henderson, U.C. Berkeley).

TCP transmission windows are small because the buffer size bounds the receiver-advertised window, which limits the number of segments in transit at any point in time. As the socket buffer size increases, the performance of both protocols improves markedly until the buffer size reaches the connection's bandwidth-delay product of 23 KB. At buffer sizes larger than this, SACK performance is between 15% and 30% better than Reno because of its ability to recover from multiple losses in a transmission window without incurring as many timeouts. Thus, over Internet paths where the bandwidth-delay products are large enough to permit large transmission windows, SACK performance is perceptibly better than Reno when losses occur.

Figure 7.5 shows the performance of SACK+Newreno, SACK and Newreno as a function of the connection round-trip time for long TCP transfers with a socket buffer size of 240 KB. No other traffic was present during these experiments, so all losses were caused due to overflows induced by TCP's probing for extra bandwidth.

We find that even when there is no competing traffic, the occurrence of multiple losses in a window significantly degrades the performance of Newreno as round-trip times increase. SACK does better because it recovers more quickly, but the each loss in the window reduces the window by a factor of two. The sender does recover from all losses without a timeout, but its window drops to a small value after this. Then, it takes several round-trips to build again to the bandwidth-delay prod-

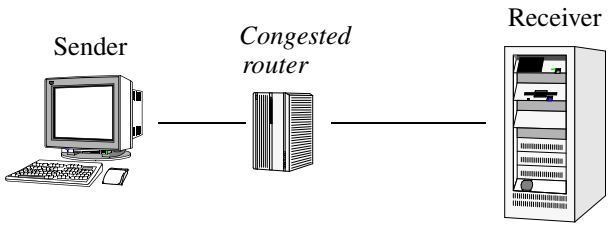

Figure 7.6 Topology for ER simulation experiments.

uct, which explains why this implementation of SACK shows performance degradation at large round-trip times. Finally, the performance of SACK+Newreno is close to optimal, independent of the round-trip time of the connection. This is because SACK promptly recovers from multiple losses and Newreno prevents the window from retracting multiple times. We also note that we do not need Newreno with SACK to perform well; the same benefit can be obtained by ensuring that TCP does not retract its window multiple times in the same round-trip epoch.

7.7.2 ER

We performed an *ns* simulation using the simple topology shown in Figure 7.6 to validate and test the ER algorithm. We generated cross-traffic using a combination of UDP (constant rate) and TCP (Web-like) workloads to force the measured TCP sender to cope with frequent losses at the bottleneck router. These frequent losses resulted in a small average congestion window, affecting data-driven loss recovery and forcing numerous timeouts. This experiment was done to recreate situations that occur in the Olympic Web traces in a controlled manner, where congestion is the result of a large number of flows competing for scarce resources at network bottlenecks. Each simulation experiment consisted of a single TCP transfer from the sender to receiver for a duration of 10 seconds, using SACK+Newreno and ER+Newreno at the sender.

Figure 7.7 shows the sequence plots for TCP with SACK+Newreno and ER+Newreno. As indicated by the large gaps in the sequence plot, the SACK transfer experiences many more coarse timeouts than the transfer using ER. These coarse timeouts are a result of frequent losses that generate two or fewer duplicate ACKs. These duplicates are not sufficient to trigger fast retransmissions at the sender. Therefore, in a traditional TCP implementation the loss is only recovered from after a coarse timeout occurs. With ER, the sender transmits a new packet when each these duplicate ACKs arrive, which elicit additional duplicate ACKs from the receiver. These additional duplicates

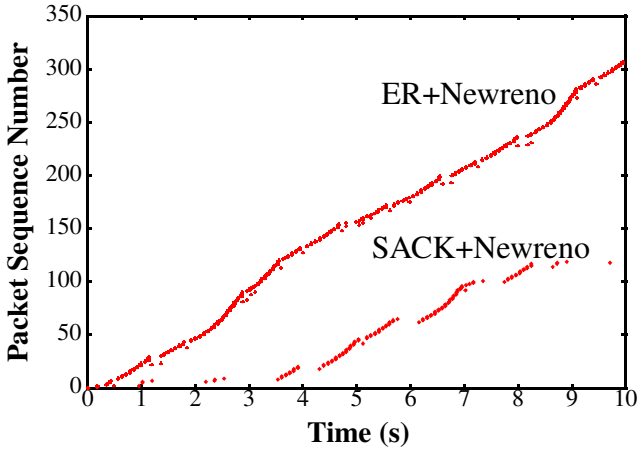

Figure 7.7 TCP with and without ER.

allow the sender to use fast retransmissions to recover from the losses. By eliminating the coarse timeouts, the sender using ER completes the transfer more than twice as fast as the SACK transmitter.

We performed the same experiments using our implementations of SACK+Newreno and ER+Newreno over busy cross-country Internet and wireless paths. We used a buffer size of 32 KB at the receiver. A sample result from one of the runs over the same path as the experiments in Section 7.7.1 is shown in Figure 7.8, which depicts a 5.5-second sequence number trace of the two versions of TCP. We find that SACK+Newreno incurs two long timeouts in this trace, while ER+Newreno successfully prevents any timeouts, leading to a performance improvement of over 100%. We have observed similar results and corresponding performance improvements for ER in experiments conducted over other congested and loss-prone network paths.

7.8 Summary

In this chapter, we analyzed the problems that arise due to small TCP transmission windows and solved them by enhancing TCP's data-driven loss recovery mechanism. Small transmission windows occur because of the inherently low channel bandwidths in many wireless networks and the short lengths of today's Web transfers. The result of small TCP windows is that data-driven loss recovery using fast retransmissions is not triggered often: most losses require expensive timeouts to recover from, keeping the connection idle for long periods of time.

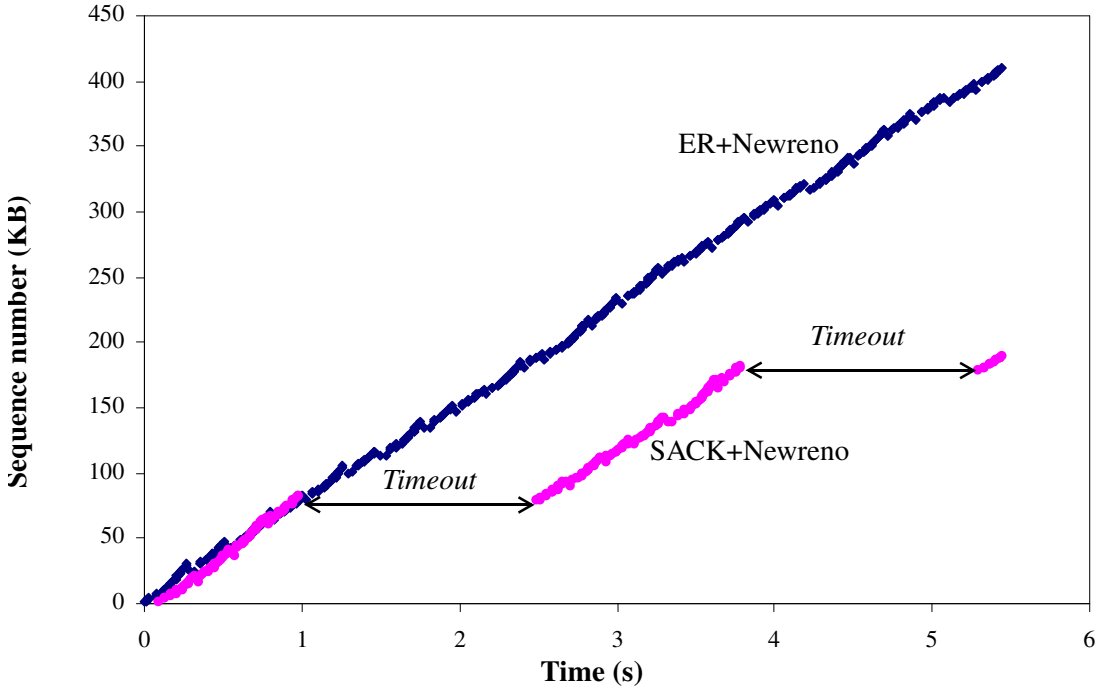

Figure 7.8 Sequence number traces of ER and SACK for a short wide-area transfer. ER successfully prevents sender timeouts, whereas SACK incurs two long timeouts that lead to less than half the throughput of ER in this experiment.

We described two enhancements to TCP's loss recovery algorithms to reduce the occurrence of expensive timeouts, relying instead on data-driven modes of loss recovery: SACK and ER. After discussing the benefits and limitations of SACK, we present a new Enhanced Recovery (ER) scheme that uses the principle of packet conservation to probe the network's state of congestion upon the arrival of early duplicate ACKs in a small window. The successful receipt of these probe segments leads to additional duplicate ACKs from the receiver, which the sender uses to distinguish reordered segments from genuine losses and therefore retransmit missing segments. Together with Newreno, ER ensures that TCP timeouts now occur only if losses are persistent in the network.

"I just want to say one word to you—just one word—'plastics.'"

— From "The Graduate," 1967.

Chapter 8

Conclusions and Future Work

We conclude this dissertation with a summary of our contributions and directions for future work.

8.1 Summary

This thesis identifies three fundamental challenges that degrade the performance of TCP in heterogeneous wireless networks:

1. The preponderance of packet losses due to wireless bit-errors and user mobility.

   TCP performance in many wireless networks suffers because bit-error-induced packet losses, which occur in bursts because of the nature of the wireless channel, are misinterpreted by the TCP sender. TCP attributes these losses to network congestion because of the implicit assumptions made by its congestion control algorithms today. This causes it to reduce its transmission window in response and often causes long timeouts during loss recovery that keep the connection idle. Thus, bit-error losses lead to degraded end-to-end performance. In addition, packet losses that occur due to user mobility cause the TCP sender to remain idle for long periods of time even after the handoff is completed, resulting in unacceptably low throughput.

2. Asymmetric effects and latency variability.

TCP performance in bandwidth-asymmetric networks like wireless cable modem networks suffers because of the low bandwidth ACK channel. TCP over packet radio networks comprised of half-duplex radios does not perform well because of large latency variation and ACK queueing that occurs in the network. In both these networks, the root cause of problems is that TCP performance degrades if the ACK feedback is imperfect, slow or variable.

3. Small transmission windows due to low bandwidths and short connections.

   The bit rate commonly available between distant hosts in the Internet today is often limited and time-varying. This is especially true in many wide-area wireless networks where maximum bandwidths are often orders of magnitude lower than their wired counterparts. In addition, today's dominant TCP application, the World Wide Web, uses short TCP connections because most Web transfers are short. When the available bandwidth is low, so is the transmission window. Small TCP transmission windows prevent the sender from recovering from losses without incurring expensive timeouts that keep the connection idle for long periods of time. As a result, end-to-end throughputs are significantly lower than the already-low maximum channel bandwidth: TCP transfers in these situations yield throughputs that are only a fraction of what is achievable.

While TCP adapts well to network congestion, it does not adequately handle these problems of wireless media. This dissertation analyzed the problems posed by the above challenges, and solved them using a combination of link-layer techniques and modifications and enhancements to TCP at the sender and receiver.

- **The Berkeley Snoop protocol:** This protocol combines a *transport-aware link protocol* and a *link-aware transport protocol,* which improves performance in networks composed of wired and single-hop cellular wireless links. The key idea here is the deployment of a soft-state agent at the base station to perform local loss recovery and shield the TCP sender from the vagaries of the wireless link. The details of the protocol are discussed in Chapter 4.
- **Explicit Loss Notification (ELN):** ELN is a general *framework* by which receivers, base stations, or other elements in the network can inform the TCP sender of losses that occur for reasons other than congestion. When combined with algorithms to distinguish congestion losses

from corruption, this framework provides a powerful way by which TCP senders can separate congestion control from loss recovery and recover from non-congestion-related losses without invoking congestion control. The details of ELN are discussed in Chapters 4 and 5.

- **Asymmetric TCP enhancements:** We first formally defined asymmetry with respect to TCP performance in terms of how the characteristics of one direction affect performance in the other and classified different types of asymmetry based on bandwidth, latency, media-access, and loss-rate asymmetries. We then presented a suite of end-to-end and link-layer algorithms to overcome the adverse effects of asymmetry. These involve techniques to reduce the frequency of ACKs (ACK filtering) and techniques to handle reduced ACK feedback (TCP sender adaptation and ACK reconstruction), and improve performance in both packet radio networks that exhibit variable latencies and ACK queueing, and in bandwidth-asymmetric networks with low-bandwidth ACK paths. These details are discussed in Chapter 6.
- **Enhanced TCP loss recovery:** This is an enhancement to TCP's data-driven loss recovery mechanism that improves performance by reducing the number of timeouts when transmission windows are small. These details are discussed in Chapter 7.

In addition to solving the specific problems in wireless networks discussed above, we also derive a set of general principles and lessons to design transport protocols for heterogeneous networks. These include:

- **Cross-layer protocol optimizations:** In our work, we develop and refine the idea of *cross-layer protocol optimizations* as a design methodology by which information is exposed across multiple layers of the conventional protocol stack to optimize end-to-end performance. For example, the Snoop protocol involves a transport-aware link agent, which takes advantage of the reliability mechanisms of the transport layer and does not generate its own independent messages. It also suppresses certain duplicate TCP ACKs from the sender. Another example is the design of the ACK filter and ACK reconstructor agents to combat the adverse effects of asymmetry; these agents perform TCP-aware connection-specific actions to achieve better performance.

  We believe that this is a powerful technique that can be generalized to many other situations where protocols are being designed with good network performance in mind. This design principle is orthogonal to Integrated Layer Processing (ILP) [Clark90], which proposes that multi-

ple layers be coalesced to achieve efficient end-host implementations; in contrast, cross-layer interactions preserve the principle of layering, but expose certain critical parts of different layers to each other for improved network performance. Chapter 5 discusses this principle in greater detail.

- **Soft-state:** A second general design principle that we use in the design of many of the protocols presented here is the notion of soft-state agents in the network. To enhance performance, agents are often deployed in the network and have state associated with them. However, this state is soft [31], i.e., it is not essential for the correct end-to-end functioning of the connection. It is also periodically refreshed by the arrival of successive packets and ACKs, which implies that even if it gets corrupted, the connection itself is not compromised. This is in direct contrast to protocols that deploy hard-state agents in the network, such as split-connection methods for wireless TCP. Chapters 4, 5, and 6 discussed this principle in detail in the context of specific network agents, such as the Snoop protocol, ELN-enhanced TCP, and link-layer solutions for asymmetric networks.

  In our work, the design of soft-state agents has been guided by the end-to-end argument in systems design [127]. These agents must be viewed as performance optimizers, are not essential for correctness, and do not compromise semantics. Another guiding principle has been the conformance to the IP service model; in particular, we use the fact that packets can be dropped or duplicated in an IP network to great advantage, by suppressing certain types of ACKs, generating local retransmissions, etc.

- **Data-driven loss recovery:** This applies in general to the design of network protocols that attempt to achieve reliability using retransmissions. There are two methods by which reliable transport protocols decide that packets are lost and then retransmit them: using timers (*timer-driven retransmissions*) and using information from other data packets that have reached, usually later ones (*data-driven retransmissions*). Because it is important for transport protocols not to retransmit packets that might still be in transit, retransmission timers are necessarily conservative. Thus, to achieve low recovery latencies and avoid long idle periods that timeouts impose, it is important to make loss recovery as data-driven as possible. We achieve this in the

Snoop protocol by taking advantage of the information provided by TCP cumulative ACKs (and SACK, when available) as well as in the enhanced TCP loss recovery for small window connections.

8.2 Availability

All the protocol and software implementations described in this dissertation are available on-line in source and binary forms.

The Berkeley Snoop protocol is available for BSD/OS 2.1 and BSD/OS 3.0 from

`http://www.cs.berkeley.edu/~hari/software/snoop.html`

Our TCP Selective Acknowledgments implementation for BSD/OS 2.1 and 3.0 is available from

`http://www.cs.berkeley.edu/~hari/software/tcpsack.html`

Implementations of the protocols presented in this thesis are also available in the *ns* network simulator. This can be downloaded from

`http://www-mash.cs.berkeley.edu/ns/`

The `netperf` network performance utility is available at

`ftp://daedalus.cs.berkeley.edu/netperf/np.tar.gz`

8.3 Future Directions

There are several interesting directions for future work based on the work described in this dissertation. Some of these are extensions of our work, while some others are motivated by the more general problem of ubiquitous information access over wireless technologies.

Transport over other wireless networks: In this dissertation, we investigated transport performance over in-room and in-building wireless LANs, the Ricochet packet radio network, and the asymmetric-bandwidth Hybrid wireless cable network. Several other wireless technologies have not been addressed in this thesis, including Cellular Digital Packet Data (CDPD), Global System

for Mobility (GSM), and satellite networks. The CDPD network piggybacks packet data on the existing analog cellular network and provides of less than 19.2 Kbps. In addition to wireless bit-errors that are overcome using a link-layer protocol, this network displays asymmetry in media-access. Base stations gain easy access to the channel, while end-stations have to contend with each other to send packets. This leads to variations in latency and degrades performance. In the GSM network, the current link-layer error control protocol (RLP) interacts adversely with TCP, as mentioned in Section 2.5.2.

Satellite networks pose interesting problems because of large communication latencies. In addition, several satellite networks like the Direct Broadcast Satellite (DBS) system display bandwidth asymmetry. While we expect some of our techniques from Chapter 6 to apply to these networks, there are other important problems to overcome, such as unfairness issues and error control. This is an area of active research (e.g., [140], [53]).

Automatic detection of heterogeneity: Throughout our work, we have assumed that we know in advance what networks are present on our path, thus being able to deploy our solutions. For example, by explicitly knowing that the network is bandwidth asymmetric, we know to deploy schemes like ACK filtering for that connection. Ideally, we should deduce that a connection traverses an asymmetric path from end-to-end measurements and observations, and automatically configure the appropriate protocols.

Impact of short-term unfairness in wireless media-access protocols: Common wireless media-access protocols like CSMA/CA provide long-term (asymptotic) fairness, but are unfair in the short-term. That is, over short time-scales, multiple contending stations do not gain equal access to the wireless medium. This affects the performance of higher-layer protocols and applications in many ways. For example, constant-rate audio streams have large jitter values, increasing playback buffer requirements at end hosts. For TCP, ACKs arrive in bursts rather than interleaved with segments in a window, leading to ACK compression and burst transmissions at the sender. We need to develop a framework to quantify short-term unfairness and design distributed contention resolution protocols (e.g., [19]) that have better fairness properties than CSMA/CA.

Networks of Devices (NOD): Until now, research in the areas of mobile computing and wireless networking has focused on providing access to information “anytime” and “anywhere.” This

includes our work too. There are several applications, such as remote sensing, surveying, monitoring, and control applications, that are enabled by providing network access to unconventional devices. Examples of these devices include video cassette recorders, temperature sensors, climate monitors, household appliances, remote controls, automobiles, printers, copiers and cameras. Now, in addition to "anytime" and "anywhere" access, these devices enable access "*from anywhere*." Easy access to information located at remote and inconvenient sites is a key ingredient in the vision of ubiquitous information access.

However, there are numerous difficult technical challenges that must be overcome if this vision is to become a reality. We must handle issues of scale, reliability, power consumption, unconventional and harsh communication channels (e.g., wireless links and power lines), and the possible scarcity of computational resources at end devices. Scaling a NOD to numbers on the order of several thousand or more devices is important in many contexts, such as in wide-area sensor networks. Many wireless protocols are notorious for poor performance when the communication channel is shared by more than a small number of hosts. Improving media-access protocols and designing scalable control protocols between NOD entities will result in a system that works well as the number of devices grows.

As an example of the rich class of applications enabled by NODs, consider a large cooperative network of video (or still-image) cameras surveying a remote geographical area. The goal of this system is to provide a reliable and complete a coverage of the remote area, disseminating data in real-time to a remote station. Each camera can change its orientation based on messages sent to it and may also move. Clearly, robust wireless communication is essential for this to work, which implies the need for protocols to combat channel errors and other problems posed by this harsh environment. Most transport protocols today optimize throughput or latency, without considering power consumption in end devices. In this environment, we have to explicitly consider this issue in protocol design. More importantly, we need to design scalable, multicast-based, application-specific control protocols between devices, making sure that message implosion is avoided. Finally, such large amorphous networks must be designed to *self-organize* without any prior structure or hierarchy being imposed on the system.

A fruitful approach to this research goal would be to build some applications like the one described above, and then extract a toolkit that other new applications could use to perform tasks related to scalable control messages, self-organization, and error-free data transport.

Home networking: Another example of a new genre of networks is the class of networked consumer devices in homes and offices. The challenges here are similar to those in NODs (e.g., self-organization), but system security and service location become especially important problems. Furthermore, there are likely to be several types of communication elements in networks at home (wireless, telephone, cable, power lines), making it important to understand and solve performance problems.

Location-dependent applications and services: Little attention has been paid so far to developing novel location-dependent applications for wireless mobile systems, especially on a large scale involving hundreds to thousands of devices. Developing a general infrastructure for applications to exploit knowledge about the geographical or physical locations of users, devices and services in likely to be an area of fertile research in the near future, with the promise of significant commercial impact. We believe that the deployment of a general purpose system infrastructure for location-dependent applications will foster the development of new, compelling applications and accelerate the deployment and use of wireless technologies.

If I have seen further it is by standing on the shoulders of Giants.

— Sir Isaac Newton, letter to Sir Robert Hooke, Feb. 5, 1676.

Bibliography

[1] N. Abramson. The ALOHA system – Another Alternative for Computer Communications. In *Fall Joint Computer Conference, AFIPS Conference Proceedings*, volume 37, pages 281–285, 1970.

[2] A. S. Acampora and M. Naghshineh. An Architecture and Methodology for Mobile-Executed Handoff in Cellular ATM. *IEEE Journal on Selected Areas in Communications*, 12(8):1365–1375, October 1994.

[3] IBM AlphaWorks Home Page. http://www.alphaworks.ibm.com, 1996.

[4] E. Amir, S. McCanne, and H. Zhang. An Application Level Video Gateway. In *Proc. ACM Multimedia*, November 1995.

[5] Aironet Wireless Communications Home Page. http://www.aironet.com, 1998.

[6] Infrared Data Association. *IrDA Serial Infrared (SIR) Physical Layer Link Specification (IrPHY) 1.1*. Infrared Data Association, 1995.

[7] E. Ayanoglu, S. Paul, T. F. LaPorta, K. K. Sabnani, and R. D. Gitlin. AIRMAIL: A Link-Layer Protocol for Wireless Networks. *ACM ACM/Baltzer Wireless Networks Journal*, 1:47–60, February 1995.

[8] A. Bakre and B. R. Badrinath. Handoff and System Support for Indirect TCP/IP. In *Proc. Second Usenix Symp. on Mobile and Location-Independent Computing*, April 1995.

[9] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *Proc. 15th International Conf. on Distributed Computing Systems (ICDCS)*, May 1995.

[10] A. Bakre and B. R. Badrinath. Implementation and Performance Evaluation of Indirect-TCP. *IEEE Trans. on Computers*, 46(3), March 1997.

[11] H. Balakrishnan. An Implementation of TCP Selective Acknowledgments. ftp://daedalus.cs.berkeley.edu/pub/tcpsack/, 1996.

[12] H. Balakrishnan and R.H. Katz. Explicit Loss Notification and Wireless Web Performance. In *Proc. Globecom98 (Internet Mini Conference)*, November 1998.

[13] H. Balakrishnan, V. N. Padmanabhan, and R.H. Katz. The Effects of Asymmetry on TCP Performance. In *Proc. ACM/IEEE MOBICOM Conf.*, September 1997.

[14] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R.H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Trans. on Networking*, 5(6), December 1997.

[15] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R.H. Katz. TCP Behavior of a Busy Web Server: Analysis and Improvements. In *Proc. IEEE INFOCOM*, March 1998.

[16] H. Balakrishnan, S. Seshan, and R.H. Katz. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *ACM Wireless Networks*, 1(4), December 1995.

[17] H. Balakrishnan, S. Seshan, M. Stemm, and R.H. Katz. Analyzing Stability in Wide-Area Network Performance. In *Proc. ACM SIGMETRICS '97*, June 1997.

[18] T. Berners-Lee et al. The World Wide Web. *Communications of the ACM*, 37(8):76–82, Aug 1994.

[19] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: A Media-Access Protocol for Packet Radio. In *Proc. ACM SIGCOMM*, 1994.

[20] R. T. Braden. *Requirements for Internet Hosts – Communication Layers*. Information Sciences Institute, Marina del Rey, CA, October 1989. RFC-1122.

[21] 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*, August 1994.

[22] K. Brown and S. Singh. M-TCP: TCP for Mobile Cellular Networks. *Computer Commun. Review*, 27(5), October 1997.

[23] Berkeley Software Design, Inc. http://www.bsdi.com, 1998.

[24] Cablelabs Home Page. http://www.cablelabs.com, 1998.

[25] R. Caceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE Journal on Selected Areas in Communications*, 13(5), Jun 1995.

[26] R. Caceres and V. N. Padmanabhan. Fast and Scalable Handoffs in Wireless Internetworks. In *Proc. 1st ACM Conf. on Mobile Computing and Networking*, November 1996.

[27] D. Cheriton. VMTP: A Transport Protocol for the Next Generation of Communication Systems. In *Proc. ACM SIGCOMM '86*, pages 406–415, September 1986.

[28] D. Cheriton and C. Williamson. VMTP as the Transport Layer for High-Performance Distributed Systems. *IEEE Commun. Mag.*, pages 37–44, June 1989.

[29] S. Cheshire and M. Baker. A Wireless Network in MosquitoNet. *IEEE Micro*, Feb 1996.

[30] D-M. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, 17:1–14, 1989.

[31] D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proc. ACM SIGCOMM*, August 1988.

[32] D. Clark, M. L. Lambert, and L. Zhang. NETBLT: A High Throughput Transport Protocol. In *Proc. ACM SIGCOMM*, August 1988.

[33] D. Clark and D. Tennenhouse. Architectural Consideration for a New Generation of Protocols. In *Proc. ACM SIGCOMM*, September 1990.

[34] D. C. Clark, V. Jacobson, J. Romkey, and H. Salwen. An Analysis of TCP Processing Overhead. *IEEE Communication Magazine*, Jun 1989.

[35] R. Cohen and S. Ramanathan. TCP for High Performance in Hybrid Fiber Coaxial Broad-Band Access Networks. *IEEE/ACM Trans. on Networking*, 6(1), February 1998.

[36] The Daedalus Group Home Page. http://daedalus.CS.Berkeley.EDU/, 1998.

[37] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulations of a Fair-Queueing Algorithm. *Internetworking: Research and Experience*, V(17):3–26, 1990.

[38] A. DeSimone, M. C. Chuah, and O. C. Yue. Throughput Performance of Transport-Layer Protocols over Wireless LANs. In *Proc. Globecom '93*, December 1993.

[39] DirecPC Home Page. http://www.direcpc.com, 1998.

[40] W. Doeringer et al. A Survey of Light-Weight Transport Protocols for High-Speed Networks. *IEEE Transactions on Communications*, 38(11):2025–2038, November 1990.

[41] R. Durst, G. Miller, and E. Travis. TCP Extensions for Space Communications. In *Proc. ACM Mobicom Conference*, November 1996.

[42] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and Sack TCP. *Computer Communications Review*, July 1996.

[43] Federal Communications Commission (FCC) Home Page. http://www.fcc.gov, 1998.

[44] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, Jan 1997. RFC-2068.

[45] S. Floyd. TCP and Explicit Congestion Notification. *Computer Communications Review*, 24(5), October 1994.

[46] S. Floyd. Internet Research: Comments on Formulating the Problem. ftp://ftp.ee.lbl.gov/papers/assumptions.ps, 1998. Unpublished manuscript.

[47] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4), August 1993.

[48] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Cluster-based Scalable Network Services. In *Proc. 16th ACM Symposium on Operating Systems Principles*, October 1997.

[49] V. Garg and J. Wilkes. *Wireless and Personal Communications Systems*. Prentice-Hall, 1996. Chapter 8.

[50] R. Ghai and S. Singh. An Architecture and Communications Protocol for Picocellular Networks. *IEEE Personal Communications Magazine*, 1(3):36–46, 1994.

[51] Survey by Peter D. Hart Associates, 1998. Available from http://www.wow-com.com.

[52] G. H. Heilmeier. Global Begins at Home. *IEEE Communications Magazine*, October 1992.

[53] T. Henderson and R.H. Katz. Transport Protocols for Internet-Compatible Satellite Networks. *IEEE Journal on Selected Areas in Communications*, 1998. Submitted.

[54] T. Hodes and R.H. Katz. Composable Ad hoc Location-based Services for Heterogeneous Mobile Clients. *ACM Wireless Networks*, 1998. Special issue on Mobile Computing, to appear.

[55] J. C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *Proc. ACM SIGCOMM '96*, August 1996.

[56] Hybrid Networks Inc. http://www.hybrid.com, 1998.

[57] J. Ioannidis, D. Duchamp, and G. Q. Maguire. IP-based Protocols for Mobile Internetworking. In *Proc. ACM SIGCOMM '91*, pages 235–245, 1991.

[58] J. Ioannidis and G. Q. Maguire. The Design and Implementation of a Mobile Internetworking Architecture. In *Proc. Winter '93 Usenix Conference*, San Diego, CA, January 1993.

[59] V. Jacobson. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM 88*, August 1988.

[60] V. Jacobson. *Compressing TCP/IP Headers for Low-speed Serial Links*. RFC-1144, Feb 1990.

[61] V. Jacobson, R. Braden, and D. Borman. *TCP Extensions for High Performance*, May 1992. RFC-1323.

[62] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, 1991.

[63] R. Jain. Myths About Congestion Management in High-Speed Networks. *Internetworking: Research and Experience*, 3:101–113, 1992.

[64] A. Jameel, M. Stuempfle, D. Jiang, and A. Fuchs. Web on Wheels: Toward Internet-Enabled Cars. *IEEE Computer*, January 1998.

[65] J. Jubin and J. Tarnow. The DARPA Packet Radio Network Protocols. *Proc. of the IEEE*, 75(1):21–32, January 1987.

[66] J. Kahn and J.R. Barry. Wireless Infrared Communications. *Proc. of the IEEE*, February 1997.

[67] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan. Improving TCP Throughput over Two-Way Asymmetric Links: Analysis and Solutions. In *Proc. ACM SIGMETRICS*, June 1998.

[68] P. Karn. MACA – A New Channel Access Method for Packet Radio. In *Proc. 9th ARRL Computer Networking Conference*, 1990.

[69] P. Karn. The Qualcomm CDMA Digital Cellular System. In *Proc. 1993 USENIX Symp. on Mobile and Location-Independent Computing*, pages 35–40, August 1993.

[70] P. Karn. Dropping TCP acks. Mail to the end-to-end mailing list, February 1996.

[71] P. Karn and C. Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. *ACM Transactions on Computer Systems*, 9(4):364–373, November 1991.

[72] R. H. Katz and E. A. Brewer. The Case for Wireless Overlay Networks. In *Proc. SPIE Multimedia and Networking Conference (MMNC'96)*, San Jose, CA, Jan 1996.

[73] J. Kay and J. Pasquale. The Importance of Non-Data Touching Processing Overheads in TCP/IP. In *Proc. ACM SIGCOMM '93*, San Francisco, CA, Sep 1993.

[74] S. Keshav. REAL: A Network Simulator. Technical Report 88/472, Computer Science Division, Univ. of California at Berkeley, 1988.

[75] S. Keshav. Packet-Pair Flow Control. *IEEE/ACM Transactions on Networking*, February 1995.

[76] S. Keshav and S. Morgan. SMART Retransmission: Performance with Overload and Random Losses. In *Proc. Infocom '97*, 1997.

[77] K. Lai, M. Roussopoulos, D. Tang, X. Zhao, and M. Baker. Experiences with a Mobile Testbed. In *Proc. 2nd Intl. Conf. on Worldwide Computing*, March 1998.

[78] T. V. Lakshman, U. Madhow, and B. Suter. Window-based Error Recovery and Flow Control with a Slow Acknowledgement Channel: A study of TCP/IP Performance. In *Proc. Infocom 97*, April 1997.

[79] B. Leiner, D. Nielson, and F. Tobagi. Issues in packet radio network design. *Proc. of the IEEE*, 75(1):6 – 20, January 1987.

[80] T. Lewis. *Empire of the Air: The Men Who Made Radio*. Harper Collins, 1991.

[81] M. Liljeberg, H. Helin, M. Kojo, and K. Raatikainen. Mowgli WWW Software: Improved Usability of WWW in Mobile WAN Environments. In *Proc. Global Internet*, pages 33–37, November 1996.

[82] D. Lin and H. Kung. TCP Fast Recovery Strategies: Analysis and Improvements. In *Proc. INFOCOM*, March 1998.

[83] S. Lin and D. J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Inc., 1983.

[84] J.-P. Linnartz. *Narrowband Land-Mobile Radio Networks*. Artech House, 1993.

[85] D. E. Long et al. REINAS: A Real-time System for Managing Environmental Data. In *Proc. Conf. on Software Engg. and Knowledge Engg.*, 1996.

[86] M. Lottor. http://www.nw.com/zone/WWW/top.html, 1997.

[87] R. Ludwig. Exploiting Local Information for End-to-End Congestion Control, 1998. In preparation.

[88] B. A. Mah. An Empirical Model of HTTP Network Traffic. In *Proc. InfoComm '97*, April 1997.

[89] D. Maltz and P. Bhagwat. MSOCKS: An Architecture for Transport Layer Mobility. In *Proc. INFOCOM*, March 1998.

[90] P. Manzoni, D. Ghosal, and G. Serazzi. Impact of Mobility on TCP/IP: An Integrated Performance Study. *IEEE Journal on Selected Areas in Communications*, 13(5):858–867, June 1995.

[91] M. Mathis and J. Mahdavi. Forward Acknowledgement: Refining TCP Congestion Control. In *Proc. ACM SIGCOMM*, Aug 1996.

[92] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. *TCP Selective Acknowledgment Options*, 1996. RFC-2018.

[93] A. J. McAuley. Error Control for Messaging Applications in a Wireless Environment. In *Proc. INFOCOM Conf.*, April 1995.

[94] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-Level Packet Capture. In *Proc. Winter '93 USENIX Conference*, San Diego, CA, January 1993.

[95] S. McCanne and V. Jacobson. vic: A Flexible Framework for Packet Video. In *Proc. ACM Multimedia '95*, November 1995.

[96] S. McCanne et al. Toward a Common Infrastructure for Multimedia-Networking Middleware. In *Network and OS Support for Digitial Audio and Video Workshop*, May 1997.

[97] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman. *The Design and Implementation of the 4.4 BSD Operating System*. Addison-Wesley, Reading, MA, 1996.

[98] Metricom, Inc. http://www.metricom.com and http://www.ricochet.net, 1998.

[99] M. Mouly and M.-B. Pautet. *The GSM System for Mobile Communications*. Cell & Sys, 1992.

[100] A. Myles, D.B. Johnson, and C. Perkins. A Mobile Host Protocol Supporting Route Optimization and Authentication. *IEEE Journal on Selected Areas in Communications*, 13(5), June 1995.

[101] 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.

[102] G. T. Nguyen, R. H. Katz, B. D. Noble, and M. Satyanarayanan. A Trace-based Approach for Modeling Wireless Channel Behavior. In *Proc. Winter Simulation Conference*, Dec 1996.

[103] ns-2 Network Simulator. http://www-mash.cs.berkeley.edu/ns/, 1998.

[104] Official 1996 olympic web site - home page. http://www.atlanta.olympic.org, 1996.

[105] OSI Transport Protocol Specification, 1986. Standard ISO-8073.

[106] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley Publishing Company, Reading, MA, 1994.

[107] V. N. Padmanabhan. *Addressing the Challenges of Web Data Transport*. PhD thesis, Univ. of California, Berkeley, 1998. In preparation.

[108] V. N. Padmanabhan and J. C. Mogul. Improving HTTP Latency. In *Proc. Second International WWW Conference*, October 1994.

[109] C. Partridge. *Gigabit Networking*. Addison-Wesley, 1994.

[110] V. Paxson. End-to-End Routing Behavior in the Internet. In *Proc. ACM SIGCOMM '96*, August 1996.

[111] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, U. C. Berkeley, May 1997.

[112] C. Perkins. *IP Mobility Support*. RFC, Oct 1996. RFC-2002.

[113] PalmPilot Home Page. http://palmpilot.3com.com, 1998.

[114] J. B. Postel. *User Datagram Protocol*, August 1980. RFC-768.

[115] J. B. Postel. *Internet Protocol*. RFC, Information Sciences Institute, Marina del Rey, CA, September 1981. RFC-791.

[116] J. B. Postel. *Transmission Control Protocol*. Information Sciences Institute, Marina del Rey, CA, September 1981. RFC-793.

[117] J. B. Postel. *Simple Mail Transfer Protocol*. Information Sciences Institute, Marina del Rey, CA, August 1982. RFC-821.

[118] J. B. Postel and J. Reynolds. *TELNET Protocol Specification*. Information Sciences Institute, Marina del Rey, CA, May 1983. RFC-854.

[119] J. B. Postel and J. Reynolds. *File Transfer Protocol (FTP)*. Information Sciences Institute, Marina del Rey, CA, Oct 1985. RFC-821.

[120] Proxim — Wireless LAN Networking Products Based on Spread Spectrum Technology. http://www.proxim.com, 1998.

[121] R. Quick and K. Balachandran. Overview of the Cellular Digital Packet Data (CDPD) System. In *Proc. of the PIMRC*, pages 338–343, 1993.

[122] K. K. Ramakrishnan and R. Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Transactions on Computer Systems*, 8(2):158–181, May 1990.

[123] K.K. Ramakrishnan and S. Floyd. A Proposal to Add Explicit Congestion Notification (ECN) to IPv6 and to TCP. Internet Draft draft-kksjf-ecn-00.txt, November 1997. Work in progress.

[124] T.S. Rappaport. *Wireless Communications — Principles and Practice*. Prentice-Hall, Englewood Cliffs, NJ, 1996.

[125] Reliable Multicast Research Group. http://www.east.isi.edu/RMRG/, 1997.

[126] M. Ritter. The Metricom Autobahn Architecture. Personal communication, 1997.

[127] J. Saltzer, D. Reed, and D. Clark. End-to-end Arguments in System Design. *ACM Transactions on Computer Systems*, 2:277–288, Nov 1984.

[128] R. Scholtz, M. Simon, J. Omura, and B. Levitt. *Spread Spectrum Communications Handbook*. McGraw-Hill, 1994.

[129] S. Seshan. *Low Latency Handoffs in Cellular Data Networks*. PhD thesis, University of California at Berkeley, December 1995.

[130] S. Seshan and H. Balakrishnan. netperf: Network Performance Utility. ftp://daedalus.cs.berkeley.edu/pub/netperf, 1996.

[131] S. Seshan, H. Balakrishnan, and R. H. Katz. Handoffs in Cellular Wireless Networks: The Daedalus Implementation and Experience. *Kluwer Journal on Wireless Personal Communications*, January 1997.

[132] W. Simpson. *The Point-to-Point Protocol (PPP)*. RFC, Dec 1993. RFC-1548.

[133] M Stemm. Vertical Handoffs in Wireless Overlay Networks. Technical Report csd-96-903, University of California at Berkeley, May 1996. Master's Thesis.

[134] W. R. Stevens. *UNIX Network Programming*. Addison-Wesley, Reading, MA, 1992.

[135] W. R. Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley, Reading, MA, Nov 1994.

[136] W. R. Stevens. *TCP/IP Illustrated, Volume 3*. Addison-Wesley, Reading, MA, April 1996.

[137] W. T. Strayer, B. J. Dempsey, and A. C. Weaver. *Xtp: The Xpress Transfer Protocol*. Addison-Wesley, Reading, MA, 1992.

[138] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, MA, 1997.

[139] A. Tanenbaum. *Computer Networks*. Prentice-Hall, 1996.

[140] IETF TCP over satellite working group. E-mail: tcp-over-satellite@achtung.sp.trw.com, 1998.

[141] K. Thompson, G. J. Miller, and R. Wilder. Wide-area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11(6), November/December 1997.

[142] VINT Project. http://netweb.usc.edu/vint, 1998.

[143] Wireless Application Protocol. http://www.wapforum.com, 1998.

[144] R. W. Watson and S. A. Mamrak. Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices. *ACM Transactions on Computer Systems*, 5(2):97–120, May 1987.

[145] The WaveLAN Home Page. http://www.wavelan.com, 1998.

[146] D. Wetherall and C. Lindblad. Extending Tcl for Dynamic Object-Oriented Programming. In *Proc. Tcl/Tk Workshop*, July 1995.

[147] WOM Boiler Room. http://www.womplex.ibm.com, 1996.

[148] G.R. Wright and W. R. Stevens. *TCP/IP Illustrated, Volume 2*. Addison-Wesley, Reading, MA, Jan 1995.

[149] 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.

[150] L. Zhang. Why TCP Timers Don't Work Well. In *Proc. ACM SIGCOMM*, pages 397–405, August 1986.

[151] H. Zimmermann. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Trans. on Communications*, 28(4), April 1980.