"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.

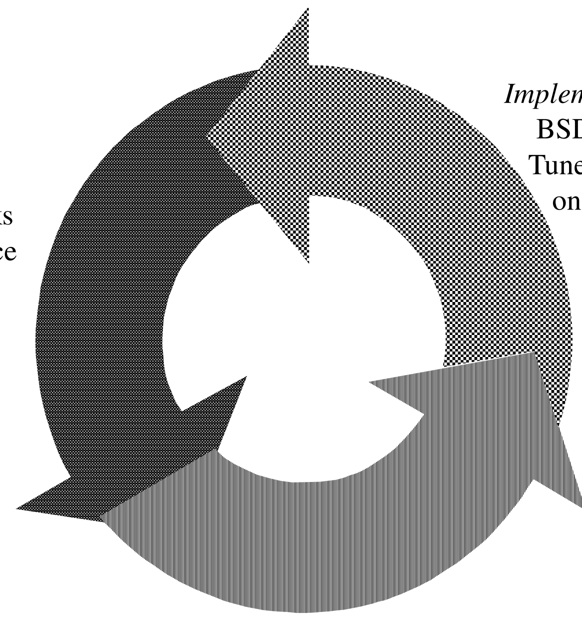

Figure 3.1 Three phase research methodology involving analysis, simulation and implementation.

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.

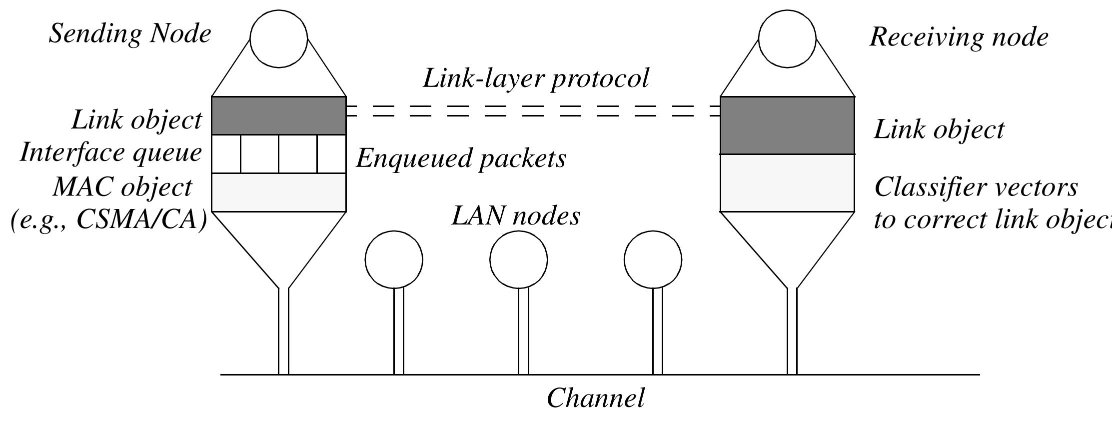

Figure 3.2 Schematic representation of LAN objects in *ns*.

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.

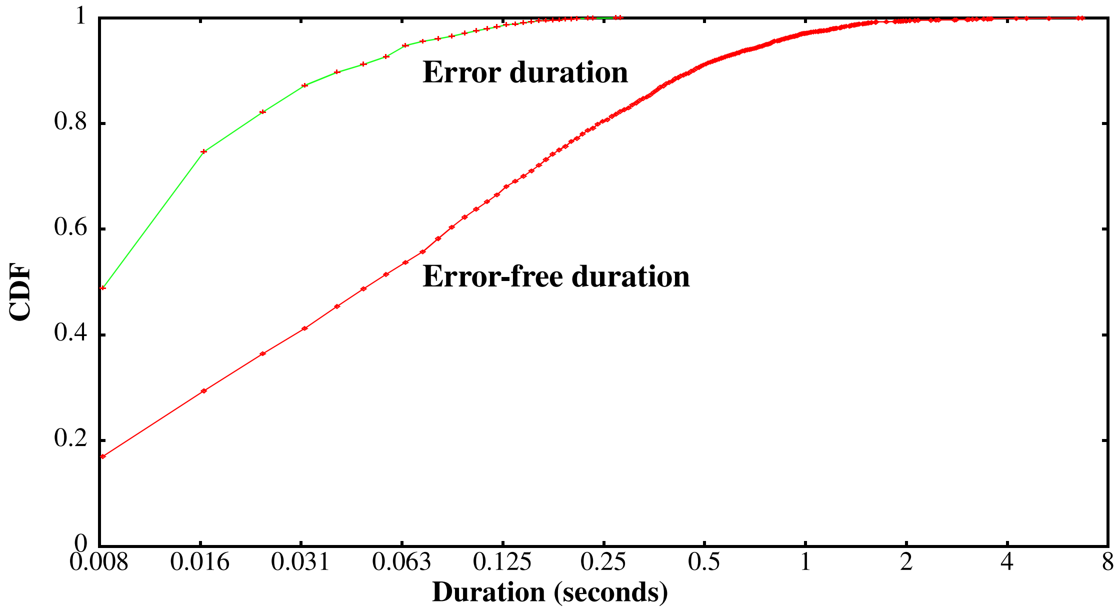

Figure 3.3 Cumulative Distribution Functions (CDF) of empirically measured error and error-free durations. The CDF was computed over of 100,000 packets sent over 800 seconds in the Reinas outdoor wireless network.

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

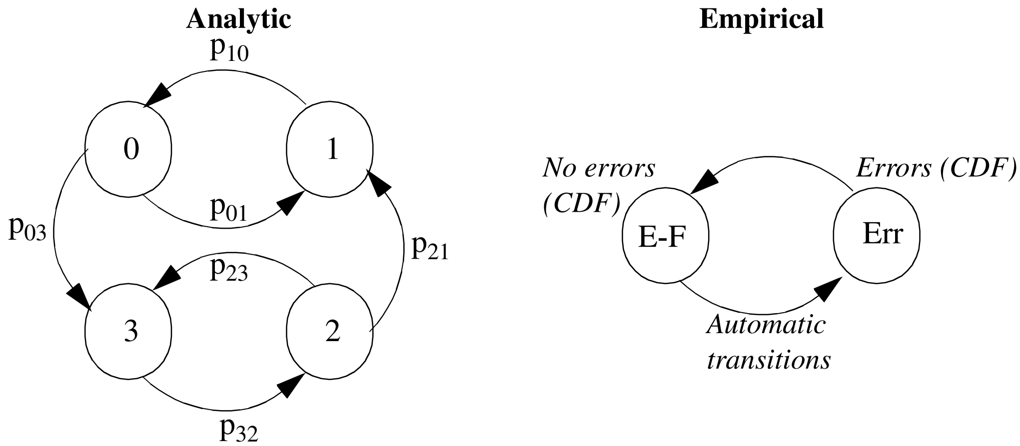

Figure 3.4 Conventional multi-state analytic error models (left) and empirical distribution-based models (right). In analytic models, the different states correspond to different base error rates, and the arc probabilities (p_{ij}) govern state transitions. Our empirical models are easy to fit to observed 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.

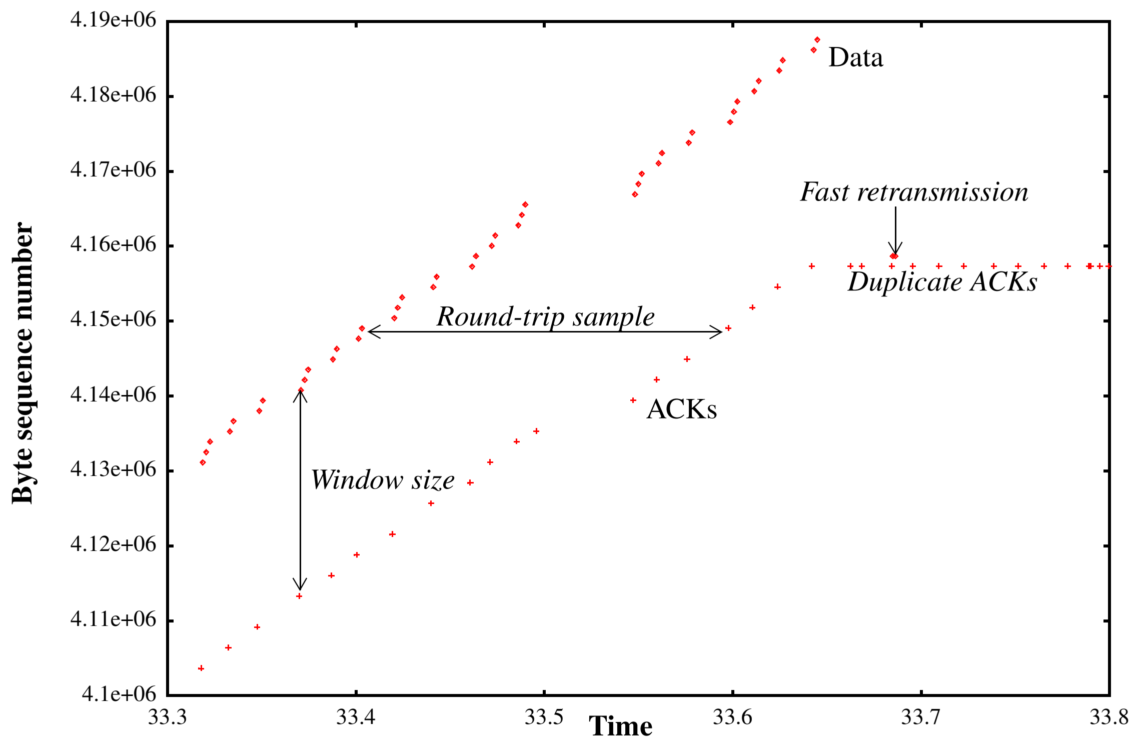

Figure 3.7 Sequence number plot of a portion of a TCP transfer.

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.