An Internet-style Approach to Managing Wireless Link Errors

David A. Eckhardt

May 2002

CMU-CS-02-141

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:
Peter Steenkiste, Chair
Allan Fisher
Hui Zhang
M. Satyanarayanan
Reiner Ludwig, Ericsson Research

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA) under contract F19628-92-C-0116 and DARPA/ITO monitored by NRaD under contract N66001-96-C-8528, by an Intel Corporation Graduate Fellowship and by the Digital Equipment Corporation.

Keywords: Mobile computing, wireless networking, wireless link errors, wireless scheduling, fair scheduling, link-level error control, adaptive error control, forward error correction, ARQ, wireless fair scheduling, effort-limited fair, ELF

The day is short, and the work is great,
and the labourers are sluggish,
and the reward is much,
and the Master is urgent [...]

It is not thy duty to complete the work,
but neither art thou free to desist from it [...]
(Avot: 2:20-21) [58]

Acknowledgements

Let me begin by saying that this journey turned out to be longer and more complex than I had expected. In August of 1989 I arrived in Pittsburgh to begin my graduate studies. I had the fortune to share a truly beautiful apartment with a gentleman and a scholar, Josh Bloch. At the time he was finishing up his dissertation while working for a startup. Then, as now, I was impressed by his design sense, sense of humor, breadth of interest, and diversity of talents. Josh's Sunday activities might begin with consuming the New York Times while regaling me with a blow-by-blow account of discovering a bug in a thread package (note to posterity: spin-waiting for a mutex on a multiprocessor is probably counterproductive), continue through changing the oil in his Toyota Tercel, and culminate in levelling his landlord's shrubbery with a Milwaukee Sawzall. In short, I had met the first of many impressive role models at Carnegie Mellon's School of Computer Science. While I aspired to emulate these role models in many ways, the only objectively measurable triumph I can point to is having exceeded them all in terms of the duration of my graduate career.

I owe many other people thanks, which I will attempt to offer in roughly chronological order, beginning with my parents, Bob and Carey Eckhardt. Thank you for providing me with a caring, stable, education-focused family life, for supporting my tendency to break loose from the curricula envisioned by others, and for your continued close companionship. Also, thank you for knowing exactly which things belonged in the "goals" and "objectives" boxes on those forms.

I am grateful for three true friendships, dating from elementary school and junior high, which persist to this day. Ben, Chris, and Angela—thank you for the times you told me I was wrong, the times you told me I wasn't, and all the other times too.

In some sense this dissertation is a distressingly incomplete repayment of the debt I owe to Robert Michael Owens, known variously as "Bob," "Mike," "Mikey," and "the guy with the bad attitude." He and his research partner Mary Jane Irwin, through vehicles as concrete as accounts on a PDP-11 running Version 7 Unix and as abstract as that "bad attitude" (which I carry in my heart every day), were responsible for involving me in the community of Computer Science before, during, and after my undergraduate studies at Penn State. I wish I could present him with a copy of this dissertation.

Though it seems a little odd in these days of Internet-connected Coke machines, I also owe thanks to the Gifted Program at the State College Area High School for harboring a terminal and a 300 baud dialup modem, which turned many study halls into precious time online.

Many people contributed to making my time at Penn State meaningful. Among those who particularly contributed to preparing me for graduate school were Bob, Janie, Joe Lambert, Jonathan Goldstine, and Jesse Barlow. Romayne Bernitt, the "UREP secretary," always brightened my day, and I always knew that her partners in the main office, Helen DeFurio, Flo Thomas, and Vicki Keller, were rooting for me too. Over the years, I have had excellent luck in housemates, which began with Ron Bayline and Dean Batten. My friend Sohail Malik has helped me in countless ways over the years, while serving as an example both professionally and personally. I must thank Larry Spence, my Political Science advisor at Penn State, for waiting so patiently for me to emerge from the distractions of technology to confront my true destiny (and also for those Friday afternoons).

I came to Carnegie Mellon because it seemed like the kind of place I wanted to be a part of. I was right. The tradition of collegiality in the School of Computer Science makes it a gem among the academic programs I am familiar with. I guess first of all I should thank Stewart Clamen, for that thing I shouldn't really know about. I want to thank Xuemei Wang for helping me pass AI. My life was enriched by Vince Cate in many ways, including his cheerful contributions to those Nectar demos, his freewheeling curiosity, and his generosity in letting me sleep on his floor for that week while the paint dried. Thank you, Zoran Popović (aka "Dorn Povic"), for sustenance, companionship, penetrating insight (wry and otherwise), and occasional visitations involving truly excellent pastry. Ian Davis bared his soul to me during his post-Ph.D. job search, and his final decision inspires me still. From my first advisor, H.T. Kung, I learned several important attitudes toward research. Over the years various people went to bat for me. Due to necessary veils of secrecy I don't know who all of them are, but I am pretty sure I owe Bob Harper and Thomas Gross. Paul Bennett and the rest of the espresso machine gang are proof that the legendary cheese co-op spirit is alive and well. Jeannette Wing truly cares about the Ph.D. program, and it shows. I owe quite a bit to Mike Accetta, Howard Wactlar, Daryl Clevenger, Dale Moore, and Bob Cosgrove, for employing me inside the community while I brought this odyssey to completion. Karen Olack always figured out how to make my student account balance, no matter what mysterious thing happened to it on the other side of campus. She also figured out how to arrange things so that, in the spring of 2000, I was able to simultaneously fulfill almost every possible CMU role: student, alumnus, staff member, and faculty member. Through it all, Sharon Burks managed to herd all the cats and jump all the tigers through the hoops, cheerfully.

I should thank Cindy Bloch for being a great housemate (even though she did kidnap Josh). Larry and Nancy Jacobs were responsible for many relaxing weekend evenings (and also that time we bent the copper pipe). I am grateful to Dan Bloch for his ongoing stream of Unix questions and `vi` answers (and, of course, the beer).

Among my fellow students I must thank Lily Mummert, for her professionalism, concern, counsel, and dogs; Corey Kosak, for our collaboration and for being who he is; and John Pane, for introducing me to Palm OS, the computing platform that has run my exobrain since the end of 1996. Bruce Lowekamp, Joseph O'Sullivan, Matt Zekauskas, and Darrell Kindred helped me negotiate with LaTeX. When this dissertation manages to express ideas clearly, it is frequently because I emulated the structural plans used by Brian Noble and Lily Mummert in theirs.

As for the Ph.D. thesis process, I must begin with my advisor, Peter Steenkiste. There is no way this dissertation would have been completed without him. In many subtle ways, which I discovered over time, he was the ideal advisor for me. Other ways were less subtle, such as allowing me to choose a topic far from what his research group was working on. Also, he *never* gave up. The other members of my thesis committee were great, too. Hui Zhang was patient when I showed up in his office to rave about wireless packet scheduling. Allan Fisher was as dependable, insightful, and gentle as I expected. Satya's contributions were above and beyond the call of duty. Aside from the inspiration provided by the Coda and Odyssey projects, the long-term loan of two laptops was critical. I have to thank Dan Duchamp, my original external member, for years of patience and for the sage advice he provided at my proposal (I got bogged down exactly where he warned me not to). Reiner Ludwig, when called to serve as my final external member, was consummately professional and otherwise ideal.

Robert Baron of the Coda distributed filesystem project wrote the WaveLAN device driver which I extended (and later re-wrote); members of the Coda group lent their time and laptops to provide interference. Both my sister Amy Eckhardt and Professor Doug Tygar graciously made long-term loans of cordless phones used to produce error traces. John Stonick, while he was at CMU, helped me understand details of the WaveLAN modulation scheme. Phil Karn provided inspiration, useful advice, and the Reed-Solomon codec implementation I used. Jamshid Mahdavi and Matt Mathis, then at the Pittsburgh Supercomputer Center, and Vern Paxson of the Lawrence Berkeley National Laboratory deepened my understanding of TCP retransmission timers. David Johnson and David Maltz gave me many useful comments. Ion Stoica provided extensive comments on my scheduling proposals. David Matsumoto discovered an embarrassing arithmetic error.

Equipment support was provided by the Digital Equipment Corporation (may it rest in peace); the Intel Corporation provided me with equipment, a full-year fellowship, and the opportunity to learn about the "real world" while I was briefly an insider (particular thanks to Haim Sadger, Bob Galin, Kevin Kahn, Farhad Mighani, and Yiftach Tzori). My life was made much easier by Shawn Ostermann (tcptrace), the Network Research Group at the Lawrence Berkeley Laboratory (tcpdump), and Larry Wall (perl). My brother Jon Eckhardt gave me last-minute help with a chart. From proposal to completion, NetBSD was a solid platform.

The staff of the Cyert Center for Early Education made it possible for me to abandon my young daughter, over and over, secure in the knowledge she was being not only taken care of, but educated. I'm sure the university would be more productive if we could all figure out how to extend this wonderful program, or sibling programs, to all CMU kids.

As for my wife, Xiaolin Zang... in addition to everything you've given me pursuant to your decision to marry, and entrust your son to, an alien-country-person, I must thank you for letting me stick with this project even after it became painfully clear that there was just no way for me to predict when it would actually be done.

Finally, I'm sure that, the minute *after* this pile of paper lands with a "thunk" on Sharon's desk, I will realize I left out somebody I should have thanked. Please accept both my sincere apology and my belated thanks (through this one level of indirection).

Abstract

As wired computer networks support increasingly sophisticated applications and wireless local area networks become ubiquitous and fast, it is more natural for users to seek a "wireless Internet" experience that is qualitatively the same as that provided by the wired Internet. However, wireless LANs pose two fundamental challenges to this vision. Harsh and dynamic error environments challenge end-to-end adaptation at the transport and application layers. In addition, dynamic and location-dependent errors challenge the notion of "fair" scheduling of flows sharing a wireless link.

This dissertation advances the claim that a combination of protocol-blind link-level error control and error-sensitive link scheduling effectively addresses these two challenges.

The first step is a bit-level trace-based analysis of the error environment provided by a particular wireless link technology (AT&T WaveLAN I) in the face of attenuation and interference. Based on the insights revealed by this analysis, the next step is designing an adaptive link-level error control module. Finally, we propose a new notion of fairness appropriate for error-prone wireless links. This new scheduling approach balances application sensitivity to error-induced throughput changes against the need to preserve link efficiency.

The proposed mechanisms are evaluated by deploying them in an actual operating system, running on real hardware, and subjecting them to trace replay of the error environments we observed. Particular attention is given to the interaction between link-level error control and TCP's end-to-end error and congestion control mechanisms. The result is a system that can noticeably improve application-level throughput in a wide variety of error environments and can sensibly allocate limited and dynamic network capacity among network flows.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

During the 1990's the Internet came of age, becoming a hugely successful infrastructure for delivering electronic mail and hypertext to any computer plugged into a wall. Now it is expanding simultaneously in two directions. First, as the "last mile" connectivity evolves from analog dial-up telephone lines to high-speed links, it is increasingly practical to deliver high-bandwidth and/or real-time audio-visual content. Second, as the exponential growth in processor power is applied to radio communications, cellular wireless networks are increasingly ubiquitous and gaining enough per-user capacity to support data and multimedia traffic in addition to voice. However, the wireless environment poses fundamental challenges which make it difficult to offer wireless users the same fidelity of experience that they expect based on using wired networks.

The first major challenge is that wireless transmissions suffer from attenuation, fading, and interference from competing transmissions. These can result in significant rates of transmission errors, which may manifest themselves as bit corruptions within a packet, partially-received (truncated) packets, or packets that are sent but not received. These misfortunes must be overcome if we are to provide applications with a high-quality network environment.

The second major challenge is that the severity of wireless link errors may vary suddenly and dramatically due to unpredictable changes in the surrounding environment. This impedes applications that either require a certain quality of network service or that try to adapt their communication strategies to prevailing network conditions.

This introductory chapter is structured as follows. The relationship between the Internet architecture and wireless link errors is discussed in Section 1.1. The thesis statement is presented in Section 1.2 and a brief overview of the supporting results presented in later chapters is found in Section 1.3. Section 1.4 lists background assumptions used to direct the research agenda. Section 1.5 discusses research efforts that are related to the thesis work as a whole; other, more-detailed examinations are made in the body chapters. Background information on the wireless local area network technology used in the course of this research effort, WaveLAN I, is found in Section 1.6.

1.1 Can the Internet Architecture Manage Wireless Errors?

Wireless links differ from traditional "wired" (including fiber-optic) links in two ways that challenge current implementations of the standard Internet protocol suite. First, on wireless links packet loss or corruption due to transmission errors is not rare. This calls into question the standard Internet assumptions that transmission errors should be corrected by transport-level protocols at end systems [119, 24] and that end-to-end packet loss typically indicates network congestion [60]. Second, end systems sharing a wireless link may experience different error rates depending on error conditions, such as interference, at their different locations ("location-dependent errors"). Traditional Internet link-scheduling mechanisms (ranging from FIFO through Weighted Fair Queueing) assume that packet transmission failure is not only rare but also independent of which station the packet is addressed to. Violating these assumptions calls into question the meanings of "fair" scheduling, per-flow quality of service, and even looser notions such as service level agreements.

Meanwhile, much of the Internet's success to date has resulted from its architecture, a *lingua franca* unifying disparate network types. The original architects of the Internet were faced with multiple incompatible networks and a desire to enable communication among them [32]. Much of the Internet's success has come from their approach of standardizing base-line, widely-usable mechanisms (such as addressing and best-effort delivery) without specifying "the right way" to perform every useful function at every network node. In particular, the Internet architecture has encouraged addressing problems locally when there is compelling local knowledge and deferring solutions to end systems when there is no known generally-applicable local solution.

For example, maximum transmission unit (MTU) size is a per-link decision because a specific link may have size constraints different from the rest of the Internet. Queue service order and queue-drop policies are per-link, which makes sense since only a given link knows its instantaneous traffic load and typical load patterns. Nearly every link includes some form of error detection, such as per-character parity or a per-frame checksum, tuned to match that link's expected corruption patterns; some links include link-specific error control mechanisms such as error coding or retransmission.

In the other direction, flow control, congestion control, in-order delivery, retransmission policies (including timing and persistence), application-visible framing, authentication, and privacy have been deferred to end systems, either because end-system applications have widely divergent needs in these areas or because no generally-accepted per-hop solution is known.

In some cases where network support for a feature is clearly required but not fully understood, such as Quality of Service, the Internet architecture includes coarse-grained hinting, such as the Differentiated Services (previously "Type of Service") header field [91].

The relationship between the Internet architecture and wireless link errors can be broken down into two main questions:

1. To what extent should the responsibility for overcoming wireless link errors reside at

end-system hosts versus wireless links, and

2. Do parts of the existing Internet protocol infrastructure need to be replaced or reworked? Might we need to replace or upgrade transport protocols so they can work across wireless links (Section 4.7.8) or might wireless links need to "fix up" the operation of specific end-to-end protocols (Sections 4.7.9 and 4.7.10)?

Ideally, wireless link errors could be addressed without changes to non-wireless infrastructure and without special support for every combination of transport protocol and wireless link type.

1.2 The Thesis

It is possible to effectively address wireless link errors locally, through a combination of protocol-blind link-level error control, which lessens the damage, and error-sensitive link scheduling, which ensures sensible outcomes in response to unrecoverable link capacity loss.

First, we believe that most wireless link errors can and should be handled locally by using per-link knowledge of the error environment to design an appropriate error coding and retransmission scheme. Second, we believe that, while link-level error control can insulate end-to-end traffic flows from the worst effects of the dynamic wireless error environment, severe and location-dependent errors require a new notion of the "fair share" of a wireless link. We propose and evaluate *effort-limited fairness*, which allows wireless links to consider both per-flow throughput stability needs and whole-link efficiency concerns. When we combine adaptive link-level error control with error-sensitive link scheduling, unmodified existing end-to-end protocols can effectively use wireless links despite dynamic and severe error environments.

This thesis was investigated by focusing closely on one particular wireless LAN environment, WaveLAN I (see Section 1.6). In that context, we began with a detailed characterization of how this wireless LAN behaves in response to typical environmental challenges. We collected detailed traces of the system in operation. Analysis of these traces revealed specific challenges for reliable data networking and suggested possible remedies. Next, we designed and implemented a multi-layered network protocol stack, running in a production operating system kernel on real hardware, to test mechanisms and policies for responding to these challenges. Finally, we used trace replay to evaluate the quality of our proposed solutions.

1.3 Overview of Results

The evidence for the thesis is established over the course of four chapters. A brief summary of each one will provide the reader with a coherent vision of the remainder of the dissertation document.

1.3.1 Characterizing Environmental Challenges

Chapter 2 summarizes how WaveLAN responds to a variety of environmental threats, such as various interfering radiation sources and obstacles. From this investigation we learn that common environmental challenges pose serious threats to reliable wireless communication.

Some sources of observed errors, such as signal absorption by human beings and interference from mobile communication appliances, suggest that wireless link errors will frequently be dynamic on a variety of time scales.

Measurements reveal that error environments are sensitive to a wide variety of factors, including the construction materials used by different buildings.

The results in this chapter were previously published in the proceedings of ACM SIGCOMM '96 [38] (copyright © 1996 Association for Computing Machinery).

1.3.2 Trace-based Evaluation of Adaptive Error Control

Chapter 3 deepens this investigation by selecting detailed error traces from several particularly interesting error environments and simulating the effectiveness of various link-level error control strategies.

This investigation yields two important results. First, efficiency in the face of dynamic errors demands adaptive error control strategies—while static policies may work well in some error environments, they perform poorly or fail in others. Second, it is feasible to construct strategies that are both simple to implement and perform well in various dynamic error environments.

This study was selected for a special issue of *Mobile Networks and Applications* (MONET), published by Kluwer Academic Publishers, on the topic of adaptive mobile networking [40].

1.3.3 End-to-end Evaluation of Local Adaptive Error Control

Chapter 4 addresses two architectural questions about deploying adaptive error control.

The final authority on error control is, necessarily, some end-to-end protocol between communicating entities [119, 24]. However, for practical reasons, every network link provides some per-hop error control functionality, if only byte parity or a simple frame checksum. This conflict poses the question of how much effort should be devoted by a wireless link to overcoming its own errors and how those efforts will interact with end-to-end error control. This question is important because, if the primary responsibility for wireless link errors will belong to end systems, potentially every end-to-end protocol would need to become "wireless aware."

If any error control is to be done locally, this raises the question of whether specific mechanisms will be required to support each end-to-end transport protocol or whether this can be done in the protocol-blind fashion typical of the Internet to date.

We argue that it is *desirable* for wireless link errors to be addressed on a per-link basis and for this error control to be independent of the end-to-end transport protocol in use. We demonstrate that this approach is *feasible* by examining the interactions between link-level error control and TCP. Chapter 4 also expands on Chapter 3's arguments in favor of adaptive error control strategies by retargeting our error environment traces from the realm of abstract throughput simulation to a prototype LAN implemented in an open-source operating system kernel. The result is a confirmation of our per-link protocol-blind adaptive error control approach.

This work was first presented at the Sixth IEEE International Conference on Network Protocols (ICNP '98) [39].

1.3.4 Effort-limited Fair (ELF) Wireless Link Scheduling

Link-level error control techniques can increase the effective throughput available over an error-challenged link by converting many end-to-end packet losses into mild throughput reductions. For example, if a wireless link begins to lose half of the packets transmitted across it, retransmission might transform the 50% end-to-end packet loss rate into a 0% end-to-end packet loss rate, but the available throughput would still drop by half. In a different situation, error coding might reduce the packet error rate due to bit corruption from 90% to 5% but coding overhead could cost 20% of each packet body. The result would be that throughput would increase from 10% to 76%. In each case substantial link capacity is irreversibly lost.

If some application flows need a certain amount of throughput to function correctly, or seek to adapt their behavior according to throughput availability, wireless link errors become a scheduling problem. That is, a wireless link scheduler can transfer throughput losses from some flows to others by giving error-prone flows extra air time. This may be done to ensure a particular quality of service for some flows or to improve the ability of other flows to track and adapt to changing conditions. Which flows deserve how much extra air time support is a complex policy issue.

Once a policy has been arrived at, design problems remain. Many link schedulers make complex, long-term decisions about the service a flow will receive. Some flows are allocated a specific fraction of a link's throughput based on administrative considerations such as cooperative bandwidth purchases. Other flows may request particular service guarantees and are subject to the outcomes of complex admission control algorithms. These pre-computed decisions must be reconsidered when environmental conditions change in a way that drastically increases the error rates of one or more flows. If the error environment changes frequently and dramatically, link scheduling algorithms must be redesigned around this central fact of dynamic capacity.

Chapter 5 describes a new approach to wireless link scheduling, "Effort-limited Fairness," which builds on the existing Weighted Fair Queueing paradigm while intelligently managing highly dynamic wireless link capacity. We translate this new policy framework into a scheduler algorithm and evaluate its behavior by both simulation and deployment in our network prototype.

An early version of this chapter was presented at IEEE INFOCOM 2000 [41].

1.3.5 Result Summary

The result of our investigation is a demonstration that wireless link errors can be addressed without upheaval to the existing Internet architecture. Even severe and dynamic errors can be ameliorated by protocol-blind adaptive link-level error control when it is designed to address the error characteristics of a particular link. Our investigation presents evidence that interactions between per-link and end-to-end error control are not troublesome. Effort-limited fair scheduling yields sensible and intuitive results for flows in the face of enormous error rate variability. This scheduling approach can benefit not only flows with specific reservations, but also the best-effort flows that make up the vast majority of Internet traffic today.

A summary of this dissertation recently appeared in a special issue of *Wireless Communications and Mobile Computing* (JWCMC), devoted to the topic of reliable transport protocols for mobile computing [42] (copyright © 2002 John Wiley & Sons, Ltd.).

1.4 Scope

This work embodies a variety of assumptions, falling into two main categories. We assume certain plausible architectural features that help us focus on both error management techniques and guide our evaluation of them. In addition, we omit detailed consideration of certain important but complicated architectural areas due to time and space constraints.

First, we assume the following architectural features:

- *non-minimal mobile hosts*:

  When this research effort began, we chose to focus on mobile nodes equivalent in power to laptops (at the time, roughly a 33 MHz processor with 8-32 megabytes of RAM), despite their limited portability. This decision, which was based on our belief in the power of technological growth [86], was in conscious contrast with research into "lightweight" mobile hosts [121, 133]. Over time platforms possessing this degree of computational power, which provides benefits such as operation even while disconnected from a network, flexibility of data formats and error control strategies, and a large, evolving software base, have shrunk to pocket size (as of 2001 multiple cellular telephones with full-blown HTML browsers were available on the U.S. market).

- *cellular radio architecture*:

  Cellular radio networks [109] re-use the radio frequency spectrum by partitioning a large area into smaller "cells" and assigning a subset of the spectrum to each cell. Each cell, or small group of cells, typically has its own connection to a wired backbone network. Cellular systems are often designed with the expectation that traffic between mobile hosts in a single cell is rare; this assumption can simplify both hardware and software. At any time, a single cell contains a known number of mobile clients, and the cell's base station can assign resources, such as bandwidth and transmission power, to these clients according to their needs. Advances in radio hardware, signal processing, and antenna system design are steadily increasing the practical per-cell bandwidth at the same time as decreasing hardware prices make it practical to devote this bandwidth to smaller cells (i.e., fewer mobile hosts).

Ad-hoc networks [27] are an alternative organization in which a large, geographically dispersed collection of mobile hosts unite to form a network infrastructure. There is no formal division into cells with its attendant carefully engineered frequency re-use; distributed medium-access control protocols allow a single channel to be re-used in a randomized fashion. At present, ad-hoc networks are an ongoing research area and there is is little experience with providing ad-hoc Quality of Service support for applications. For these reasons, this work considers cellular data networks.

- *shared channel*:

  This work focuses on a shared radio channel of 1 to 20 Mb/s due to the widespread commercial availability of products in this range, including the IEEE 802.11 Wirless LAN standard [70]. Having multiple hosts share access to a single channel may not obviously be desirable. For example, the Qualcomm CDMA Cellular Phone System [118, 106, 64] uses CDMA to provide each handset with a dedicated channel to and from the base station. This approach requires less-intelligent handsets and is cheaper since the per-channel hardware at each base station is operating at a low bit rate. But multimedia networks with computationally powerful hosts may be better served by a system which allows a more flexible division of bandwidth among hosts, and which allows multicast transmission of data such as video streams. A shared channel arbitrated by some form of time-division multiplexing, such as IEEE 802.11 or HiperLAN 2 [104] should provide these benefits and greatly reduce the cost of base stations, since they would need only one high-speed data path (correlator, codec, modem, transceiver) instead of many. It seems likely that a single channel (i.e., wireless cell) will typically be occupied by between 1 and 100 mobile hosts at any time.

- *IP version 4*:

  During implementation of the prototype network evaluated in this work, we used version 4 of the Internet Protocol (IP) [101]. However, only minor details would need to be changed in order to support IPv6 or even non-IP-based networks.

Second, we do not devote significant time to

- *handoff planning and management*:

  Allowing a mobile host to move from one radio cell to another requires substantial infrastructure support. For example, the inter-cell backbone must track the location of every mobile host in order to route data to and from it. Furthermore, if a cellular network wishes to provide Quality of Service support, it may be necessary for it to carefully balance mobile hosts across cells in order to ensure that each cell has enough resources to support its mobile clients. In addition, it may be necessary for adjacent cells to carefully coordinate the transfer of a mobile host between them. There is substantial ongoing research in the area of handoff support and management [1, 75], and this is somewhat orthogonal to the issues of how best to support mobile hosts within a single cell. Therefore, this work omits support for, and consideration of, handoff.

- *inter-network blame assignment*:

  Whenever application data needs to cross multiple networks, such as a campus cellular radio network, a regional Internet Service Provider, and the Internet backbone, and the application expects a particular quality of service, there is the possibility that the application will receive lower quality than it needs, even if it has carefully reserved resources on each network it is using. When this happens, the first step toward remedying the problem will be determining which network is not meeting its stated level of quality. This will require networks providing quality of service to track what level of quality they are actually providing to each application and to share that information with applications or with other networks. While this sort of information is similar to information that will be necessary to manage intra-cell wireless error control strategies, this work does not consider the more general problem of inter-network blame assignment.

1.5 Related Research Efforts

While more-detailed reviews of related research will be presented in the following chapters, a brief overview of several efforts in the area of wireless network architectures will clarify the high-level differences between this work and that of others.

1.5.1 IEEE 802.11

The IEEE 802.11 wireless LAN effort [70] began as a "second generation wireless Ethernet." The primary goals were to replace proprietary wireless LAN technologies with multi-vendor interoperability and to apply research results to improving Medium Access Control performance. The 802.11 standard provides a simple packet-level retransmission facility and supports several levels of physical-layer error coding. An optional "Point Coordination Function" provides rudimentary support for link scheduling by allowing a base station to allocate air time among stations by polling.

If we consider link-level error control, there are both similarities and differences with this work. For example, both the 802.11e draft standard [59] and our prototype use Reed-Solomon block codes to recover from errors experienced by an underlying direct-sequence spread spectrum (DSSS) physical layer; both the widely-used 802.11b standard [69] and our prototype segment and reassemble network-level datagrams into smaller link-level frames. Chapters 3 and 4 could be used as trace-based experimental evidence for setting certain 802.11 MAC parameters such as retransmission counters. However, our results go beyond this to investigate the utility of dynamic frame size adjustment and the feasibility of designing adaptive error control policies that work across a wide range of error environments. Also, while 802.11b supports adaptive adjustment of the transmission symbol rate, "the algorithm for performing rate switching is beyond the scope of this standard." The demonstrated utility of our FLEX adaptation policy (Section 3.3) could prove useful in this regard.

The original 802.11 specification said very little about how a base station should perform link scheduling, and we are not aware of a commercial 802.11 implementation that includes PCF polling. The IEEE 802.11e working group [59] is upgrading the standard to add Quality of Service support including both traffic specifications and scheduling. It is likely that, once again, the exact scheduling algorithm will not be specified. One feature of the current 802.11e draft is priority-based scheduling. In Chapter 5 we will argue that this approach has limited applicability. Furthermore, the ELF approach we will advocate has an architectural advantage significant to the Internet: an ELF scheduler naturally improves the performance stability of the Internet's "bread and butter" best-effort traffic without requiring it to declare a traffic specification and pre-allocate link throughput.

1.5.2 Wireless ATM

For approximately a decade researchers have investigated ways to extend the ATM network protocol family [5] to mobile wireless links [22, 116, 128, 67]. While this effort necessarily faces many of the same problems that "wireless Internet" research does, there are important differences.

The ATM quality of service model is very complicated. A flow may be queued according to one of several disciplines: UBR (unspecified bit rate), roughly equivalent to traditional best-effort delivery; ABR (available bit rate), similar in scope to ECN [110]; CBR (constant bit rate); or VBR (variable bit rate). These disciplines involve traffic specifications and shaping functions with many parameters. ATM forbids out-of-order delivery and encourages flows to specify delay and jitter requirements. Because ATM data payloads are 48 bytes and packetized speech is an architecturally significant flow type, there is a tendency for flows to consist of tiny evenly-spaced packets. This poses significant challenges to medium access control (MAC) protocol designs and link schedulers.

Wireless ATM inherits a focus we will refer to "hard QoS." By this we mean several things. First, flows are encouraged to express detailed quality of service requirements and, assuming flow set-up succeeds, expect that these requirements will be met precisely for long periods of time. Second, flows sharing a link may specify QoS requirements that differ not only in quantity but in kind–a link scheduler must be prepared to simultaneously and faithfully execute contracts specified in ABR, CBR, and VBR terms, with per-flow throughputs varying by multiple orders of magnitude. The computation required to do this, split between connection setup and real-time link scheduling, may be substantial. Finally, record-keeping, billing, and periodic status reporting may be complex as well.

Our work focuses on a simpler notion of quality of service. First, we believe it is important to do a good job for best-effort traffic. In Chapter 5 we will demonstrate a way to balance efficiency and fairness concerns that supports best-effort flows as well as constant bit-rate flows. On the other hand, our scheduling work is not as feature-rich as that of others; in particular, we do not explicitly manage delay jitter.

While experimental Wireless ATM networks have been constructed [100, 115], the research to date does not appear to have resulted in standardization or commercial products.

1.5.3 HiperLAN

Since 1991 the European Telecommunications Standards Institute has been developing a wireless data infrastructure known as HIPERLAN (High Performance Radio Local Area Network). The HiperLAN 1 functional specification was released in 1996 and updated in 1998 [103]; however, that effort did not culminate in released products [57]. The HiperLAN 2 definition [104] was released in 2000; commercial implementations are expected sometime in 2002.

HiperLAN 2, like Wireless ATM, has a "hard QoS" focus [61] beginning with multiple data-link connections per station. A link adaptation function uses multiple modulation systems and error correction codes to trade off throughput and reliability. Power control is used for a mixture of technical and regulatory reasons.

It is difficult to predict whether our trace data would generalize from WaveLAN to HiperLAN 2, since the latter uses a different modulation approach and operates in a different frequency band. However, we expect adaptive link-level error control to be as important to performance for HiperLAN 2 as it has been for our prototype network. We believe that severe and dynamic error conditions will be particularly challenging for "hard QoS" schedulers since they will be in the position of writing very detailed contracts and being unable to meet them; the ELF approach could serve as a model for "what to do when you can't do what you said you would."

1.5.4 Reiner Ludwig's "Cross-Layer Interactions" dissertation

The doctoral dissertation of Reiner Ludwig [78] is relevant to this work because it investigates in depth the interactions between link-level and end-to-end error control. While that dissertation and this one overlap in areas and display substantial philosophical compatibility, they differ in terms of network environment, experimental approach, and contribution areas.

Ludwig's work takes place in a different network environment. The radio link studied is that of GSM cellular phones [44]. GSM's RLP provides a 9.6 Kbit/s channel. Error control includes a fixed FEC (Forward Error Correction) code, cross-frame interleaving (which induces a 90 ms latency), and link-level retransmission (which can delay a packet up to 2.5 s).

That work, like this one, is based on real-world trace data. However, one of his research contributions is an infrastructure for simultaneously tracing, in a correlated fashion, multiple protocol layers. This allows him to study cross-layer interactions in finer detail. Ludwig's error traces consider not only errors due to hazards of the radio channel but also handoff-related errors. In the other direction, the error traces presented in Chapter 3 are detailed enough to allow investigation of adaptive error coding.

Both dissertations make contributions to the understanding of link-layer error control and continue on to suggest changes in higher protocol layers. Based on his detailed multi-layer traces, Ludwig proposed modifications to the TCP transport protocol, where this dissertation focuses instead on the relationship between link errors and packet scheduling.

Further details of the relationship between these research efforts will be covered in Section 4.7.2.

1.5.5 IETF PILC working group

The Performance Implications of Link Characteristics ("pilc") working group of the Internet Engineering Task Force has produced several documents containing a variety of advice for designers of both end-to-end protocols such as TCP [34] and specific network link technologies, including link-level retransmission [45]. In addition to communicating consensus on design issues when it is available, these documents are also designed to expose technology implementors to a variety of ongoing research efforts when consensus has not yet been reached. They therefore frequently refer to academic research, and [45] cites an earlier version of Chapter 4.

1.6 Experimental platform: WaveLAN I

The wireless LAN employed in this work is WaveLAN I [127], originally known as NCR WaveLAN, then AT&T WaveLAN, then Lucent WaveLAN. WaveLAN I has been superseded in the marketplace by Lucent WaveLAN II and other implementations of the IEEE 802.11 Wireless LAN standard [70].

Though WaveLAN I is now obsolete, it is an attractive research platform for this work. First, it represents a "competent" physical layer in the sense that it uses direct sequence modulation and antenna diversity to effectively combat multipath interference. It allows us to capture the complete bodies of packets that have experienced data corruption, which frequently allows us to determine the degree and precise nature of the corruption. This is important for designing and evaluating error coding strategies. Most existing implementations of the IEEE 802.11 wireless LAN standard [70] appear to perform the non-trivial medium access control protocol in hardware, which would significantly constrain our investigation of integrating flow scheduling with medium access control scheduling.

Several similarities between WaveLAN I and 802.11b [69] (the popular "Wi-Fi" flavor of 802.11) suggest results of this work may apply to modern systems. First, 802.11b uses a Direct Sequence Spread Spectrum modulation scheme very similar to WaveLAN I's, and the 802.11e draft standard [59] suggests adding basically the same error correction code that we evaluate in Chapter 3. Also, the original 802.11 specification includes an optional Medium Access Control (MAC) extension called the "Point Control Function" (PCF), updated in 802.11e, which would allow deployment of the ELF scheduling approach we discuss in Chapter 5.

1.7 Road Map

The remainder of this document consists of five chapters. Chapter 2 investigates the severity of various typical environmental hazards and the sensitivity of WaveLAN to details of these hazards. In addition to interfering radiation sources, we will consider issues such as building construction. We will observe that the damage inflicted by each hazard can vary widely depending on such details as object location or construction materials, meaning that the importance of each hazard varies according to situation.

Chapter 3 delves deeper by collecting bit-level error traces in selected error environments and then uses these traces to simulate the effectiveness of various link-level error control strategies. Chapter 4 investigates how link-level error control behaves “in the real world.” In particular, in the context of a running prototype system, we will evaluate the extent to which link-level error control is compatible with TCP’s end-to-end error control.

Through careful measurement and analysis, and by implementing link-level error control, we are able to greatly reduce the impact of wireless link errors. However, this only lessens the extent to which dynamic error patterns result in unpredictable application-level performance. Chapter 5 describes how a new approach to wireless link scheduling, “Effort-limited Fairness,” builds on the existing Weighted Fair Queueing paradigm while intelligently managing highly dynamic wireless link capacity.

Contributions of this work are summarized in Chapter 6.

Chapter 2

Characterizing Environmental Challenges

2.1 Introduction

Before embarking on ambitious plans to resolve problems posed by wireless links, we believe it is important to characterize the damage caused by various threats. The purpose of this chapter is to survey the major threats to WaveLAN in a typical office environment. We wish to address the following questions:

- Which potential hazards matter the most in practice?
- How sensitive is the severity of each hazard to minor environmental perturbations, such as mobile host location or building materials?
- Which hazards that affect WaveLAN are likely to affect other wireless LAN technologies in similar fashions?

We will find that in many situations communication can proceed unimpeded by errors, suggesting that heavyweight error-control mechanisms are frequently inappropriate, but that in some situations errors are significant enough to demand careful attention if communication is to proceed at all. For example, WaveLAN can communicate effectively through multiple walls, but is significantly affected by certain cordless telephone technologies. Once we have identified which error sources require more-detailed investigation, the next chapter will employ a trace study to characterize error patterns down to the bit level, determine how dynamic these errors are, and evaluate adaptive error control solutions.

The remainder of this chapter is organized as follows. In Sections 2.2 and 2.3 we present a brief summary of the sources of wireless errors and our methodology for characterizing the error environment. We then present the results of our study: characterization of in-room, line-of-sight communication (Section 2.4), measurements of the errors caused by passive obstacles (Section 2.5) and competing radiation sources (Section 2.6). We discuss related work in Section 2.7 and summarize in Section 2.8.

2.2 Sources of Wireless Errors

For the purposes of this study, we organize the possible sources of errors in a wireless network into three groups.

The first group contains error sources we investigated and report on in this chapter:

- *attenuation.* When electromagnetic energy encounters matter, some of it is lost in the form of heat. WaveLAN can usually penetrate several walls while maintaining a very low error rate, but we found attenuation to be a significant source of errors.

- *front end overload.* If a very powerful transmitter of one frequency band is near a receiver of another band, the transmitter may overwhelm filters in the receiver and inject substantial noise. As we expect wireless computer networks to be employed in close proximity to microwave ovens and cellular phones, we have made an initial investigation into errors due to front-end overload.

- *narrowband interference.* This is due to an unfriendly transmitter occupying a small frequency band overlapping (perhaps totally) with the band we wish to use. We investigated the effects of two 900 MHz FM cordless phones.

- *spread-spectrum interference.* This is due to an unfriendly transmitter either switching between narrowband frequencies or spreading its energy simultaneously across a wide frequency band. We have investigated interference between competing WaveLAN transmitters and between WaveLAN and other 900 MHz spread-spectrum sources.

The second group contains error sources which might have an impact on WaveLAN performance, but which we did not or could not control the behavior of:

- *natural background noise.* For example, infrared wireless networks may perform poorly if they are near sources of direct sunlight. We did not attempt to measure or control for background noise.

- *multipath interference.* When electromagnetic radiation reflects off of or diffracts around objects, it takes multiple paths between the transmitter and the receiver. Since these paths are typically of different lengths, there will be destructive interference, which can greatly reduce signal strength. We have not studied how multipath effects interfere with WaveLAN, in part because they are difficult to model and study, and in part because WaveLAN is explicitly designed to resist them.

The third group contains error sources we did not consider in this chapter:

- *path loss (dispersion).* The intensity of electromagnetic energy reaching a receiver is decreased by distance even in free space. We found that WaveLAN did not suffer errors due to path loss even in large lecture halls.

- *motion.* If two communicating objects are moving with respect to each other, the frequency of the electromagnetic energy changes according to the Doppler effect. While this effect may be significant in some radio environments [56], the Doppler shift due to moving a WaveLAN unit at the speed of sound would be substantially less than the inaccuracy of the clock crystals employed by WaveLAN[127]. Hence we have not investigated errors due to motion.

- *data dependent effects.* Some modulation schemes can lose clock synchronization in the face of certain long bit patterns. We will investigate data-dependent effects as part of our bit-level trace study presented in in Chapter 3.

2.3 Methodology

2.3.1 WaveLAN

WaveLAN I [127] (henceforth: "WaveLAN") was designed to be an affordable, easy-to-install wireless extension of an existing bridged Ethernet system. Products include ISA and PCMCIA network interfaces for PC-compatible computers and stand-alone WaveLAN-to-Ethernet packet bridges. They operate in either the 902-928 MHz or the 2.4-2.8 GHz ISM (Industrial, Scientific, and Medical) license-free band. The PCMCIA units comprise a Type II PCMCIA card and an external unit a little larger and heavier than a deck of cards.

Internally, the WaveLAN interface contains a standard Intel 82593 single-chip CSMA/CD LAN controller, custom logic for signal processing and modem control, and a custom radio transceiver. The transmitter applies DQPSK modulation to a 2 megabit/s data stream, yielding a 1 megabaud signal. This signal is further modulated by an 11 chip per bit sequence (hence "direct sequence") to produce an 11 MHz wide signal which is transmitted with a power of 500 milliwatts. The receiver selects between two perpendicular antennas and multiple incoming signal paths to combat multipath interference.

The modem control unit prepends a 16-bit "network ID" to every packet on transmit, and can be set to reject all but one network ID on receive. In addition, it informs the host of the channel condition upon each packet arrival by reporting signal level, silence level, signal quality, and antenna selected for each packet. The signal and silence levels (5 bits) are derived from the receiver's automatic gain control (AGC) setting just after the beginning and end of the packet, respectively. Because the MAC protocol discourages simultaneous transmission and immediately consecutive transmissions, measuring the silence level during an inter-packet time is typically a good indication of the amount of non-WaveLAN background interference. The signal quality (4 bits) is sampled just after the beginning of the packet and is derived from the information the receiver uses to select between the two antennas. These values are summarized in Table 2.1.

As it is difficult to detect collisions in this radio environment, WaveLAN employs a CSMA/CA (collision avoidance) MAC protocol [6]. In CSMA/CD, a station which becomes ready to transmit while the medium is busy will make its first transmission attempt as soon as the medium is free, based on the optimistic assumption that it is the only waiting station.

Packet parameter	Range
Signal level	0–32
Silence level	0–32
Signal quality	0–16
Antenna	0–1

Table 2.1: WaveLAN per-packet modem information. The units of measure are not presented in [127].

If this assumption is wrong, all waiting stations will quickly learn their mistake by sensing a collision. Since WaveLAN cannot sense collisions, they result in packet losses which must be dealt with by higher layer protocols. WaveLAN CSMA/CA attempts to avoid collision losses by treating a busy medium as a collision. That is, any stations that become ready to transmit while the medium is busy will delay for a random interval when the medium becomes free. Aside from the modified MAC protocol and lower data rate, the 82593 performs all standard Ethernet functions, including framing, address recognition and filtering, CRC generation and checking, and transmission scheduling with exponential backoff.

Many of the techniques cellular radio systems [106, 64, 109] employ to re-use frequencies in nearby areas, such as power control, frequency diversity and code diversity, are easiest to employ in a point-to-point environment. Since WaveLAN follows the Ethernet protocol, where stations multicast to dynamic sets of peers rather than communicating exclusively with a central base station, it would be difficult to estimate the power required to contact a particular set of receivers for each packet transmission, and expensive or complex to synchronize with many stations each using a different spreading sequence. Perhaps for these reasons, WaveLAN does not provide the ability to vary transmit power and does not use frequency or code diversity. Instead, hosts may program their WaveLAN radio receivers to mask out weak signals by specifying minimum threshold values for signal level and signal quality. This improves throughput and may be sufficient to simulate cell boundaries in some cases even though WaveLAN is inherently a single shared channel. In addition, the "network ID" provides multiple logical Ethernet address spaces, which allows WaveLAN-to-Ethernet bridges to use standard bridge routing protocols.

2.3.2 Data Collection

To characterize WaveLAN error environment, we monitored the quality of data transfers between two identical DECpc 425SL laptops (25 MHz 80486) running NetBSD 1.0A. For different tests, we placed the PCs in different environments or added competing radiation sources.

The data transfers consisted of specially-formatted UDP datagrams. On the receiver, the kernel device driver was modified to place both the Ethernet controller and the modem control unit into "promiscuous" mode and to log, for each incoming packet, every bit and all available status information, even if the packet failed the Ethernet CRC check. We decided to collect bursts of packets at the maximum possible transmission rate (roughly 1.4 Mb/s for this machine and protocol stack), aggregating multiple bursts to form a long trial. Within each trial, packets consisted of 256 32-bit words wrapped inside UDP, IP, Ethernet, and modem framing. Within each packet the data words were identical to facilitate identification even in the face of substantial noise, and the data value was incremented between packets.

There are many reasons why a transmitted test packet might not be received intact, and we will consider them in order from the outside of the packet toward the center. First, certain errors might cause the modem unit to miss the beginning-of-frame marker, resulting in a slightly-damaged packet being totally lost. Second, it is possible for a packet to arrive correctly but be lost by the receiver due to unrelated system activity, even though we tried to reduce this to a minimum. Third, errors in the packet headers and trailers might lead the Ethernet or IP layers to discard the packet due to damage or mis-addressing. To address this, we enabled promiscuous receive and disabled automatic CRC filtering at the Ethernet level and used a heuristic matching procedure to determine whether a given packet was one of the test series. Finally, a packet may arrive but the body may be truncated and/or some bit values may be incorrect. We applied a second heuristic procedure to determine the sequence number of any packet believed to be a test packet. Since the packet body consists of a single word repeated multiple times, truncated packet bodies are ambiguous–it is not possible to know which words are missing. Therefore, we produced an estimated error syndrome (bit corruption pattern) for only those test packets which were damaged but not truncated. Furthermore, we report precise bit error figures only for errors in the data portion of the packet, as reconstructing the exact IP header (and thus the Ethernet CRC trailer) is difficult. Therefore, there are some packets which we knew to be damaged but for which we could not determine the exact number of corrupted bits. Due to these factors, our packet loss rate and bit error rate (BER) figures are necessarily only estimates.

Column Name	Meaning
Packets Received	Test packets received
Packet Loss	Percentage of transmitted test packets that were lost
Packets Truncated	Number of received test packets that were truncated
Bits received	Number of *body* bits received, rounded down
Wrapper Damaged	Number of packets with damaged headers or trailers
Body Bits	Total number of body bits damaged in trial
Worst Body	Number of bits damaged in most-corrupted packet body

Table 2.2: Column heading explanations

Throughout this chapter, we will present data not only on error events (lost or damaged packets) but also on the radio signal characteristics reported by the WaveLAN unit as a side effect of every packet reception. We hope this will offer insight into the reasons behind certain error events, as opposed to merely recording their frequencies. Table 2.2 explains the column headings we use in most tables. When we present signal level, silence level, and signal quality, we give the minimum observation (marked $\downarrow$), mean (μ), standard deviation

(σ, in parentheses), and maximum observation ($\uparrow$). Unless otherwise specified, all runs use a receive threshold of 3 and a quality threshold of 1. The "Body Bits" and "Worst Body" headings typically refer to the *number* of affected bits, except in Table 2.12, where the frequency of these events merits displaying them in percentage form.

In some trials we received packets from WaveLAN units in nearby rooms or in other buildings. Typically these packets were few, had poor signal characteristics, and were damaged. Frequently we could determine that they were ARP packets or inter-bridge routing packets. We present them, labelled "Outsiders," only when they are significant (due to number or signal characteristics). It is possible for reasons explained above that some packets we identify as outsiders may instead be badly corrupted test packets.

Many independent variables could be investigated for effects on loss and error rates. Examples include: noise emitted by different models of laptop computers, relative position of transmitter and receiver, orientation of antenna, temperature, humidity, position of human bodies, different transmit and receive capabilities of individual units, different materials of walls, ceilings, floors, and furniture, and effects of very distant competing transmitters. Many of these variables have too many values to investigate with any thoroughness, may vary greatly from building to building, or, in the case of interference, may be nearly impossible to measure outside a "clean room" environment. Therefore, our approach for this study was to select certain variables, manipulate them coarsely, and focus on those with the most obvious effect, i.e., attenuation, obstacles, and active radiation sources.

2.4 In-room line-of-sight communication

We discuss WaveLAN behavior in the best case, two stations communicating in the same room. We first present the results of a series of long-running trials designed to estimate the BER of the link under good conditions. We then briefly describe how signal propagation is affected by distance in the absence of obstacles, and finish by examining the effect of the receive threshold on in-room communication.

2.4.1 Base case

In Table 2.3 we present the results of several long trials in an office for a signal level of approximately 29.5.

Two points are worthy of note. First, the bit error rate is very low. These trials represent more than 10^{10} bits, and we experienced very few errors. This is certainly low enough for optimism about extending even fairly error-intolerant applications to a wireless network. Second, some process is causing packets to be lost even in a near perfect environment, though at a rate of well under one per thousand. This could be due to some critical host resource being overloaded, or perhaps could indicate that the modem unit's AGC occasionally reacts too slowly and causes the beginning of a packet to be missed. In a running network, this loss would probably correspond to less than one packet per second, which would plausibly go unnoticed even by a multimedia application.

Trial Name	Packets Received	Packet Loss	Packets Truncated	Bits Received	Wrapper Damaged	Body Bits	Worst Body
office1	102720	.03%	1	8×10^8	0	0	–
office2	40080	0%	0	3×10^8	0	0	–
office3	102720	.01%	0	8×10^8	0	0	–
office4	122159	.02%	0	10^9	0	0	–
office5	488399	.07%	0	4×10^9	0	0	–
office6	122160	.04%	0	10^9	1	1	1
office7	122160	.02%	0	10^9	1	0	–
office8	125040	.02%	0	10^9	0	0	–
office9	122160	.02%	0	10^9	0	0	–

Table 2.3: Results of in-room experiment

2.4.2 Path loss

Next we look at how the signal level changes as a function of distance. This is shown in Figure 2.1. In this experiment the receiver is held fixed adjacent to one wall of a large lecture hall while the transmitter is moved away from it to various distances (the zero point represents the two modem units in physical contact). To the extent that the setup approximates omnidirectional antennas in free space, one would expect to see a smooth dropoff in signal level as distance increases. Indeed, that is the dominant theme. The dips at six and thirty feet are probably due to multipath interference (similar WaveLAN non-monotonicity was observed in [37]), and are likely to be particular to the room where the measurements were taken.

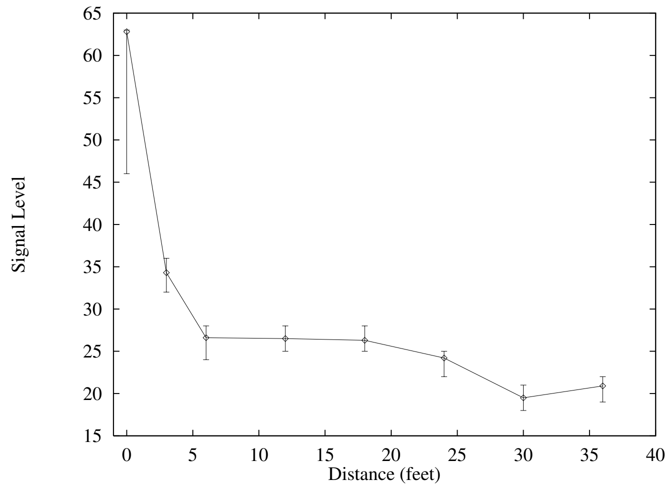

Figure 2.1: Signal level as a function of distance. Error bars represent minimum and maximum observed signal levels.

2.4.3 Summary of in-room operation

In typical cases, the bit error rate and packet loss rate of the network are very low, and there is reason to believe this is true throughout even large rooms.

2.5 Errors due to passive obstacles

In this section, we focus on mundane obstacles which we expect WaveLAN stations to typically encounter. We present the results of three experiments. In the first, we measure the effect created by a single wall. In the second, we hold a receiver in a fixed position and measure the propagation environment to it from four transmitter locations. Finally, we interpose a human body between two WaveLAN units and observe the effects on data transport and signal metrics.

2.5.1 Single wall

In order to measure the effects of walls on WaveLAN signal propagation, we performed two experiments, summarized in Table 2.4. We collected data when a transmitter and receiver were separated by approximately 7 feet ("Air 1"), and then when they were further separated by approximately 6 inches of wall ("Wall 1"). In the case of the second wall, we were forced to interpose not only the wall but an additional four feet of free space, which should have a negligible effect. In each of the four receiver locations we collected 10^8 bits with no loss or error whatsoever.

Trial Name	Packets Received	Level				Silence				Quality			
		$\downarrow$	μ	σ	$\uparrow$	$\downarrow$	μ	σ	$\uparrow$	$\downarrow$	μ	σ	$\uparrow$
Air 1	12720	29	30.58	(0.70)	32	0	1.80	(1.38)	7	15	15.00	(0.00)	15
Wall 1	12720	24	25.78	(0.67)	28	0	1.25	(1.27)	7	14	15.00	(0.03)	15
Air 2	12715	25	28.58	(0.60)	30	0	3.35	(1.11)	13	15	15.00	(0.00)	15
Wall 2	12720	25	26.66	(0.59)	28	0	3.25	(1.10)	8	15	15.00	(0.00)	15

Table 2.4: Signal metrics with a single wall

However, the WaveLAN reports slightly different propagation quality numbers for the different scenarios. We see that the wall affects the signal *level* in a way that is similar to a noticeable move across a room, though the signal *quality* is not significantly reduced. The first wall is plaster with a wire mesh core and it reduces the signal level by about 5 points. The second wall consists of concrete blocks and reduces the signal level by only 2 points, i.e., concrete walls seem to be less hindrance for these signals than plaster over wire mesh walls. Because interposing a wall requires moving one unit, and because small position changes may result in noticeable propagation changes, we cannot be certain how much the wall itself is to blame. However, we have observed similar signal strength decreases due to walls in several other experiments.

2.5.2 Multiple obstacles

In the second experiment we use a more complex setup in the building with concrete block walls ("Wall 2" of the previous experiment). The layout is shown in Figure 2.2, and the results of the test are summarized in Table 2.5. The receiver and the first transmitter location are at diagonally opposite sides of a single office, and we obtain results similar to the previous in-office case. The second transmitter location is approximately four feet away through a single concrete block wall, and corresponds to the single wall experiment of the previous section. The other two transmitter locations are more distant, with several intervening walls and metal objects. They are at distances of roughly 45 and 30 feet respectively from the receiver. The fourth transmitter location shows us our first corrupted packet bodies. Twenty-five of the received packets have a total of 82 bit errors, with the worst packet containing seven bit corruptions. While this number is trivial to correct using error coding, the existing WaveLAN system does not include such a mechanism. The fourth transmitter location demonstrates a single packet truncation of roughly 10% of the packet body.

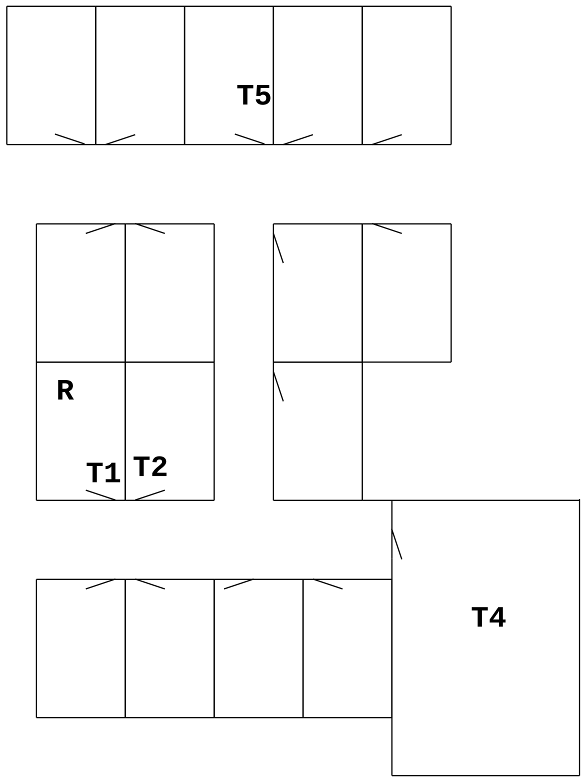

Figure 2.2: Layout of multi-room experiment

It is instructive to make a more detailed comparison of the propagation environments along these four paths (Tables 2.6 and 2.7). First we see that moving through a single wall from Tx1 to Tx2 has not significantly changed the propagation environment. Next we observe that passing through more walls and a door to Tx5 has a more noticeable effect. Finally, if we look in more detail at the Tx5 location, we see that the corrupted packets have

Trial Name	Packets Received	Packet Loss	Packets Truncated	Bits Received	Wrapper Damaged	Body Bits	Worst Body
Tx1	12715	0%	0	10^8	0	0	–
Tx2	12720	.007%	0	10^8	0	0	–
Tx4	1440	.07%	0	10^7	0	0	–
Tx5	1440	.07%	1	10^7	0	82	7

Table 2.5: Results of multi-room experiments

noticeably reduced signal *level*, while the sole truncated packet has a noticeably reduced signal *quality*. We will see similar correlations in other scenarios, so they may point to some feature of the WaveLAN hardware. For example, it is possible that data decoding and clock recovery are impaired by different signal features.

From this brief investigation of obstacles, we learn that different construction materials have noticeably different effects on WaveLAN propagation.

Trial	Packets	Level				Silence				Quality			
Name	Received	↓	μ	σ	↑	↓	μ	σ	↑	↓	μ	σ	↑
Tx1	12715	25	28.58	(0.60)	30	0	3.35	(1.11)	13	15	15.00	(0.00)	15
Tx2	12720	25	26.66	(0.59)	28	0	3.25	(1.10)	8	15	15.00	(0.00)	15
Tx4	1440	11	13.81	(0.66)	15	0	4.03	(1.27)	9	15	15.00	(0.00)	15
Tx5	1440	7	9.50	(0.93)	12	0	4.08	(1.29)	8	11	14.99	(0.15)	15

Table 2.6: Signal metrics for multi-room experiment

Packet	Packets	Level				Silence				Quality			
Type	Received	↓	μ	σ	↑	↓	μ	σ	↑	↓	μ	σ	↑
All	1440	7	9.50	(0.93)	12	0	4.08	(1.29)	8	11	14.99	(0.15)	15
Error-Free	1414	7	9.51	(0.92)	12	0	4.08	(1.28)	8	14	15.00	(0.03)	15
Truncated	1	10	10.00	(-.–)	10	3	3.00	(-.–)	3	12	12.00	(-.–)	12
Body Damaged	25	7	8.72	(0.87)	10	1	4.56	(1.27)	8	11	14.72	(0.87)	15

Table 2.7: Signal metrics for multi-room scenario Tx5

2.5.3 Human Body

After noticing in a few informal trials that WaveLAN signals seemed significantly attenuated by a human body, we decided to investigate further. In order to obtain a path with significant attenuation, we separated two WaveLAN units by placing them in two rooms across a hallway. The direct path was approximately 56 feet long and passed through two concrete block walls and some classroom furniture. We collected two packet streams, with the second impaired by the presence of a person bending over as if to examine the laptop screen closely. Tables 2.8 and 2.9 contain the results. Interposing a person has induced packet loss, truncation, and packet body damage. Furthermore, we observe a noticeable reduction in signal level. These results could be significant if similar networks were deployed in crowded lecture halls, though the effect would probably be mitigated if a WaveLAN base station were located near the ceiling.

Trial Name	Packets Received	Packet Loss	Packets Truncated	Bits Received	Wrapper Damaged	Body Bits	Worst Body
No body	1440	0%	0	10^7	0	0	-
Body	1442	.14%	3	10^7	1	224	27

Table 2.8: Effects of human body on packet loss and errors

Trial Name	Packet Type	Packets Received	Level				Silence				Quality			
			↓	μ	σ	↑	↓	μ	σ	↑	↓	μ	σ	↑
No body	All Packets	1440	11	12.55	(0.60)	15	0	4.23	(1.47)	13	15	15.00	(0.00)	15
Body	All Packets	1442	5	6.73	(0.88)	10	0	1.90	(1.64)	9	11	14.95	(0.27)	15
	Undamaged	1214	5	6.73	(0.87)	10	0	1.73	(1.59)	9	14	14.99	(0.09)	15
	Truncated	3	7	7.67	(0.47)	8	0	3.00	(2.16)	5	11	12.33	(1.25)	14
	Wrapper damaged	1	5	5.00	(0.00)	5	1	1.00	(0.00)	1	15	15.00	(0.00)	15
	Body damaged	224	5	6.74	(0.95)	9	0	2.85	(1.61)	7	13	14.77	(0.51)	15

Table 2.9: Effect of human body on signal measurements

2.5.4 Signal Level versus Errors

Because the WaveLAN design addresses threats such as multipath effects and narrowband interference, signal level is an important predictor of error rate, and an interesting question is when the signal level is too low to receive packets reliably. Table 2.10 presents the aggregated results of several trials, with slight variations of receiver position, orientation, and obstacles within each trial. While undamaged packets may have a signal level as low as 5, and damaged packets one as high as 12, the main body of damaged packets has signal levels below 8, whereas it is well above 8 for undamaged packets. In this aggregation, we observed several "outsider" packets, which we believe to be from other buildings; the most striking difference between the damaged (truncated and/or corrupted) and undamaged outsider packets is their signal quality.

2.6 Errors due to competing active radiation sources

We briefly discuss four different types of interference: front end overload, narrowband interference, spread spectrum interference, and competing WaveLAN units.

Packet Type	Packets Received	Level ↓	Level μ	Level σ	Level ↑	Silence ↓	Silence μ	Silence σ	Silence ↑	Quality ↓	Quality μ	Quality σ	Quality ↑
All test packets	8634	4	14.15	(6.32)	27	0	2.89	(1.64)	9	2	14.80	(0.86)	15
Undamaged	7942	5	14.74	(6.23)	27	0	2.83	(1.64)	9	3	14.94	(0.37)	15
Truncated	107	4	6.20	(1.44)	10	0	3.48	(1.34)	6	6	10.07	(1.74)	15
Wrapper damaged	9	6	7.56	(0.83)	9	0	2.89	(1.79)	6	10	12.89	(2.18)	15
Body damaged	576	4	7.52	(1.45)	12	0	3.61	(1.46)	9	2	13.80	(1.75)	15
Undamaged outsiders	73	5	6.23	(0.77)	8	0	2.63	(1.12)	5	9	14.49	(0.99)	15
Damaged outsiders	867	2	5.19	(1.87)	18	0	3.26	(1.37)	17	1	7.49	(2.06)	15

Table 2.10: Packet error conditions versus signal metrics

2.6.1 Front end overload

If a very powerful transmitter is close to a receiver, the early filter stages of the receiver, which are designed to reject out-of-band signals, may be overwhelmed. We tested two sources of front end overload. The first was a 144 MHz Amateur Radio Service FM transmitter emitting roughly two watts while in physical contact with the receiver's modem unit. During this test we observed no bit errors unless we separated the two WaveLAN units far enough for the signal level to be severely attenuated, which is itself a source of errors.

Microwave ovens are powerful sources of potential interference which are common in an office environment. Though we expect them to be well shielded, even a small leakage percentage would be significant. We made a crude test of a single microwave oven. The transmitter was placed at varying distances from the receiver, which was in physical contact with a microwave oven (operating with the door closed), and no errors were observed. Since most microwave ovens operate in the 2 GHz range, it is possible that 2.4 GHz WaveLAN units would receive more interference.

2.6.2 Narrowband interference

We briefly investigated the effects of two narrowband 900 MHz cordless phones (AT&T 9100, Panasonic KX-T9500) on WaveLAN reception. We placed our WaveLAN transmitter and receiver approximately 20 feet apart in a large lecture hall and subjected them to various telephone interference. The results of this investigation are summarized in Table 2.11. In the "phones off" trial, the handsets and base units were turned off; in the "cluster" trial, both handsets and base units were activated and placed a few inches from the receiver's modem unit. These represent the extreme cases. For the other three trials, we moved either the handsets or the base units to an office across the hall from the lecture hall to investigate whether the phones were using some form of power control.[1] In the "handsets nearby talking" case, a human body was necessarily near the receiver, but in all other cases nobody was in the immediate vicinity of the WaveLAN units.

[1] Power control is the practice of reducing transmitter power as long as it does not induce errors; this saves battery life and allows frequency re-use.

Trial Name	Packet Source	Packets Received	Level ↓	Level μ	Level σ	Level ↑	Silence ↓	Silence μ	Silence σ	Silence ↑	Quality ↓	Quality μ	Quality σ	Quality ↑
Phones off	Test	1432	25	26.71	(0.66)	28	0	2.40	(1.32)	6	14	14.98	(0.14)	15
	Outsiders	330	2	4.69	(1.56)	25	0	2.57	(1.37)	6	1	6.67	(1.99)	15
Cluster	Test	1440	25	26.89	(0.61)	28	8	15.45	(1.22)	19	15	15.00	(0.00)	15
Handsets nearby	Test	1440	25	26.62	(0.62)	28	3	11.33	(2.62)	15	13	14.93	(0.26)	15
Handsets nearby talking	Test	1431	25	26.63	(0.59)	28	0	6.11	(1.98)	10	14	14.96	(0.20)	15
	Outsiders	213	6	7.85	(0.87)	11	2	6.26	(1.73)	11	1	9.93	(1.93)	15
Bases nearby	Test	1440	25	26.31	(0.65)	28	17	19.32	(1.18)	21	15	15.00	(0.00)	15

Table 2.11: The effects of narrowband 900 MHz cordless phones

Except for the “cluster” trial, both handsets received nearly solid static, and the Panasonic unit beeped, presumably to indicate poor signal conditions. In sharp contrast, WaveLAN experienced no damaged test packets, and only background levels of packet loss (the “phones off” and “handsets nearby talking” trials each experienced a single packet loss). The telephones affected the silence level to varying degrees; in the two cases where the silence level was lowest, we observed packets from stations not participating in the test (they were relatively weak and all were damaged in some way, so we suspect they were leakage from one or more nearby buildings which we know contain WaveLAN units). The fact that the highest silence level is in the “bases nearby” trial, when the handsets were distant, rather than the “cluster” trial, suggests that the cordless phones may be using power control.

We also briefly investigated the effects of an Advanced Mobile Phone Service (AMPS) narrowband FM cellular phone on WaveLAN reception. Once again, at varying distances, the WaveLAN seemed immune to bit errors. On the other hand, the cellular phone received significant amounts of white noise, totally overwhelming the audio signal, when it was close to an operating WaveLAN transmitter. We did not control for the effects of power control or channel selection by the cellular phone system.

WaveLAN's resistance to these interference sources is probably due to the DSSS modulation, which is known to be resistant to narrowband sources [99].

2.6.3 900 MHz spread spectrum cordless phone

We also investigated the effects of two 900 MHz spread spectrum cordless phones, an AT&T 9300 and a Radio Shack ET-909.[2] The two phones were quite similar in user interface and effects on WaveLAN reception. Unfortunately, if the handsets were within approximately three feet of each other, they could not simultaneously maintain connections to their respective base stations, even when multiple channels were employed. Our WaveLAN transmitter and receiver were approximately 25 feet apart in a conference room. The “near” location used for these trials was several inches from the receiver's modem unit, and the “far” location was approximately 14 feet from the receiver and 20 feet from the transmitter. For all runs we collected enough received packets to yield roughly 10^7 bits of packet body. Throughout

[2]The Radio Shack ET-909 reportedly uses DSSS modulation, and we suspect the AT&T 9300 does as well.

the tests the telephones maintained acceptable audio signals. There were occasional clicks, but no actual outages, and the handsets did not complain about poor signal conditions.

The results of this investigation are summarized in Table 2.12. Three cases indicate that these phones can severely damage the WaveLAN environment: half of the packets are totally lost, while every packet that arrives is truncated. On the other hand, the "RS remote cluster" case indicates that reasonable separation between the WaveLAN and telephone leaves the link unharmed (though we will see below that the signal characteristics change noticeably). Finally, the "AT&T handset" case demonstrates that there is a significant intermediate effect: while a small number of packets are lost or truncated, nearly two thirds of the remainder contain correctable errors (the worst corruption of a packet body observed was 5% of the bits).

Trial Name	Phone Type	Handset Location	Base Location	Packet Loss	Packets Truncated	Wrapper Damaged	Body Bits	Worst Body
Phones off	-	-	-	.5%	0%	0%	0%	-
RS base	RS	far	near	52%	100%	0%	0%	-
RS cluster	RS	near	near	51%	100%	0%	0%	-
AT&T cluster	AT&T	near	near	52%	100%	0%	0%	-
RS remote cluster	RS	far	far	0%	0%	0%	0%	-
AT&T handset	AT&T	near	far	1%	4%	1%	59%	4.9%

Table 2.12: Summary of spread spectrum cordless phones

The signal information presented in Tables 2.13 and 2.14 suggests the following observations:

- The phones add a significant amount of noise to the environment. Every case except for "phones off" has a very high silence level.
- Very low signal quality seems to be a good predictor of truncation.
- If the signal level is high but signal quality is not outstanding, bit errors are likely.
- Based on signal information, in some trials some "outsiders" may be test packets corrupted beyond recognition.

In summary, ISM band spread spectrum cordless phones, depending on their location, can be harmless, severely disrupt a WaveLAN, or inject mild, plausibly correctable errors.

2.6.4 Competing WaveLAN units

The single-channel design of the WaveLAN suggests that non-cooperating WaveLAN units could provide significant mutual interference. Using the experimental layout described in Section 2.5.2, we placed additional WaveLAN transmitters at the Tx4 and Tx5 locations, and raised their receive threshold to 35, thus ensuring they would transmit continuously,

Trial	Packet	Packets	Level				Silence				Quality			
Name	Source	Received	↓	μ	σ	↑	↓	μ	σ	↑	↓	μ	σ	↑
Phones off	Test	1389	28	29.63	(0.60)	31	0	2.20	(1.10)	7	15	15.00	(0.00)	15
	Outsiders	619	2	7.35	(5.54)	29	0	2.51	(1.17)	7	1	9.84	(4.41)	15
RS base	Test	1597	28	31.54	(2.57)	38	1	32.73	(5.44)	38	2	10.07	(3.29)	15
	Outsiders	316	4	31.03	(5.77)	37	2	27.07	(8.06)	37	1	7.10	(4.07)	15
RS cluster	Test	1488	28	32.01	(1.73)	37	0	30.73	(6.58)	37	1	8.91	(3.09)	15
	Outsiders	1818	3	32.57	(2.71)	37	0	28.97	(6.84)	37	1	5.48	(2.98)	15
AT&T cluster	Test	1766	29	32.52	(4.18)	41	1	38.96	(3.16)	41	1	7.45	(5.42)	15
	Outsiders	157	4	28.69	(8.95)	40	1	37.77	(7.13)	41	1	8.30	(5.43)	15
RS remote cluster	Test	1440	28	29.83	(0.65)	32	0	21.81	(5.91)	27	15	15.00	(0.00)	15
	Outsiders	9	3	4.22	(0.63)	5	21	24.33	(1.41)	26	7	8.11	(1.10)	11
AT&T handset	Test	1456	28	30.04	(1.04)	33	0	23.52	(6.57)	32	1	13.46	(2.14)	15
	Outsiders	265	4	31.31	(6.23)	33	3	23.29	(6.33)	32	1	6.15	(1.57)	12

Table 2.13: Signal measurements for spread spectrum phones

Packet	Packets	Level				Silence				Quality			
Type	Received	↓	μ	σ	↑	↓	μ	σ	↑	↓	μ	σ	↑
All test	9136	28	31.01	(2.57)	41	0	25.78	(12.67)	41	1	11.44	(4.35)	15
Undamaged	3341	28	29.81	(0.72)	32	0	13.96	(10.97)	30	1	14.81	(0.77)	15
Truncated	4911	28	32.02	(3.09)	41	0	34.27	(6.35)	41	1	8.76	(4.28)	15
Wrapper damaged	21	29	30.62	(0.65)	32	3	24.71	(5.79)	29	9	12.24	(1.63)	15
Body damaged	863	28	29.89	(1.00)	33	0	23.26	(6.79)	30	3	13.62	(1.98)	15

Table 2.14: Signal breakdown for spread spectrum phone test packets

without deferring to any nearby stations, including each other. We then attempted to transmit from location Tx1 to the receiver. Using the standard receive threshold value of 3, the link was completely unusable. For example, hundreds of unknown Ethernet addresses appear in the receiver's log, often attached to packet bodies transmitted by our test programs, indicating that the Ethernet station address field was frequently corrupted. Many transmission attempts by Tx1 were aborted due to repeated "medium busy" conditions. However, some Tx1 packets arrived at the receiver with very little corruption.

Raising the receive thresholds of Tx1 and the receiver to 25, a value safely above the signal levels we had measured from the hostile transmitters' locations (Table 2.6), allowed the communicating stations to completely mask out the competition. We collected 10^8 bits with and without competition. In the case with competition, we experienced a .02% packet loss rate, which is not clearly significant, and no bit errors. If we compare the signal statistics (Table 2.15), we see that the background ("silence") level has increased significantly, but that the signal level and quality are essentially unchanged.

The observations are consistent with the following explanation. Raising the Tx1's receive threshold allowed it to transmit without deferring to the distant interfering stations. At this point, various factors enabled the receiving station to capture almost every packet without errors: the near-far effect, its elevated receive threshold, and WaveLAN's multipath-resistant signal processing.

These results suggest that transmitter power control is useful for wireless LANs, though it would probably need to be coupled with link-level retransmission (discussed in the next

Trial Name	Packet Source	Packets Received	Level				Silence				Quality			
			↓	μ	σ	↑	↓	μ	σ	↑	↓	μ	σ	↑
Without interference	Test	12715	25	28.58	(0.60)	30	0	3.35	(1.11)	13	15	15.00	(0.00)	15
With interference	Test	12717	28	28.65	(0.49)	30	0	13.62	(3.39)	21	15	15.00	(0.00)	15
	Outsiders	31	24	28.65	(0.51)	32	0	13.62	(3.39)	28	15	15.00	(0.00)	15

Table 2.15: Signal metrics with and without interfering WaveLAN transmitters

chapter) to mask occasional packet losses that would result. We have observed, though not experimentally quantified, that, when operated with high receiver thresholds, WaveLAN is fairly resistant to errors caused by hidden transmitters.

2.7 Related work

Duchamp and Reynolds have observed signal quality, throughput and error characteristics of an earlier model of 900MHz WaveLAN installed in ISA bus PCs, subjected to various challenges such as distance and multipath propagation [37]. Their testing regime included a propagation environment impeded by distance and local scatter induced by reflections from a wall. In this environment they observed packet loss and corruption rates both typically below 1%, except when a combination of attenuation and local scatter produced packet loss rates in the vicinity of 10% with a peak around 15% and packet corruption rates ranging as high as 40%. In the difficult environment, both rates varied non-monotonically with distance, making it very unstable and unpredictable in the face of small motions. We observed similar loss and error rates due to attenuation and/or obstacles. Our work extends theirs by considering more error sources and by our investigation of the feasibility of pseudo-cellular WaveLAN operation.

Lewis and Guy measured the performance of the Arlan 610 system [73] and evaluated its utility for mobile multimedia applications. An interesting study [13] suggests it may be possible to obtain as much as 14 Mb/s from a single European DECT-style phone channel, albeit with a BER of 10^{-5}.

Rathke, Schlaeger, Wolisz [114] observed WaveLAN and Arlan in an office environment. They took TCP throughput measurements at every intersection of a 2-meter grid. For each LAN technology they observed throughput that was very poorly related to distance, including numerous "dead spots" which were not easily predictable. They observed WaveLAN packet loss rates of up to 40%.

2.8 Conclusions

We used detailed packet collection to investigate the effects of distance, obstacles, and different interference sources on the error and loss rates of WaveLAN a 2 Mb/s wireless LAN designed specifically for an indoor fading environment. The DSSS modulation and multipath resistant receiver design seemed to confer resistance to many environmental hazards.

Distance alone seemed to have little effect over a fairly large area and WaveLAN can frequently penetrate formidable obstacles, although it eventually succumbs to attenuation. In general we observed very few errors. The worst errors were induced by spread spectrum cordless phones operating in the same frequency band, and even these were substantially reduced by distance.

In many cases, we observed a near-perfect link, arguing that error coding would be useless overhead in most situations. However, there were other situations, some plausibly predictable by signal measurements, in which there is frequent packet corruption. Our observations, especially the spread spectrum phone results in Section 2.6.3, argue that the errors we did observe might be recoverable through a variable forward error correction mechanism, and this will be addressed in the next chapter.

In general, it appeared that environments which seriously threaten communication are common. Some of the more serious threats, including human beings near radio transceivers and spectrum crowding, are likely to affect many wireless LAN technologies. For example, the IEEE 802.11b wireless LAN inhabits the same frequency band used by European DECT cordless phones. The interference posed by competing WaveLAN transmitters IEEE 802.11b systems, which do not currently support power control, might observe similar outcomes. Finally, researchers seeking to characterize "real-world" error environments must be prepared to vary many parameters, including building construction style.

Chapter 3

Trace-based Evaluation of Adaptive Error Control

Now that the initial investigation presented in Chapter 2 has determined which environmental threats are most serious to reliable and predictable wireless LAN communication, it is time to investigate the major threats in more detail in order to design and verify effective responses.

The main issues are

- How severe are commonly-encountered errors? What measures are effective against them?
- Since fixed error control policies are simpler and cheaper to implement than dynamic error control, do fixed policies perform acceptably?
- Even if it is theoretically possible for a dynamic error control policy with perfect knowledge of the future to significantly outperform a static policy, this is not useful for designing real systems. Effective real-world adaptation will require that recent environmental conditions effectively predict conditions in the near future and that collecting historical data and generating predictions are not computationally expensive.
- Finally, an error control policy that increases performance significantly across many dynamic error environments is probably more useful than policies that perform very well in particular dynamic environments and very poorly in others.

The core of this chapter will be a performance comparison of various fixed and dynamic error control policies in multiple error environments.

Evaluating adaptive error control in a wireless environment is challenging because two factors make repeatable experiments difficult. First, the wireless environment cannot easily be isolated, so the dynamic error environment is hard to control. Second, the adaptation process changes the behavior of the system, e.g., the packet stream sent with adaptation will be different from the stream sent without adaptation, making a direct comparison of different adaptation policies difficult. For these reasons we evaluate adaptive error control using a trace-based evaluation methodology. The first step of this process is characterizing the error environment by collecting a set of detailed error traces. These traces then form the input to a simulator that evaluates adaptation policies based on forward error correction and packet shrinking. The simulator uses a performance metric that accounts for the traffic stream changes introduced by the adaptation policies.

Section 3.1 presents the error traces used later to evaluate error control techniques and policies. Section 3.2 analyzes the traces in terms of the frequency of bit corruption and the amount of error correction overhead needed to reduce it to meaningful levels, the frequency and degree of packet truncation and the extent to which packet length adjustment overcomes truncation, and the frequency of packet loss. Section 3.3 describes the fixed and adaptive error control policies we will compare using the simulator described in Section 3.4. Section 3.5 evaluates how each policy performs in relation to a perfectly-informed "oracle" and to each other. Section 3.7 concludes with a summary of how necessary, how effective, and how practical link-level adaptive error correction appears to be.

The evaluation in this chapter is somewhat independent of architectural considerations such as whether error control should be per-hop versus end-to-end or whether it should be a link-layer, network-layer, or transport-layer function. We will address these issues in the next chapter.

3.1 Trace-based Error Environment

We investigate the nature of the WaveLAN error environment by collecting detailed error traces for a set of interference scenarios. The traces we present span a wide range of packet loss, truncation, and bit corruption rates and will be the basis for our later evaluation of adaptive error control.

3.1.1 Trace Collection and Format

To characterize error environments, we monitored data transfers between two identical DECpc 425SL laptops (25 MHz 80486) running NetBSD 1.1. For different tests, we placed the PCs in different environments or added competing radiation sources. Our laptops use PCMCIA WaveLAN [127] interfaces operating in the 902-928 MHz frequency band. Section 1.6 provides more details on the WaveLAN interface.

We instrumented our kernel device driver to collect all packets (even runt packets and those that fail Ethernet CRC) into an 8-megabyte kernel trace buffer. This allows collection of approximately 30 seconds of continuous network activity. Packets are stored along with a sub-millisecond timestamp and signal information from the RF modem: signal level during reception (related to the receiver's auto-gain-control level), "silence level" (signal level just after packet reception), signal quality, and selected receive antenna. The designated transmitter sends specially-formatted UDP datagrams that contain several repetitions of a single Walsh code word. We selected Hadamard-Walsh codes for the packet contents because they allow us to evaluate the sensitivity of the errors to packet contents (see Section 3.4.1). When the kernel trace buffer is full, the contents are compressed and stored on disk for off-line analysis, and the transmitter is instructed to send another stream of packets.

The first phase of the analysis consists of processing the binary trace file into a text file containing a content-insensitive summary of errors. Figure 3.1 shows an example. In general, each processed trace event contains a timestamp, the transmitted and received packet lengths (expressed as a count of 32-bit words), signal information from the RF modem, a packet sequence number, and a list of corrupted (inverted) bits. In this example, packet 151 was missing 3 out of 352 32-bit words, but was otherwise undamaged; packets 152 and 153 were lost; packet 154 was severely truncated and nine bits of the remainder were corrupted; after packet 154 a 7-word packet of unrecognizable contents was received. The second phase of the analysis extracts various summary information from the traces, which is described in the remainder of the section.

```
@2738775 {352 349} [40 2 4 1] 151:
lost 152 153
@2757194 {352 30} [34 40 10 1] 154: 156 160 163 320 324 327 411 415 418
@2780328 {-1 7} [34 40 6 0] alien
```

Figure 3.1: Example of human-readable trace file

3.1.2 Interference Scenarios

Among the error sources we could have investigated are attenuation, interfering radio sources, multipath interference (which can vary due to details of antenna placement and orientation), motion, and data dependent effects. We chose to focus on the effects of the most significant error sources, as determined by our investigation in Chapter 2, namely active interference sources and attenuation. Note that the encoding used by WaveLAN already addresses many problems such as multipath interference (Section 1.6).

To explore the range of error severity due to interference, we investigated 30-second traces in four different situations. In the "office" scenario, the communicating machines were separated by a distance of approximately 5 feet, and there were no known interference sources. In the "walking" scenario, the communicating machines were separated by roughly 3 feet. A cordless telephone base station was a few inches from the receiver's modem unit (as might be the case on an office desk), and the telephone handset moved repeatedly at walking speed back and forth from roughly a foot away from the base station to a point approximately 30 feet away (where the handset complained about signal loss). The cordless phone used was a Radio Shack ET-909, which uses spread spectrum modulation in the same band as our WaveLAN units. We observed in the previous chapter that it is a particularly harsh interference source. The "adjacent" case was the same as the "walking" case except that the handset was fixed adjacent to the receiver's modem unit and was power-cycled at a rate of approximately twice per second (in an attempt to provide an error environment with more challenging variability than the "walking" case). In the "table" case, two communicating units were approximately three feet apart, with the telephone handset and base station about an inch and about five inches from the receiver's modem unit. In these traces we observe packet loss rates of up to 30% and packet truncation rates of up to 23%. We notice that the bit error rate varies by more than an order of magnitude, and that this variation is not straightforwardly linked to the other error rates.

3.1.3 Attenuation

While attenuation can be reduced by provisioning a network more densely with base stations, mobile users may still encounter occasional coverage "holes." Therefore, we investigated the effects of attenuation to see how they differed from those of active interference sources. The fifth, "walls," trace represents a path with significant attenuation: we separated two units by placing them in two rooms across a hallway. The direct path was approximately 17 feet long and passed through two thick concrete block walls, a metal display case, and some classroom furniture. As compared to the interference environments, attenuation has an almost insignificant truncation rate, but has significant packet corruption.

3.1.4 Summary

These five examples illustrate the wide range of damage induced by interference and attenuation. Table 3.1 details the three types of errors commonly encountered on wireless links: lost packets, truncation, and bit corruption. Notice that the packet loss rate and bit error rate both vary by three orders of magnitude while the the truncation rate varies by two orders of magnitude, and that there is only weak correlation of those metrics (bit error rate and packet corruption rate are only loosely related, as well). While the "office" trace is essentially error-free, most end-to-end transport protocols would be seriously challenged by the "walking" trace and essentially unable to operate in environments represented by the other three traces (column headings are cumulative, so "packet corruption" is the percentage of non-lost, non-truncated packets which were corrupted).

An important question is how the error environment changes over time. Figure 3.2 displays a packet-by-packet accounting of the dynamic behavior of four of the traces (the essentially-error-free "office" trace is omitted). Again the traces differ noticeably. While "walking" displays scattered packet corruption, "adjacent" is characterized by on/off pulses of loss and truncation, "table" is nearly-continuous packet corruption with occasional loss and truncation, while "walls" is a more-complicated mixture of loss, truncation, and corruption. In Chapter 2 we saw that the main error sources are attenuation caused by obstacles (walls and humans) and interference caused by some active radio sources that use the same frequency band as the WaveLAN card. For these types of error sources we expect that changes will often be caused by human actions such as switching the competing source on or off, or moving the mobile host or competing source. Indeed, as the "adjacent" interference source is turned on and off, two steady but alternating error environments are visible. Even though this is fairly rapid human action, long periods of similar error behavior occur.

Trial Name	Packets Sent	Bits Received	Packet Loss Rate	Packet Truncation Rate	Packet Corruption Rate	Bit Error Rate
Office	5729	6×10^7	0	0	3.4×10^{-4}	0
Walking	5729	6×10^7	7×10^{-4}	9×10^{-4}	5.4×10^{-2}	1.6×10^{-4}
Adjacent	10487	6×10^7	3.1×10^{-1}	2.3×10^{-1}	3.5×10^{-3}	1×10^{-3}
Table	5916	6×10^7	5.4×10^{-2}	1.9×10^{-2}	9.4×10^{-1}	4.9×10^{-3}
Walls	5776	6×10^7	6×10^{-3}	1.7×10^{-2}	2.7×10^{-1}	3.8×10^{-4}

Table 3.1: Summary of various error scenarios. Since the trace buffer is of a fixed size, lost packets cause it to fill more slowly, and it captures a wider range of sent-packet sequence numbers. "Packet loss rate" is the fraction of packets that were sent but not received; "packet truncation rate" is the fraction of packets that were received but truncated, and "packet corruption rate" is the fraction of packets that were received without truncation but with at least one corrupted bit in the UDP data area. The Bit Error Rate is computed for all bits in the UDP data area which arrive at the receiver (that is, bits lost to truncation do not contribute to the BER computation).

Note that there there will also be changes in the error environment at a finer time scale. An example would be interference from another wireless network: the level of interference could change from bit to bit. This type of interference is typically addressed by methods different from what we propose in this dissertation. For example, WaveLAN uses direct sequence modulation to combat a variety of error sources (fading and narrowband interference signals); wireless networks could also employ code diversity, frequency diversity, and base station antenna sectoring to further reduce interference. Thus, in a well-designed wireless LAN, it is plausible that most errors will be due to attenuation and interference varying on a multi-packet time scale, as described above.

3.2 Errors and Error Control Techniques

In this section we summarize the error traces in terms of three error types, bit corruption, packet truncation, and packet loss, and discuss error control options for each error type.

3.2.1 Bit Corruption

A first source of errors is bit corruption. WaveLAN typically operates as an Ethernet in that all packets are protected by a 32-bit Ethernet CRC, and all packets failing that CRC check are rejected. In Tables 3.2 and 3.3 we present a brief analysis of bit corruptions occurring in eight cordless phone interference experiments, including "Office," "Walking," "Adjacent," "Table," and four similar situations. The histogram in Table 3.2 indicates that approximately 28% of the 269,037 transmitted packets would have been discarded as failing CRC, but that essentially all of them have 5% or fewer flipped bits, well within the reach of FEC. Furthermore, interference seems to corrupt each bit *value* with essentially identical probability (Table 3.3). We did not observe any tendency for bad bits to be at any particular

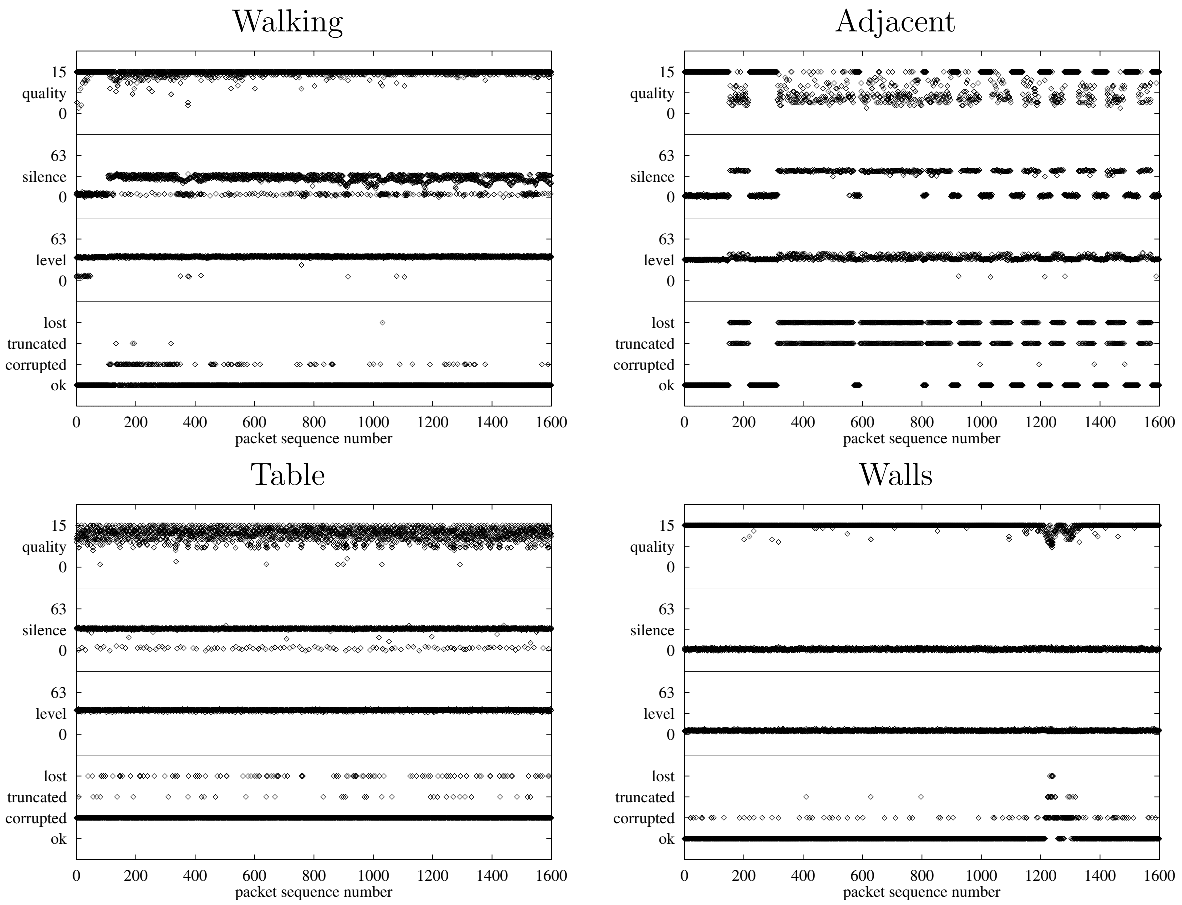

Figure 3.2: Temporal displays of four traces. Each point represents a single event, such as a loss or corruption. Only the first 1,600 events of each trace are displayed.

bit *position* within a packet. However, bit corruptions very often follow a specific pattern: three bits, separated by 3 and 2 bit positions, are flipped. This means that most error bursts occur within one or two bytes. The particular pattern is plausibly an artifact of the 8-bit-deep linear feedback scrambler used by WaveLAN.

3.2.2 Applicability of FEC to Correcting Bit Corruption

The bit corruption patterns discussed above suggest that FEC can be used to reduce the error rate. We now evaluate the effectiveness of Reed-Solomon burst-correcting block codes at correcting the errors observed in our traces and briefly discuss the corresponding implementation costs. Appendix A contains FEC background information and a discussion of why we chose to employ these particular codes.

Percentage of bits flipped	Number of Packets
0	192846
1	74000
2	982
3	509
4	364
5	208
6	72
7	21
8	12
9	7
10	10
11	4
12	1
14	1

Table 3.2: Histogram of per-packet bit error rate

Original bit	Times flipped	Times sent
0	998945	1509063744
1	997311	1509568640

Table 3.3: Bit flipping behavior observed during the combined interference runs. The overall bit error rate was 6.6×10^{-4}.

Effectiveness

Reed-Solomon codes with 8-bit symbols have a block size of 255 bytes, which may be arbitrarily divided among user data bytes and overhead bytes. The decoder can correct as many corrupted *bytes* as half the number of overhead bytes, and doesn't care how many bits in a given byte are corrupted. This means that these codes are particularly suited to error bursts of fewer than 8 bits. The 255-byte block size poses some inconvenience, as packet body lengths are typically even and often slightly longer than a power of two. However, we can break a packet into some number of 255-byte blocks and one "shortened" block, which we zero-pad for encoding and decoding purposes.

We ran the traces presented in the previous section through a software implementation of Reed-Solomon encoding, and evaluated how many corrupted packets could be recovered. Figure 3.3 shows how effective FEC is at reducing data loss due to bit corruption. The graph shows how the percentage of uncorrectable packet bodies decreases as the FEC overhead is increased. We processed the traces described above by removing all records that reflect losses, unrecognized packets, and truncated packets (for which full corruption patterns are necessarily unavailable).

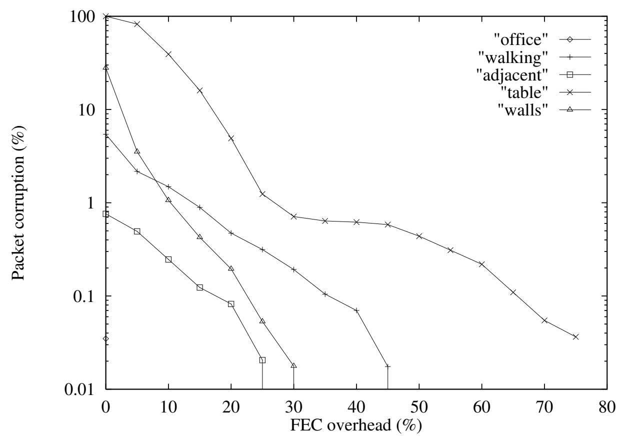

Figure 3.3: Residual packet error rate versus error correction overhead. The residual packet error rate is the percentage of packets that arrived without being lost or truncated and had one or more bit errors not corrected by the Reed-Solomon code. The leftmost (0%) point on each curve represents the uncorrected packet corruption rate given in Table 3.1. The "office" dataset has no uncorrected errors once FEC is enabled, and hence is represented on the graph by the single point (0,.03%).

It is difficult to say exactly what residual packet error rate is appropriate. If truncation and loss (discussed below) were not threats and we were not using link-level retransmission, we might select a threshold error coding failure rate of 1%, since that would cause little or no disruption to higher-layer protocols such as TCP. It is worthy of note that the FEC overhead level required to reach the neighborhood of 1% varies from 0% in the "office" environment to 20% or 30% in the "table" environment. If retransmission is used, there is an efficiency argument in favor of moderate residual error rates. Using the "table" environment as an example, it may be better to suffer a 10% packet error rate rather than pay a 15% error coding overhead. On the other hand, "optimizing" the packet error rate will increase latency.

These error coding efficiency results are conservative as we have not investigated interleaving, which would tend to average out the number of corrupted symbols per block, quite possibly improving performance. Even so, they show that FEC is effective at recovering from bit corruption. The graphs also show that it would be costly to always employ error coding strong enough for the worst case: substantial bandwidth would be wasted in the other cases. Selecting an error coding appropriate for the average error rate is also suboptimal: it will be too aggressive in some cases and not aggressive enough in others. Given the high variance in error rates, the average is not a useful metric.

Implementation Costs

The costs associated with the implementation of FEC are additional bandwidth on the link, as discussed above, and the FEC coder/decoder. FEC codecs can be implemented in either hardware or software. For example, multi-rate single-chip convolutional codecs that operate at tens of megabits per second are available as commodity parts [107], making hardware implementations practical and economical. For this work we relied on a publicly available software implementation of Reed-Solomon codes [66], which can encode at up to 2.7 Mb/s and decode at up to 1.5 Mb/s on a 100 MHz Pentium.

3.2.3 Packet Truncation

Truncated packets are a serious problem for network reliability. While erasure-correcting FEC can be designed to recover from truncation, this requires overhead beyond that necessary to correct occasional symbol errors. We have observed that WaveLAN experiences significant truncation in heavy interference situations. While it seems intuitive that short packets are less likely to be truncated than long packets, the truncation rate might depend on the packet length in a number of ways. For example, the probability of truncation could be proportional to the packet length (some interfering source occasionally truncates whichever packet is being transmitted, independent of its length), or packets could be truncated to some fixed length (by some process within the receiver). Identifying the precise error behavior is important if we want to reduce the truncation rate.

In order to better understand the relationship between packet length and truncation errors, we collected packet traces that use a mix of packet sizes. Since we do not understand the cause of the truncation, it is important that all packet sizes get sufficient exposure to the potential cause of truncation. For this reason, we arranged the packet sequences so that roughly equal *time* was spent transmitting packets of each observed length, i.e., more of the shorter packets were sent. One interesting detail is that the WaveLAN link header and trailer occupy air time equivalent to 38 bytes [6] and Ethernet, IP, and UDP headers and trailers occupy 48 bytes. This 86-byte overhead is significant when we send test packets of only 128 user bytes. We correct for this by sending packets according to a schedule which roughly balances their air time.

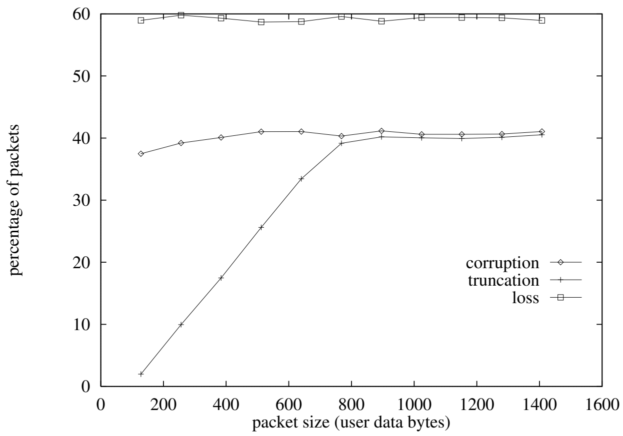

Figure 3.4: Packet injury as a function of packet length. The cordless phone is on throughout the trace, and both base and handset positions are fixed.

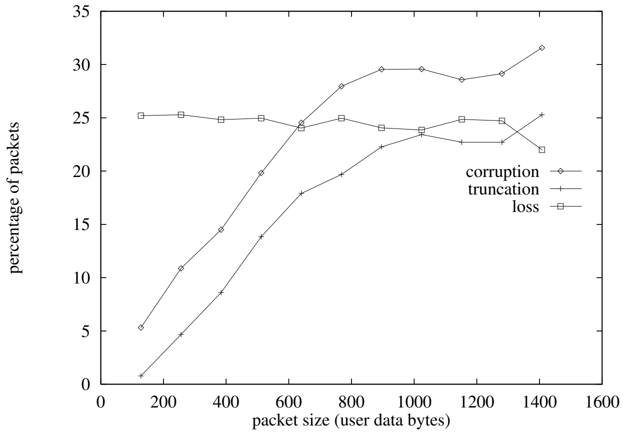

Figure 3.5: Packet injury as a function of packet length. Unsteadiness in this plot may be due to the fact that the interference source was intermittent, so the smaller number of large packets could reduce sampling quality.

We present the results in two forms. The first (Figure 3.4) is a graph of packet outcome as a function of packet length from a trace in which the cordless phone is providing steady interference from a fixed location. Packets that are not lost may be corrupted, truncated, or both (percentages in this figure and the next may add to over 100%). Note that this corresponds to a harsh error environment: with a loss rate of nearly 60% and a corruption rate of almost 40%, the packet error rate without error coding is nearly 100%. Focusing on the truncation rate, we observe that for packets longer than 400 bytes virtually all received packets are truncated, while for shorter packets the truncation rate increases linearly with the (original) packet length.

While Figure 3.4 classifies packets according to how they were injured, it does not describe the severity of the injury (the bit error rate or truncation extent). These figures are presented in Table 3.4. Again focusing on the problem of truncation, we observe that truncation is quite severe for long packets. For example, for full sized packets, usually fewer than half the bits survive, making it difficult to extract useful data from truncated packets. For shorter packets, truncation is less severe. One possible physical phenomenon that could cause this type of packet truncation is clock drift. Since the carrier frequencies of the transmitter and receiver typically vary by a small amount, WaveLAN performs active clock recovery [127]. It seems likely that under unfavorable conditions, clock drift could eventually overcome the clock recovery process, causing the receiver to abort packet reception.

Figure 3.5 demonstrates similar outcomes for a trace collected with machines in the same locations as the "table" trace. The main difference is that in Figure 3.4 the BER of nearly 10^{-2} means that essentially every packet experiences some corruption. Another way to look at this is that the corruption-versus-packet-length curve has already reached its plateau at the shortest packet length.

The above results suggest that packet size reduction is a realistic method for reducing packet truncation rates. For example, Figure 3.4 indicates that reducing the packet size to

Transmit Length	Number Observed	Length after truncation min	mean	std. dev.	% of Packets Truncated	Bit Error Rate
?	104352	0	34.19	104.55	100.00	?
128	7183	96	127.10	4.49	4.83	9.2×10^{-3}
256	4023	96	235.42	42.97	24.76	9.1×10^{-3}
384	3053	96	322.80	90.00	42.94	8.6×10^{-3}
512	2067	96	381.64	139.14	61.97	8.8×10^{-3}
640	2063	96	412.27	179.85	81.10	8.3×10^{-3}
768	2022	96	427.71	198.52	96.88	8.6×10^{-3}
896	2060	96	428.07	202.43	97.57	8.3×10^{-3}
1024	1015	96	436.45	203.77	98.62	8.6×10^{-3}
1152	1015	96	437.43	213.50	98.33	8.7×10^{-3}
1280	1016	96	428.75	217.26	98.72	9.1×10^{-3}
1408	1026	96	436.65	226.21	98.73	9.0×10^{-3}

Table 3.4: Truncation details. Some of the unrecognized packets are probably "lost" test packets; others seem to be interference mistakenly identified by the RF modem unit as a potential packet. The bit error rate is computed over the non-truncated bits of each packet.

400 bytes would cut the truncation rate from 40% to 20% of the received packets. Packet size reduction would need to be combined with retransmission of the truncated (and lost) packets to reduce the error rate to a number that is acceptable to the upper-layer protocols. Even for the extreme error environment used in this section, the link may still be quite useful: if we make the conservative assumption that all truncated packets are useless (that is, no erasure code or sub-packet blocking is employed), shrinking the packet size and applying FEC might still achieve a useful fraction of the link rate. Furthermore, it is interesting to note that even if truncated packet bodies are useless for conveying user data, we may still be able to extract a small Logical Link Control layer header containing error statistics.

As a final note, we observe that the packet loss rate (Figures 3.4 and 3.5) and BER (Table 3.4) appear to be independent of packet size.

3.2.4 Packet loss

The final type of error is packet loss. One possible explanation for packet "loss" might be that our trace analysis is failing to recognize certain packets due to extreme truncation or corruption, i.e., our approach creates false losses. We briefly investigated this hypothesis and found that, while some periods of packet "loss" correspond with arrivals of unrecognized packets, many do not. Furthermore, there are many occasions when unrecognized packets arrive intermixed with sequential test packets.

Figures 3.4 and 3.5 suggest that the packet loss rate is independent of packet size, suggesting another explanation. Each WaveLAN packet begins with a training pattern, a start delimiter, a carrier training sequence, a "network ID", and a standard Ethernet preamble [6].

Though most of these fields are transmitted with physical-layer redundancy, damage to any of them would probably cause the receiver hardware to overlook an entire packet. Table 3.1 does not clearly demonstrate that packet loss and bit error rate are linked, so this explanation may be incorrect.

The problem of packet loss can be addressed by retransmission [36, 11] (or by cross-packet interleaving, if additional latency is acceptable).

3.2.5 Summary

We used error traces to characterize a number of error environments for WaveLAN and we discussed and evaluated error control techniques. The trace data demonstrate only loose correlation among the damage we observed, namely packet loss, packet truncation, and bit corruption. We showed that bit corruption can be addressed using FEC based on Reed-Solomon codes. We also showed that packet truncation can be addressed by sending smaller packets. Both FEC and packet shrinking have the property that they increase communication overhead. As a result, it is not attractive to apply these techniques unless they are needed (lost packets are most easily addressed using retransmission, which is inherently adaptive). Our results show in fact that substantially different degrees of error coding are needed in the different error environments we evaluated. In the next two sections, we propose and evaluate adaptive error control policies.

3.3 Adaptive Error Control Strategies

In this section we discuss design considerations for adaptive error control strategies, implementation issues that may complicate adopting adaptive error control, and present the adaptation policies that we will evaluate in the next section.

3.3.1 Overview

The idea behind adaptive error correction is simple: the receiver collects information about the status of the channel, it summarizes the information into an error estimate, which is transfered to the sender, and the sender adapts how it sends data based on the error estimate. Let us briefly look at some of the implementation options for each of these three components:

- **Error information sources**. A number of different information sources can be used to determine the quality of the error environment. A hardware CRC algorithm may be used for error detection, and most error correction algorithms can provide their clients with an estimate of how close the decoded packet was to containing uncorrectable errors. Alternatively, a wireless network interface may provide signal measurement information which could warn of impending trouble, thus allowing predictive rather than reactive error control. All the policies we evaluate are reactive schemes, based on observed (rather than predicted) errors.

- **Error estimation**. Single error events must be combined to yield an error estimate. This summary will typically be generated periodically by applying a reduction method (average, minimum, or maximum) over some time window. Different methods may be used for different features. The error estimate then must be sent to the sender in a fairly robust way so that the information arrives most of the time. Finally, the sender must deal with the time-critical nature of the information, e.g., it must age state measurements and it must have some method for estimating the results of experiments it hasn't performed recently. Two aging methods are used by our adaptation policies.

- **Adaptation policy**. The sender must modify how it sends data based on the error estimation. This requires an understanding of the dependencies between error types and the possible recovery techniques, i.e., it must know how each type of error can be reduced or eliminated. Note that certain types of errors (e.g., packet loss) may not be avoided by changing how data is sent, but require more radical approaches (e.g., retransmission). The policies we use below adapt the packet size and the degree of redundancy used in FEC.

Adaptation will be effective if the system can collect reliable error estimates and react appropriately to those estimates in a timely fashion, i.e., before conditions change again. There are many reasons why adaptive error control may fail, e.g., it is not understood how errors can be reduced by changing how data is sent, or the changes in error environment happen too quickly to be tracked. Adaptive error control that works well in one environment may not be effective in a different environment. For example, errors on a satellite link and in a LAN environment are caused by different phenomena, and different error control strategies (adaptive or not) are likely to be needed.

3.3.2 Adaptation Policies

To focus on the utility of adaptation, we evaluated three fixed policies and four adaptive policies:

Bold roughly corresponds to an unmodified WaveLAN: maximally-sized packets (five 255-byte blocks) are sent with no error coding. This policy could be considered a pure ARQ link-layer policy: each packet will be transmitted repeatedly until it is received error-free.

Light transmits maximally-sized packets with 5% coding overhead. This is potentially a good policy since many packets are not badly damaged (see Section 3.2.2).

Robust attempts to excel in difficult conditions. It sends minimally-sized packets (one 255-byte block) with nearly one third of each devoted to coding overhead.

Bimodal is a simple-minded adaptive policy. It behaves exactly as "bold" when conditions are good, and as "robust" when they are poor: if two consecutive packets are truncated or corrupted, it sends small, heavily-coded packets until three consecutive packets are not damaged.

Bi-code is like "bimodal" except that it adjusts only coding overhead.

Bi-size is like "bimodal" except that it adjusts only packet size. "Bi-code" and "bi-size" are included in an attempt to understand how much coding and packet sizing contribute independently to the success of "bimodal."

Flex adapts the packet size and degree of FEC redundancy independently. Whenever two or more packets in a window of ten experience truncation, "flex" reduces its estimate of the current safe packet size to 85% of the post-truncation packet length. If three consecutive transactions do *not* experience truncation, "flex" begins to expand the safe packet size estimate, adding first 200 bytes, then 400, then 800, until the estimate reaches the maximum packet size. While "flex" maintains its estimate of the safe packet size in bytes, it rounds this value downward to an integral number of 255-byte blocks. Whenever two or more packets in a window of ten experience a decoder failure, "flex" reduces its estimate of the fraction of each Reed-Solomon block that can carry user data by 15%. If three consecutive packets do *not* experience decoder failure, "flex" begins to expand the user data share, adding first 10 bytes, then 20, then 40, until the entire block carries user data.

It is clear that more sophisticated policies are possible. However, our samples illustrate the space of possible policies and, as we will see below, highlight interesting differences among the trace environments.

3.4 Trace Replay Simulator

Evaluating adaptive error control in a wireless environment is challenging because repeatable experiments are difficult. Repeatability is difficult for two reasons.

The first problem is that a wireless environment cannot easily be isolated, so the dynamic error environment is hard to control. This problem is traditionally addressed by using trace-based evaluation. A set of traces is collected that capture features of the environment that are important to the protocols under investigation. The different protocols are then implemented and evaluated in a simulator or an emulated environment [90, 93] that is driven by the traces. By using the same set of traces for each of protocols, a direct comparison is possible.

This chapter's evaluation of adaptive error control policies follows the simulator approach, driven by the traces presented in Section 3.1 (the next chapter's evaluation of the same policies in an operating system kernel yields substantial confirmation of the simulator results; see Section 4.6.4). The simulator does a one-to-one transfer of errors from packets in the trace to the stream of packets that is submitted to it. If a packet in the trace was lost, the simulator marks the packet submitted to it as lost. If the trace shows packet truncation to

x bytes, the simulator will truncate its packet to x bytes. Finally, if the trace shows bit corruption, the simulator will corrupt the appropriate bits in whatever part of the packet escaped truncation.

The second problem is that the adaptation process itself changes the behavior of the system, so we must replay traces in situations different from the ones in which they were collected. In our case, the packet contents may change as a result of FEC and the packet size may be changed to reduce truncation errors. The correctness of applying an error trace collected using one packet stream to a different packet stream depends strongly on the characterization of the errors presented in Section 3.1. Let us consider both changing the packet contents and packet size.

3.4.1 Varying Packet Contents

Applying bit corruption information collected using one set of packets to a set of packets with different contents (e.g., as a result of adding FEC) is only correct if errors are not sensitive to packet contents. This sensitivity could arise in multiple ways. Clock synchronization can be impaired by long runs of all-one or all-zero bits. Furthermore, indoor high-speed wireless LANs may suffer from inter-symbol interference, which occurs when a delayed echo of one symbol interferes with the arrival of the next symbol. Also, changing packet contents will change outcomes if channel errors typically force whichever bit is being transmitted to a certain value (i.e., in some channels most errors occur when a one becomes a zero), in which case it would be inaccurate for a simulator to replay bit errors as inversions.

Given the particular coding used in WaveLAN,[1] one would expect error characteristics for a particular experiment to be independent of the particular data contents. To evaluate whether this is true, we chose a set of packet bodies whose contents vary significantly and then observed whether they exhibited different error behavior. We chose the 128-bit and 1024-bit Hadamard-Walsh codes [113], sets of orthogonal code words generated by a recursive algorithm (see Figure 3.6). We augmented the sets by adding an all-zero code word. Each set of codes includes a wide variety of run lengths and alternation patterns (for example, H_4 contains runs of length 1, 2, and 4). These sets are extensive but clearly not exhaustive. By aggregating multiple trace runs from each of the interference scenarios, we collected approximately 270,000 packets, or about 2100 apiece of each of the 129 different content types, representing roughly 20 minutes of network activity.

To evaluate whether the error rates are sensitive to packet contents, we plotted histograms of how many packet types experienced different rates of loss, truncation, and bit corruption. To the extent that these errors are independent of packet type, we would expect to see tight clustering of error rates around a central value with few outlying points, i.e., approximations of normal curves. The results are shown in Figure 3.7. For example, every packet type was truncated at a rate between 0.8% and 1.7%, with the greatest number of packet types

[1]WaveLAN effectively transmits an 11-“chip” sequence for each bit, resulting in many transitions regardless of user data; the WaveLAN designers expected indoor multipath delay of up to 250 nanoseconds [127], approximately one quarter of a dibit symbol time, which would mean that most inter-symbol interference would be between chips belonging to a single dibit.

$$H_M = \begin{bmatrix} H_{M/2} & H_{M/2} \\ H_{M/2} & -H_{M/2} \end{bmatrix} \quad \text{with} \quad H_2 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\text{example} \qquad H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.6: Walsh code matrices, defined by a simple recursion. Each matrix row forms one code word.

truncated at a rate of approximately 1.2%. While more extensive data could be collected and more rigorous analysis could be applied, these results indicate there is no dramatic sensitivity to packet contents. The remainder of this dissertation will assume that bit corruption and truncation patterns can be collected and then applied in a meaningful way to packet streams with different contents.

3.4.2 Varying Packet Size

Applying the truncation information collected using one packet size to a (shorter) packet is valid only if the chance of truncating a packet to x bytes is the same for all packets (longer than x), and is independent of the packet contents. Table 3.5, which presents received packet length after truncation versus transmitted packet length, is based on the same trace information presented in Table 3.4. If post-truncation received packet lengths were *a function of* the transmitted packet length, we would expect that the percentage of packets truncated to a certain length (entries in one row of Table 3.5) would vary noticeably across columns, which we do not observe. If, on the other hand, truncation lengths are independent of transmitted lengths, we would expect that rows would be internally consistent and different from each other. For example, the "448" row indicates that roughly 10% of all packets arrive with lengths in the 448-to-511-byte range, despite transmitted lengths ranging from 512 to 1408 bytes. However, the rows labelled "768" through "1408" show that essentially no packets of any transmitted length arrive at the receiver with more than 768 bytes present. This justifies treating a trace record containing a truncation to x bytes as an indication that any packet shorter than x bytes would escape truncation and that any packet longer than x bytes would be truncated to x bytes.

If we wish to also apply (part of) a long packet's bit corruption syndrome to a simulated short packet, we would wish to validate this by observing the bit error rate to be independent of packet length; Table 3.4 supports this approach.

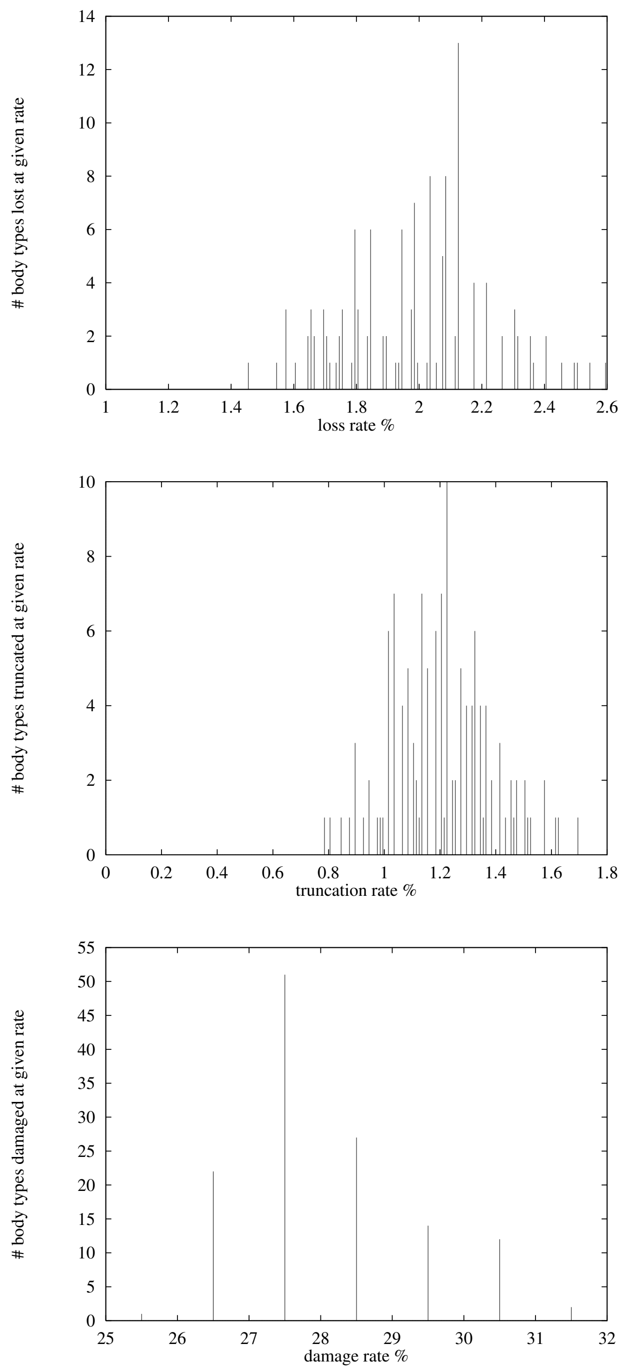

Figure 3.7: Histograms of per-packet-type loss rates (top), truncation rates (middle), and truncation rates (bottom). Note that these graphs capture the type, but not the degree, of damage.

3.4.3 Simulator Details

The architecture of the simulator is shown in Figure 3.8. The policy module implements a specific error correction strategy. Its input is information on the errors that affected previous

		Transmitted length										
		128	256	384	512	640	768	896	1024	1152	1280	1408
	0	-	-	-	-	-	-	-	-	-	-	-
	64	4%	5%	4%	5%	5%	5%	5%	4%	4%	4%	4%
	128	95%	9%	9%	8%	11%	10%	10%	9%	9%	10%	9%
	192	-	10%	8%	9%	9%	8%	10%	8%	9%	11%	11%
	256	-	75%	9%	9%	8%	8%	8%	9%	9%	9%	8%
	320	-	-	10%	9%	9%	10%	10%	11%	11%	8%	10%
	384	-	-	57%	9%	9%	9%	8%	10%	8%	8%	8%
	448	-	-	-	9%	10%	8%	10%	8%	9%	9%	9%
	512	-	-	-	38%	8%	9%	9%	9%	8%	10%	10%
	576	-	-	-	-	9%	9%	9%	9%	10%	9%	9%
Received	640	-	-	-	-	18%	8%	8%	9%	9%	7%	8%
Length	704	-	-	-	-	-	7%	6%	6%	5%	6%	5%
	768	-	-	-	-	-	3%	0%	0%	0%	0%	1%
	832	-	-	-	-	-	-	0%	-	-	0%	-
	896	-	-	-	-	-	-	2%	0%	0%	-	-
	960	-	-	-	-	-	-	-	0%	0%	0%	0%
	1024	-	-	-	-	-	-	-	1%	0%	0%	0%
	1088	-	-	-	-	-	-	-	-	0%	-	0%
	1152	-	-	-	-	-	-	-	-	1%	0%	-
	1216	-	-	-	-	-	-	-	-	-	-	0%
	1280	-	-	-	-	-	-	-	-	-	1%	-
	1344	-	-	-	-	-	-	-	-	-	-	0%
	1408	-	-	-	-	-	-	-	-	-	-	1%

Table 3.5: Received lengths versus transmitted lengths. The column headings are transmitted lengths, and the row labels are the floors of histogram buckets: row 0 represents packets received with length 0 through 63. Note that the leftmost non-blank entry of each row represents packets that were received without truncation. Columns may not add to 100% due to accumulated roundoff errors.

packets. It periodically makes a "send" decision, which consists of specifying two numbers, a count of 255-byte blocks to send and the number of user data bytes each block will carry (the remainder is used to carry Reed-Solomon parity information). The policy module currently has instantaneous access to the error estimate generated by the decoder on the receiver. This represents a delay-free, error-free back channel. While this can be approximated in many wireless LAN environments by piggybacking information on MAC-level acknowledgements (see Section 4.3.1), it would be less practical for end-to-end protocols (see Section 4.2.1). The packet is then encoded using a software Reed-Solomon encoder, and forwarded to a module that models the error characteristics of the wireless link.

The channel model takes an error trace and uses this to decide what damage to inflict on the packet, as we described above. In order to ensure that the trace's bit-corruption patterns

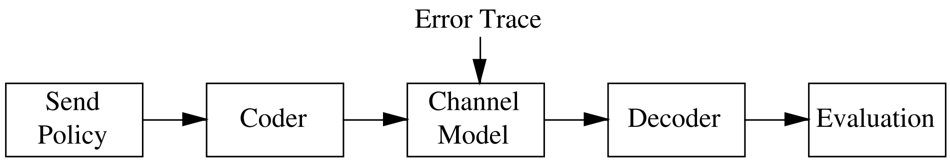

Figure 3.8: Simulator architecture

are long enough to cover any packet that our adaptation policy might choose to send, we use only traces of long fixed-size packets. The transmission result is then forwarded to the Reed-Solomon decoder, which will attempt to decode it. The decoder forwards the error information for the packet to the evaluation module, and also generates an error estimate for the policy module on the sender. Our traces contain a small number of packets which experienced bit corruption in only the Ethernet or IP envelopes, for which we cannot reconstruct the exact corruption pattern. The simulator treats these events as if no corruption had occurred. While this is inaccurate, these packets are quite rare (28, or 0.5%, in the "walls" trace, and no more than 4 in any other trace).

For each packet, the evaluation module is told whether the packet was lost, how many blocks were not truncated, and how many blocks were successfully decoded. This results in a *packet efficiency* figure: the number of correct bytes delivered to the receiver (zero in the case of packet loss) divided by the "air time," the amount of time required to send those bytes, however they were encoded, plus WaveLAN, Ethernet, IP, and UDP headers and trailers. While one would like to see an efficiency of 100%, packet headers and inter-packet gaps limit the efficiency to 94%, even in an error-free environment.

We wish to compare the efficiencies of different error control policies, each transmitting continuously, under a variety of error environments. Calculating the efficiencies of policies adapting to a specific trace is straightforward if, during each transmission interval, all policies use the same packet size as the trace packet: a policy's efficiency is the total number of user bytes delivered correctly divided by the total airtime.

Calculating efficiency is more complicated if policies use different packet sizes. The fundamental problem is that the trace has incomplete information on the error environment, e.g., no information is available on the environment during the inter-packet gap, so it is not possible to precisely simulate the errors observed by each policy. We can use the per-packet simulation outlined above, but we must apply a weight to the effect of each packet. Since each simulated packet transmission "consumes" a trace record even if the packet is shorter than the trace record, a policy that reduces packet size under high error conditions would be billed for less air time than a policy that uses fixed packet sizes. Since each trace record represents a certain amount of time, the effect would be that, in the simulation, the adaptive policy would spend less time in the bad periods than the fixed policy, and would thus be experiencing a different error environment, rendering efficiency comparisons impossible.

We correct for this by scaling the effects of each packet (bytes delivered and airtime consumed) by the ratio of the trace record's airtime to the transmitted packet's airtime. In effect, during each trace interval, a policy can send one maximally-sized packet or one and some fractional number of smaller packets. We can then compare policies in a fair manner. One minor drawback of this solution is that it artificially limits the adaptivity of algorithms, since adaptation is only allowed at fixed time intervals, rather than after every packet exchange.

3.5 Simulation Results

In this section we will present the results of operating the seven adaptation policies presented earlier in each of the five simulated error environments. The results indicate that static policies are unlikely to perform efficiently and robustly as error conditions vary and that straightforward reactive adaptation policies work well.

Table 3.6 presents the numerical efficiency ratings for the different adaptation policies. Headers, trailers, and inter-packet gaps limit these numbers to 94%. The last column, "oracle," represents the efficiency of a policy using approximately the optimal packet size and FEC parameters for each transmission slot. This is implemented by processing the error trace into an "optimal choice" trace. A truncation event causes "oracle" to choose a packet size rounded down from the observed truncation length to the nearest 255-byte Reed-Solomon block boundary. The error coding level for each transmission slot is chosen by repeatedly simulating the Reed-Solomon decoding process, varying the coding level from 0% to 30% by 5% steps, choosing the smallest coding level that results in all blocks being decoded successfully. While this policy's precise advance knowledge of every error event gives it an unfair advantage over reactive policies, it provides a good basis for comparison.

Environment	Bold	Light	Robust	Bimodal	Bi-code	Bi-size	Flex	Oracle
Office	94%	89%	53%	94%	94%	94%	94%	94%
Walking	89%	87%	53%	89%	89%	89%	89%	93%
Adjacent	43%	41%	32%	50%	43%	44%	45%	54%
Table	.06%	15%	49%	26%	33%	17%	58%	76%
Walls	69%	84%	52%	72%	73%	72%	81%	91%

Table 3.6: Performance of different error control policies.

Figure 3.9 shows the same data graphically: it shows, for each policy, its efficiency in different scenarios, normalized in each scenario to the efficiency of the "oracle" policy. The figure makes it clear that no single policy was best in all environments. In fact, all policies but "flex" performed poorly (less than 70%) in at least one scenario. However, every policy but "robust" scored above 95% (i.e., near-perfect efficiency) in at least one environment, suggesting that many policies have some merit.

Several fixed policies performed well in several environments. For example, "bold" performed acceptably in all but one case—even in the harsh "adjacent" environment, its large-packet gamble sometimes pays off well. The "light" policy, which represents a plausible fixed FEC policy, also performed well in all situations but one. "Robust" is the opposite of "light," performing well in only the "table" case.

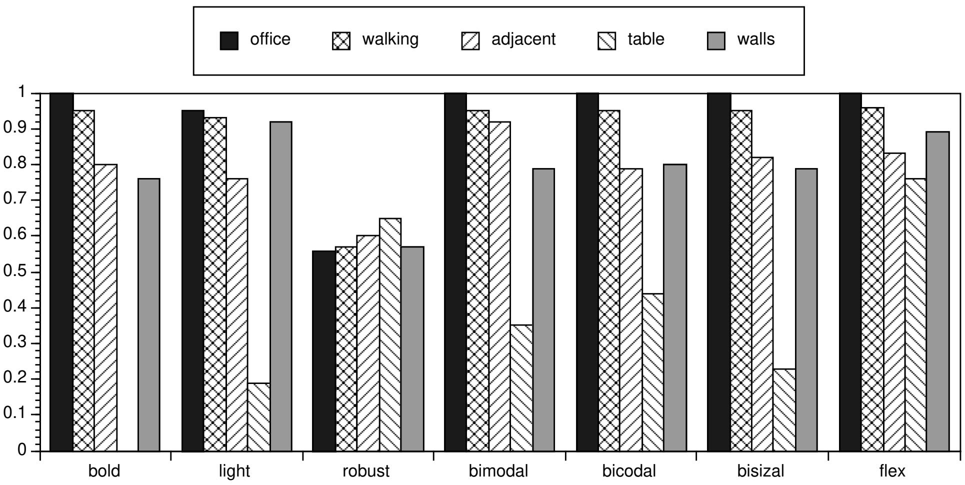

Figure 3.9: Comparison of normalized performance for each policy. For each policy, each bar represents that policy's performance in one scenario (office, walking, adjacent, table, or walls) normalized to the performance of the "oracle" in that scenario. This comparison shows that "bold" and "light" perform well except for one case, "robust" always performs satisfactorily, and "flex" always does well.

Simplistic adaptation policies show more promise, especially in the "table" case, where almost every packet is lightly corrupted. The "table" row of Table 3.6 shows that both "bimodal" and "bi-code" outperformed "bold" by an embarrassing amount and outperformed "light" by almost a factor of two. The "bi-size" policy shows that packet size is important in high-truncation environments: it performed nearly as well as the "bimodal" and "bi-code" except in the "table" case, where aggressive error coding is essential. Comparison among these three adaptive policies suggests that independently adjusting packet size and coding overhead is superior to linking them. For example, it is effective to adjust packet size and coding overhead for the "adjacent" environment, but it is better to adjust only the coding overhead in the "table" scenario.

"Flex" independently adjusts packet size and coding overhead, and also incorporates longer-term channel history. The "flex" column of Table 3.6 shows that, while "flex" is often not the star performer, it is always within 10% of the best (non-oracle) performer for each trace.

The above results clearly show that which policy performs best depends on the scenario. This raises the question of which policy one should implement. We argue that consistent good performance across a wide range of scenarios (as delivered by "flex") is more important than being the best performer in any particular scenario. For example, the fact that "robust" performs best in the "table" scenario will do little to cheer up people working in "office" conditions. This argues that the adaptive policies, especially "flex", are preferable to the fixed policies.

3.5.1 Summary

We believe that the results of this section make a strong case that adaptive error control policies are both useful and practical. Any mobile host is likely to encounter each of these scenarios (among many others) in its lifetime, so being able to adapt the error policy is important; this type of adaptation is easy since it happens on a time scale of minutes. Both the "walking" and "adjacent" case involve adaptation inside the trace, and are more challenging since changes happen on a smaller time scale. However, the results show that it is possible to track even these rapid changes. While there is room for investigating a wider variety of information sources and measurement and estimation approaches, the simple "flex" policy seems likely to suffice in many common environments.

3.5.2 Applicability of Results

Our adaptation results are based on the observation that some changes in the error environment are slow because they are caused by human actions such as mobility. We believe that similar results will be obtained for other LANs that face errors that can be ameliorated by modifying transmission resource consumption (via power control, error coding, packet shrinking, or even adaptive medium access protocols). Note that while adaptive error control has been applied to other wireless environments such as satellite links and cellular phones, differences in both the error environment and how it changes mean that our results may not generalize to these domains.

3.6 Related Work

3.6.1 Adaptive Forward Error Correction

If channel conditions such as error patterns or bandwidth available for redundancy change over time, a single error correction algorithm may be inappropriate. While it is possible to use several different FEC codes, each with dedicated hardware resources, more attractive schemes exist. Hagenauer presents a family of codes called rate-compatible punctured convolution codes [55] which use the popular Viterbi decoding algorithm [130, 49, 117]. One example code family has 13 codes with redundancy overhead varying from 12.5% to 300%. Qualcomm, Inc., a manufacturer of digital radio systems, offers a single-chip encoder/decoder designed for satellite channels which operates at up to 25 Mb/s with four levels of redundancy [108]. Karn presents a variable-rate FEC system designed for implementation by general-purpose microprocessors [65] and evaluates its operation on a link subject to short but frequent interference from radar. A study of dynamic adaptation to both channel noise and CDMA multiple access interference may be found in [43].

3.6.2 Wireless Link Error Characteristics

Since solving the problem of high error rates depends critically on understanding the types and frequencies of errors, many studies have characterized wireless LAN error environments [112, 37, 38]. We present more-detailed information (characterization of errors into loss, truncation, and corruption, including error syndromes) which allows us to evaluate the impact of specific error control mechanisms (packet shrinking and FEC) and adaptation strategies.

RFC 2041 [92] describes an extensible wireless environment trace format and matching trace replay tools, described further in [90, 93]. However, the system represents packet errors rather than the sub-packet error phenomena (truncation and bit corruption) which were necessary to obtain the positive results of this chapter. Designing and evaluating adaptive error control mechanisms would benefit from even finer-grained trace data, i.e., "soft-decision" symbol traces provided by the digital signal processing layer of a receiver.

3.6.3 Existing Adaptive Links

A final area of related work is adaptive error control for wireless links. Satellite communication has been studied widely, and since error rates can be very high, satellite links make extensive use of FEC to improve communication performance [107, 126]. In some cases, adaptive FEC is used, but the conditions are very different from those in wireless LANs: the causes and nature of the errors are very different, and the long roundtrip times means that adaptation is many orders of magnitude slower than in a LAN. The DARPA Packet Radio Network [62], a metropolitan-area mobile radio network designed to carry data and voice traffic, included a multi-rate FEC codec, although we are not aware of any published performance evaluation of this implementation. The Proxim RangeLAN frequency hopping system has two data rates, the primary rate of 1.6 Mb/s, and a secondary rate of 800 Kb/s, used to extend the usable range [105]. While having two symbol rates is more flexible than having only one, our results suggest that multiple rates would allow efficiency in more scenarios. In addition, selecting an appropriate symbol rate for each transmission requires an adaptation policy which could profit from evaluation similar to ours.

3.7 Conclusion

We began this chapter with a series of questions. We can now answer them as follows:

- Our work in this area has demonstrated that common environmental challenges can cause serious levels of bit corruption, packet truncation, and packet loss for WaveLAN. While the exact error patterns are likely to be specific to WaveLAN, most wireless LANs are faced with similar threats, such as attenuation, symbol corruption, and clock drift.

- Error correction codes and packet shrinking prove useful against the error patterns we encountered.

- It is easy to demonstrate situations in which fixed error control policies are very undesirable.

- Luckily, adaptive error control policies which are simple to implement also perform acceptably close to the ideal, and a single policy shows promise across a variety of error environments. While this particular policy's good performance may well be specific to WaveLAN, it seems plausible that other networks subject to varying error environments caused by human-timescale events (such as mobility of people and computers and operation of competing wireless radiation sources) are likely to be able to ameliorate these errors by adaptively modifying the offered traffic load via power control, error coding, packet shrinking, or even adaptive medium access protocols. Note that while adaptive error control has been applied to other wireless environments such as satellite links and car phones, differences in both the error environment and how it changes mean that results cannot be generalized to these domains.

Finally, because of the difficulty of controlling wireless environments, repeatable experiments are difficult. This makes trace-based evaluation important. Sharing traces, as is done by researchers working on memory caches and disk subsystems, would further facilitate comparative evaluation, since a direct comparison of results by different research groups becomes possible. As a first step in creating a shared set of traces, we are making our traces available at `http://www.cs.cmu.edu/~davide/wltraces/`.

Now that we have demonstrated the potential utility and practical feasibility of link-level adaptive error control, the next chapter will investigate its applicability in a real-world multi-layer, multi-protocol environment.

Chapter 4

End-to-end Evaluation of Local Adaptive Error Control

4.1 Introduction

Chapter 3 provides a detailed understanding of the WaveLAN error environment and error control techniques that effectively address each error hazard. In addition, it demonstrates that adaptive error control is both important and feasible. Evaluating adaptive error control in the real world requires design decisions, a prototype implementation, and measurement under realistic conditions. In particular, we seek answers to the following questions:

- Even after employing error coding and packet size adjustment, a non-trivial fraction of packets will be lost at the link layer. When should the necessary retransmissions be per-link versus end-to-end?
- Do we need new end-to-end transport protocols or protocol-specific accelerators in the network, or can we use protocol-blind error control?
- Do our encouraging single-hop results generalize to more realistic network topologies?
- Do our encouraging simulation results generalize to actual TCP? For example, do fluctuations in link-level throughput restrict TCP's ability to take advantage of it? Is there significant waste due to TCP retransmitting data that link-level error control is also retransmitting?

The remainder of this chapter is organized as follows. In Section 4.2 we discuss the trade-offs between local and end-to-end error control and consider local error control design issues. A description of our experimental approach, including Medium Access Control (MAC) and Logical Link Control (LLC) protocols, is found in Section 4.3. We describe the measurement framework employed in Section 4.4. Section 4.5 evaluates link-level retransmission in multiple error environments and multiple network environments while observing how well it interacts with TCP. Once this basic robustness has been demonstrated, Section 4.6 describes packet-shrinking and error coding extensions to the LLC layer, which are subjected to trace-based evaluation in Section 4.6.2. We discuss related work in Section 4.7 and summarize in Section 4.8.

We will show that local, protocol-independent, persistent, adaptive link-level error control allows TCP to obtain substantial throughput in dynamic, harsh wireless LAN error environments.

4.2 Local error control

Given the wide variety of approaches to solving the end-to-end problems caused by wireless link errors, we will briefly justify our two main architectural decisions: we pursue local per-link error control rather than end-to-end modifications to transport protocols, and our link-level error control operates independently of the higher-level transport protocols rather than targeting support toward particular ones.

4.2.1 Local versus end-to-end error control

Addressing link errors near the site of their occurrence appears attractive for several reasons.

First, entities directly connected by a link are most able to understand and manage its particular characteristics. For example, it seems impractical for end-systems to track the current propagation delay for each link they use, nor feasible for them to know which packet loss events represent congestion versus intermittent link errors (TCP's assumption that most burst loss is due to congestion is a well-known incorrect simplifying assumption). Moreover, entities on the link are likely to be able to respond more quickly to changes in the error environment. Certain error control techniques, such as soft-decision decoding and energy combining, are difficult or impossible to implement on an end-to-end basis.

Second, end-to-end error control necessarily involves multiple error-free links in solving a purely local problem. If an error-prone wireless link is the last hop on a path from a distant server to a mobile client, end-to-end retransmission demands bandwidth on every link, while local retransmission requires extra bandwidth only where it's truly needed. This argument may become increasingly important if the Internet widely adopts flow-specific packet handling and reservations. Requiring every application to reserve enough extra bandwidth on every link to work around difficulty on its worst link would be inefficient.

Third, as a practical matter, deploying a new wireless link protocol on only those links that need it is easier than modifying transport code on millions of deployed wired machines. Application-level proxies address this problem to some extent, but they are currently constrained to running on end systems, whereas local error control can operate on exactly the links that require it (such as a point-to-point link connecting two LANs).

Despite these attractions, traditional Internet engineering principles [119, 24, 45] and an early simulation study [36] argue against "too much" per-hop reliability. In the case of local error control for links with highly variable error rates there are at least three dangers. First, local error recovery mechanisms may alter the characteristics of the network, which could confuse higher layer protocols. For example, local retransmission could result in packet reordering or in large fluctuations of the round-trip time, either of which could trigger TCP timeouts and retransmissions. Second, local and end-to-end error control are adaptive mechanisms that may respond to the same events. This could result in undesirable interactions, causing inefficiencies and potentially even instability. A very simple example is duplicate retransmissions generated by the local recovery protocol and by TCP. These can result in excessive link bandwidth consumption or, worse, queue overflow. Finally, a given data packet may bear information with a limited useful lifetime, so any retransmission may be wasted effort. For example, it is better to drop a late audio packet than to retransmit it, since retransmission may make the next packet late as well.

Given the significant advantages of local error control, we will pursue a purely local approach engineered to avoid the drawbacks mentioned above.

4.2.2 Design tradeoffs for local error control

Once we decide to address wireless link errors locally, we are faced with several options, ranging from purely local solutions to solutions where the local error control interacts with the higher layers in the protocol stack to avoid undesirable effects.

Adaptive codecs and multi-rate modems [74, 105] are useful underlying technology, but do not represent a complete solution. While an intelligent modem could manage a simple point-to-point link, in a shared-channel LAN different mobile stations may experience widely varying error conditions (due to, e.g., different locations [38]). Tracking the appropriate coding level for different stations would be challenging for a transparent low-level hardware approach. Also, different flows to a single station might require different error control strategies to trade off latency and reliability.

"Pure" link-layer approaches such as MACA [63], MACAW [20], and IEEE 802.11 [70] apply error control on a per-packet basis and do so in a protocol- and application-independent fashion. These mechanisms can potentially be made "flow-aware" by tailoring the level of error control according to the declared needs of the flow. This could be as simple as allowing each flow to enable or disable link-level retransmission, or more complex, such as bounding retransmission of each packet according to a specified lifetime. Flow information could be obtained through protocols such as RSVP [26] or IP Differentiated Services marking [91, 78].

"Protocol-aware" link-layer protocols [11] may inspect the packets they pass in order to give special treatment where it is most needed. While this can significantly improve performance, it requires gateways to understand a wide variety of transport or even application protocols. The Internet currently carries a wide variety of incompatible streaming audio and video protocols, and this situation seems likely to persist. The development of protocols and the development and deployment of parsers could evolve into an "arms race" between protocol designers and network administrators.

Solutions that require network support at the highest level in the protocol stack are "gateway-style" or "indirect" error control [9, 29, 11, 135] which perform significant and stateful protocol translation, or even data transcoding [136, 50], at the border of two subnets with greatly differing characteristics. In addition to potentially needing to understand multiple protocols, this approach faces significant challenges when network routing changes, as state must be migrated from one gateway to another.

In this chapter we evaluate the performance of "pure" link-layer error control. The first reason is its simplicity. A second reason is that a number of protocols incorporating link-layer error control are being defined and deployed (e.g., 802.11 [70]), so information about TCP interactions may be valuable to these efforts. We focus on the case of reliable data transfer using TCP, which is by far the most widely used transport protocol. Our approach is purely link-layer in the sense that it treats packets as opaque, not depending on the TCP (or even IP) specification or implementation, although, as we discuss below, our results suggest that link-layer error control may benefit from awareness of flow-specific characteristics.

Our evaluation consists of two parts. First, we will observe how purely local error control improves the performance of TCP, operating in a wide variety of situations. To understand the interactions between local error control and TCP, we use simple local error control based on local retransmission only (Section 4.3.3) and apply synthetic burst loss patterns designed to provide particular stresses (Section 4.5). These results provide a LAN-independent case for straightforward, protocol-independent local error control increasing TCP throughput under a broad set of conditions. Second, we present a more realistic evaluation based on a more sophisticated adaptive local error control implementation (Section 4.6) and WaveLAN-specific error traces (Section 4.6.2).

4.3 Experimental Approach

In order to investigate the feasibility and desirability of adaptive link-level error control, we built a small experimental framework on top of WaveLAN I (Section 1.6). In addition to a bare-bones polling medium access control protocol, we added a logical link control layer that provides reliable data transfer, including packet fragmentation and reassembly, over multiple parallel flows per mobile client. This experimental prototype includes trace replay mechanisms that allow us to apply the same real-world error traces we used in the previous chapter to an actual TCP implementation running in an operating system kernel. The increased realism this represents comes at a cost, since both repeatability and fidelity are somewhat reduced.

4.3.1 MAC design

The goal of our MAC protocol design was building a vehicle for investigating the design and effectiveness of local error control and interactions between protocol layers, rather than designing a new standard or validating an existing one. Our focus is on a pico-cell environment with a small number of mobile stations and a backbone-connected base-station responsible for connecting the cell to the rest of the Internet.

Our resulting MAC is based on master/slave transactions. A master, presumably the base station, periodically sends a message to each mobile slave; the message can transfer data to the slave, and may invite the slave to transfer data itself. The MAC transactions can be divided into two types. First, the INVITE and JOIN messages are used to add new mobile hosts to the master's polling list. Second, POLL-DATA and DATA-ACK are used for data transfers. If the cell master wishes to transmit a packet to a slave, it issues a DATA message, and the slave immediately returns a ACK reply. The cell master can provide a slave with an opportunity to transmit data by issuing a POLL message, which allows the slave to issue DATA. In many instances, the master will call for a two-way data exchange by sending a single packet containing POLL and DATA, which will cause the slave to generate a single packet containing DATA and ACK.

While packet loss is often considered a challenge for LLC to resolve, it has MAC-level implications as well: the cell master must shift focus from one mobile station to another even when packets are lost. Given the polled, contention-free nature of our MAC and the low propagation delay in a LAN, the master can use simple timeouts to recover from packet loss. This approach is not universally applicable. In particular, high satellite propagation delays demand alternative approaches [126].

Each mobile station that is a member of a cell communicates with the master via one or more bi-directional "channels," which have packet queues associated with them, and are independently polled by the master. Packets are filtered onto channels according to a simple scheme that allows specifying or wildcarding hosts, protocols, and port numbers; this specification is similar to the specification used for RSVP sessions [26].

The main packet elements used by the master and slave are summarized in Table 4.1. While the MAC supports per-channel reservations, this will not be discussed until Chapter 5.

Command	Issued by	Function
Invite	master	solicit new slave stations
Join	slave	specifies IP address, (Ethernet) station address
New channel	slave	requests a new channel; specifies filter information
Kill channel	slave	deletes a channel
Scheduling request	slave	suggests scheduler parameters for a channel
Poll	master	solicitation for the slave to send a packet;
		specifies slave, channel
		acknowledges slave's last transmission on this channel
Ack	slave	acknowledges master's last Data transmission
Data	master	sent unsolicited by master
	slave	packet sent after solicitation via a Poll

Table 4.1: Summary of wireless MAC protocol. Slave stations piggyback channel management requests (New, Kill, Scheduling request) onto Ack responses.

The operation of the MAC is similar to the IEEE 802.11 Point Control Function [70]. Clearly, many features could be added. For example, one could add contention periods, similar to the 802.11 Distributed Coordination Function (DCF).

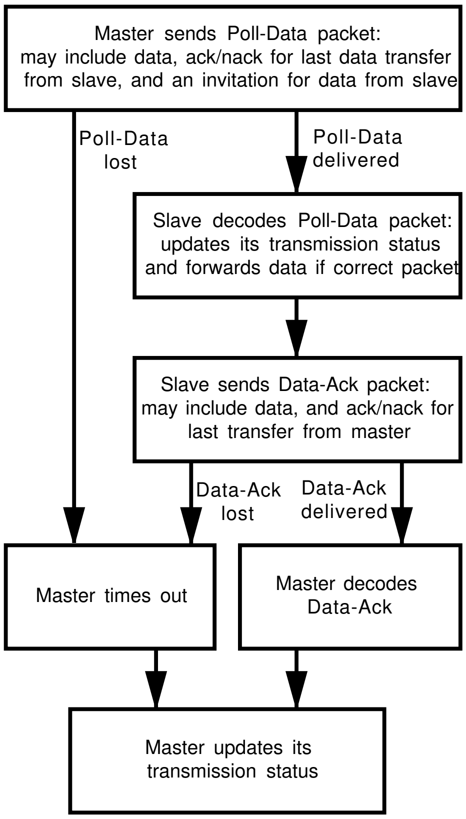

Figure 4.1: MAC and LLC actions

4.3.2 LLC design

In a wireless environment, data packets may be lost entirely, partially lost due to truncation, or corrupted by bit errors. The simplest error control mechanism, stop-and-wait retransmission, addresses all three as follows. Both parties (master and slave) keep copies of each data packet until they receive an acknowledgement from the other party via sequence number fields in the Poll and Ack commands. Stop-and-wait retransmission is acceptable in our environment because of the feasibility of immediate link-level acknowledgements, common to many wireless LAN MAC protocols [70, 63, 20]. Links with high latencies [44, 83, 31] may require sliding-window LLC.

The resulting actions executed by the MAC and LLC layers on both master and slave are shown in Figure 4.1.

4.3.3 Implementation and Performance

Our hardware platform is Intel 80486 and Pentium laptops using 915 MHz PCMCIA card WaveLAN units. We disabled the WaveLAN backoff protocol by programming the embedded Ethernet controller to make only one transmission attempt per packet. This allows us to schedule non-colliding packet transmissions. The master/slave polling protocol was then implemented in software on top of this basic packet transmission function. The master continuously polls slaves, mixing INVITE requests for new mobile hosts with POLL-DATA requests to known mobile hosts in the cell. In our experiments, one laptop operates as a master/base station, while the other laptops use the slave protocol. The protocol is implemented in a kernel device driver for NetBSD Unix.

The WaveLAN hardware has a nominal throughput of 2 Mbit/s, which drops to about 1.8 Mbit/s after considering the effects of headers and standard MAC overhead. A standard Ethernet-style WaveLAN driver achieves 1.4 Mbit/s for a single TCP stream between two hosts, compared to .8 Mbit/s when using our MAC protocol. This 43% loss in throughput is a result of the stop-and-wait error control (described above) and the high per-packet overhead of our software MAC implementation. However, this throughput is credible, and a reasonable starting point for our research.

Clearly, MAC designers with control over hardware-level microengines can improve on this implementation, especially in regard to performance. The software path on the slave to respond to a poll from the master represents a non-trivial source of overhead.

4.4 Measurement framework

Our measurements were performed using laptops running our experimental system. Our "base station" was a 25 MHz 80486 DECpc 425SL and our client was a 75 MHz Pentium Toshiba Satellite Pro 400CDT, both running NetBSD 1.2. The base station was connected to our campus Ethernet and routed traffic between it and the client. There was no attempt to provide a "clean room" network environment or to make tests only at certain times of day, but we didn't observe substantial variation between experiments.

Performing repeatable and comparable experiments in a wireless environment is challenging because the events in the wireless environment are hard to control. We address this problem by relying on a "packet killer" in the kernel device driver to drop or corrupt packets in a controlled fashion. An administrative tool is used to load a list of packet error events into the device driver. The driver can be instructed to apply the error event list to transmitted packets, received packets, or both. Each error event specifies a packet length threshold and an error coding parameter.

If the length of the packet being received or transmitted exceeds the length of the error event's threshold, the trace replay process should arrange for the packet to be truncated (a threshold value of zero indicates that a packet loss should be simulated regardless of transmitted length). This is implemented as follows: The packet killer on the sending machine, the receiving machine, or both, will handle a truncation by prefixing the packet with a special header flag specifying the truncation length. The receiving machine's LLC layer will be informed of the truncation length so it can serve as an input to the adaptation policy process, but the packet contents will otherwise be ignored.

Since we lack hardware support for forward error correction (FEC), the "packet killer" emulates the effect of FEC. Each error trace is processed off-line so that bit corruptions in a trace event are summarized by the minimum-strength Reed-Solomon code required to decode the packet. This number is obtained from our bit-level corruption patterns in a conservative fashion: even short user data blocks require a full block's worth of parity symbols, and a packet killer event encodes the code required to correct the most-damaged block in the packet. The transmitting LLC layer inserts a PAD command at the front of the packet body that both specifies the code strength in use and adds blank bytes to the packet so it is the same size as if it had been encoded with the appropriate Reed-Solomon block code. If the error encoding used by the packet is not as strong as the one specified in the appropriate "packet killer" event, a decoder failure is indicated.

Each packet "consumes" an error event from the list; when the list is exhausted, the device driver begins executing it from the beginning.

The packet killer emulates packet loss on output by setting the Ethernet destination MAC address to a constant unused by any WaveLAN unit. Since each "lost" packet is actually transmitted, exactly the appropriate amount of air time is consumed. On input, a loss event is handled by the device driver's interrupt routine immediately discarding the packet.

The "packet killer" framework allows a variety of emulated error environments. Simple deterministic traces, such as "lose every fifth packet," can be constructed by hand. More-realistic patterns that fit desirable statistical distributions can be generated off-line via random-number processes. Finally, we can convert the same real-world error syndrome traces we used in the previous chapter to the packet killer's event format.

By replaying the same list of packet error events, we can repeat experiments, or directly compare the effectiveness of different error control strategies in the same (emulated) error environment.

Throughput numbers are based on wall clock time starting when the TCP connection establishment handshake is complete and ending after the teardown is complete. Unless otherwise stated, when we apply a given loss pattern to a connection, we apply it in the direction of data flow (that is, packets bearing data are dropped, but acknowledgements are not), because the effect on TCP throughput is greater and more easily understood. Since TCP acknowledgements are cumulative while data packets are not, losing data packets is more harmful.

In some cases we present processed output from TCPDUMP. Our output tap was placed in the device driver at the beginning of the "output" driver call so that we could observe packets sent by TCP even if they were dropped due to queue overflow at the transmitter. Some other Ethernet drivers place the output tap in the "start" driver routine, after any queue dropping has taken place, so TCPDUMP observes only packets that are actually transmitted. This means that our traces more closely reflect what the sending TCP attempted to transmit, while other traces more closely reflect which packets are actually sent on the wire.

In situations where we report numbers from multiple TCP streams, our transmission program used multiple Unix processes and coordinated them so that all began transmitting at the same time and ceased transmission when the first stream was complete.

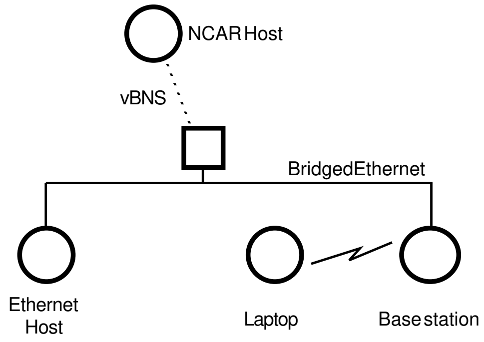

Figure 4.2: Hosts used in experiments

4.5 Pattern-based evaluation of local retransmission

In this section, we will compare the performance of TCP with and without local error control. To investigate interactions between local and end-to-end error control, we used a very simple local error control strategy, persistent local retransmission, and employed simple synthetic packet loss patterns. Because interactions between local and end-to-end error control are likely to vary as a function of the difference between the local and end-to-end round-trip latencies, experiments were run in both a LAN and WAN environment (see Figure 4.2). The LAN experiments used an Ethernet segment in addition to the wireless link. For the WAN experiments, we used a remote host at NCAR, 6 hops away from our wireless LAN. The round-trip time between CMU and NCAR is approximately 40 milliseconds; the path is lightly loaded and essentially lossless. It should be noted that this latency, while substantial, does not represent the worst available on the Internet: intercontinental latencies can easily reach 120 milliseconds.

In this section, we will demonstrate the well-known severe degradation of TCP performance in the face of modest packet loss, show how local retransmission avoids this unfortunate situation in a variety of situations, and present a more general analysis of the effectiveness of local retransmission.

4.5.1 Basic robustness evaluation

We first focus on the simplest possible scenario: a single TCP connection between a wireless slave and the master station. Figure 4.3 shows the throughput of TCP without local retransmission. As for all the results in this section, each data point summarizes the results of 5 1-megabyte runs; the error bar denotes the range of observations and the point on the bar denotes the mean. The error pattern for Figure 4.3 consists of a single burst of one to eight packets being dropped out of every hundred packets, resulting in error rates between 1% and 8%. We see that performance degrades quite quickly. TCP handles single packet drops well, but, as we would expect, when multiple packets per window are dropped, TCP congestion avoidance and timeouts are triggered [60, 47, 80] and performance drops by a factor of two. It continues to drop quickly as the burst size increases. Conventional wisdom, supported by

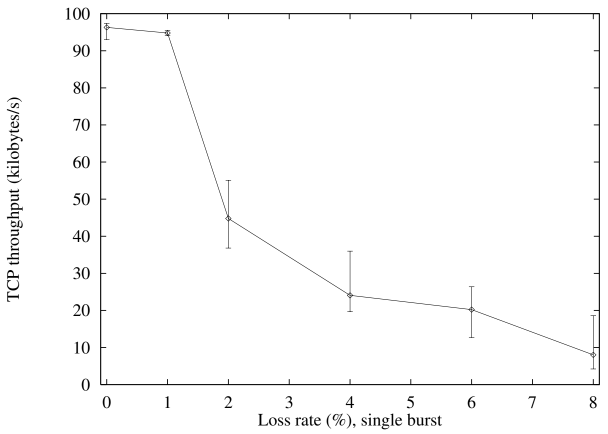

Figure 4.3: TCP versus data-packet loss in bursts of one to eight packets per hundred.

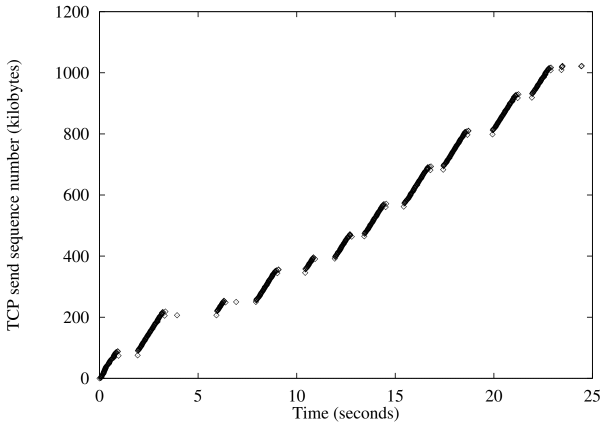

Figure 4.4: TCP encountering 2% packet loss in 2-packet bursts.

recent work [81], says that TCP Reno can handle packet loss rates of up to 1-2%; our results support this conclusion, given that burst losses are particularly challenging. Figure 4.4 shows a TCPDUMP trace of a representative observation from the 2% loss (2-packet burst) case of Figure 4.3. We see that timeouts substantially degrade performance.

Figure 4.5 shows the throughput of TCP when we enable local retransmission. Five-packet bursts were dropped from windows of 6, 12, 25, 50, and 100 packets to yield a variety of loss rates. Note that scale is very different from the scale in Figure 4.3: it covers loss rates of 0-85% instead of 0-8%. We see that, as the link capacity degrades linearly, the achieved throughput drops off in the same fashion, which is essentially ideal. Figure 4.6 shows a TCP trace from the 83% loss case. The reason that this works well is straightforward. The local retransmission hides most of the packet loss on the wireless link from TCP, so TCP stabilizes at a rate corresponding to the average throughput of the wireless link.

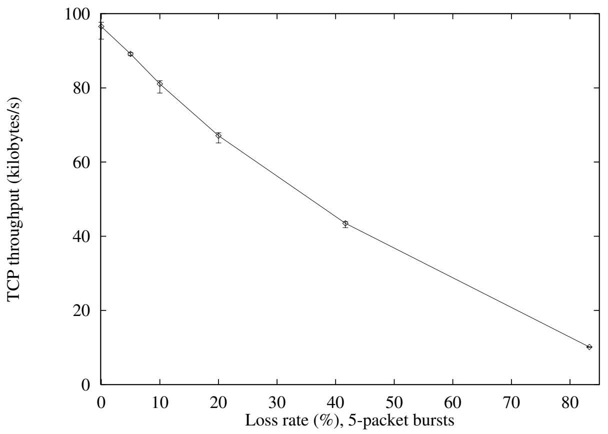

Figure 4.5: TCP with link-level retransmission, 0% to 83% loss.

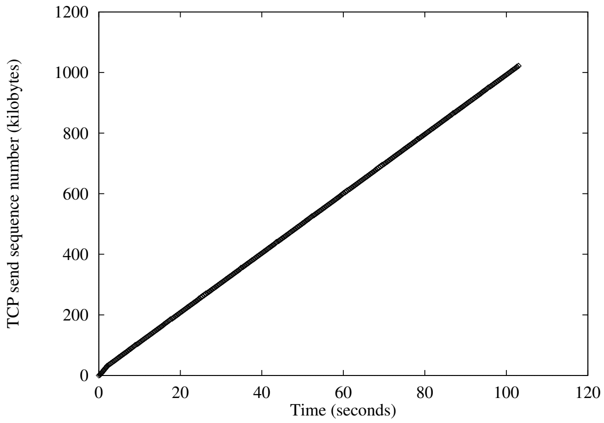

Figure 4.6: Trace of TCP with link-level retransmission, 83% loss.

4.5.2 A broad set of scenarios

Above we evaluated local retransmission in a very restricted scenario with the intent of understanding its behavior. In practice a wide range of conditions may impact the effectiveness of local retransmission. In this section we evaluate its performance under a diverse set of conditions.

So far we have presented results where only a single link is involved. A slightly more complex topology involves a single Ethernet segment and our (bottleneck) wireless link. Figure 4.7 shows the throughput under a variety of loss rates from a 100 MHz Pentium system on the same Ethernet segment as the base station. The presence of the additional link makes no noticeable difference in throughput; reversing the direction of flow (not shown) results in the same performance.

Figure 4.8 extends our investigation to a WAN connection from our campus across the vBNS to a Cray at ucar.edu. The network path comprises our base station, our departmental gateway, two intermediate nodes at our service provider, a vBNS hop, the UCAR gateway,

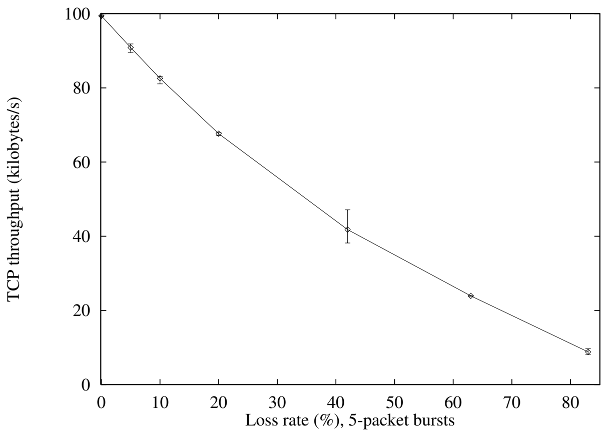

Figure 4.7: Effect of packet loss on TCP, Ethernet transmitter.

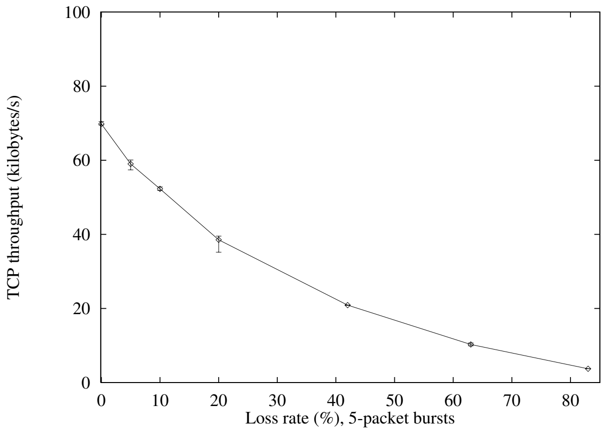

Figure 4.8: Effect of packet loss on TCP, distant transmitter.

and the Cray. The round-trip time is approximately 40 milliseconds. Again, we observe that local retransmission results in consistently good performance; reversing the flow direction (not shown) yields no significant difference. The reduction in best-case rate from 100 to 70 kilobytes per second is due to the combination of TCP's smaller long-haul maximum segment size and the high per-packet cost of our software MAC implementation (Section 4.3.3).

Figure 4.9 depicts the results of running four independent TCP streams over our local retransmission protocol. Each point represents the throughput of a single TCP stream during a single run; five runs of four streams each time result in twenty observations for each loss rate. Note that the distance between the best and worst rates observed for each loss rate is very small, even though they may be from different runs.

4.5.3 Analysis

In this section we analyze why local retransmission is so effective. This allows us to identify the conditions that must be met by local error control mechanisms for efficient interaction

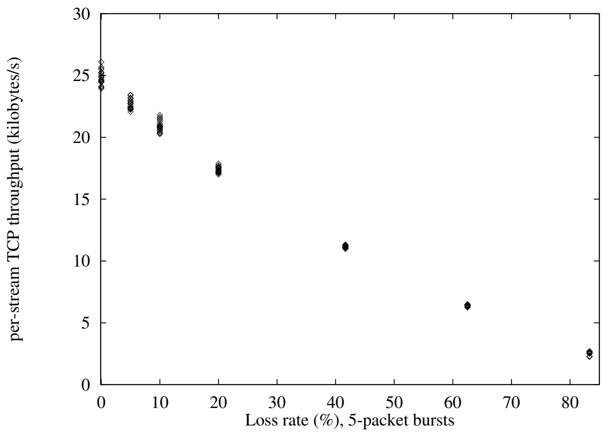

Figure 4.9: User throughput of four competing TCP streams.

with TCP.

Steady state conditions

We will begin by considering the simple case, a link with a constant error rate. With local retransmission, the impact of the wireless link on traffic will be similar to that of a regular wired link. The relative slowness of many wireless links means that buffer memory sufficient to smooth out long burst losses is affordable. Assuming the wireless link is the bottleneck, TCP congestion control will converge on the link's available throughput.

When we consider the steady state behavior of TCP, local error control must meet several requirements for TCP to exhibit stable behavior under steady error conditions:

- TCP interprets packet loss as a sign of congestion (Figure 4.3). This means that local error control should be persistent enough that lost packets almost always indicate congestion.

- Packet reordering results in the receiver generating duplicate acknowledgements, which will cause the sender to infer packet loss or even congestion. Local error control mechanisms should avoid packet reordering since it will cause unnecessary transport-level retransmissions, and thus waste bandwidth. An alternative to avoiding packet reordering is to avoid its negative effects by, for example, suppressing duplicate acknowledgements [11], but this requires special-case router code for each supported transport protocol, a weakening of IP semantics.

- TCP estimates the round-trip time and uses this estimate to determine when it should back off and retransmit data. If local error control can significantly delay packets, round-trip times may become highly variable. This could cause unnecessary timeouts and retransmissions or, alternatively, excessive backoffs that would unnecessarily delay later retransmissions. The round-trip variability problem could be exacerbated by long delays, suggesting that local retransmission may not be effective on satellite links.

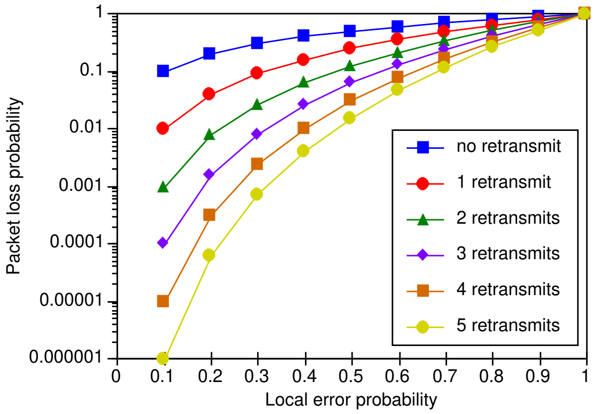

Figure 4.10: End-to-end packet loss rate as a function of local loss rate and maximum retransmit count

Dynamic error environment

Aside from steady state conditions, we also must consider the case that error conditions improve or degrade. If error conditions improve, the usable capacity of the link will improve and periodic probes by TCP senders will discover and start using the excess bandwidth. If error conditions degrade, local error control will need to retransmit harder to transfer packets across the wireless link. The result is that the queue will drain more slowly and will eventually overflow, causing packets to be lost. This will cause TCP to back off and retransmit the lost packets. Since we have a congestion condition, this is exactly the right response.

Persistence of local error control

Local error control is not an all-or-nothing arrangement, but error control strategies with different degrees of effectiveness and persistence can be implemented. This raises the question of how effective local error control must be to satisfy TCP.

Figure 4.10 shows how the end-to-end packet drop rate changes as a function of the local packet loss rate and the maximum retransmit count, assuming steady state conditions and random errors (of course, the residual error after a fixed number of retransmissions will be higher if errors are bursty). Not surprisingly, the results show that, for high error environments, retransmission may need to be very persistent if we want to keep the packet drop rate below 1%. As we discussed in Section 4.5.1, TCP performance degrades quickly if the end-to-end packet loss rate is more than a few percent, so limiting the number of retransmissions to a small constant will work well only if wireless packet loss rates are low.

A potential drawback of persistent local retransmission is that it may delay packets significantly. This may result in interference with TCP retransmission [36, 11]. To better understand what happens in such conditions, we created an error environment that should cause competing retransmissions. We chose a pattern that alternates between transferring

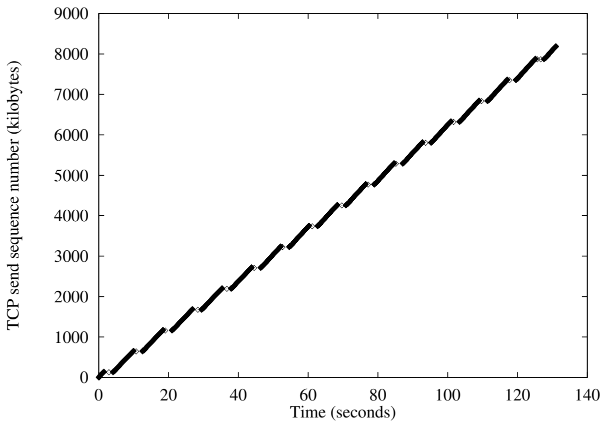

Figure 4.11: Trace of competing-retransmission scenario: 100-packet burst every 500 packet times.

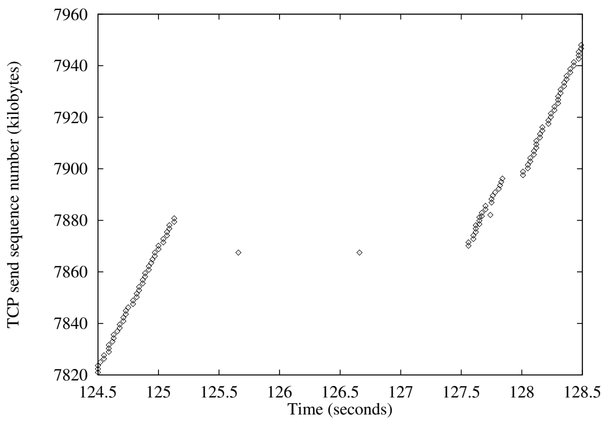

Figure 4.12: Zoomed-in trace of competing-retransmission scenario.

400 consecutive packets without errors, which will lull TCP into believing that the link has high throughput and low latency, and dropping 100 consecutive packets, which will cause a sudden significant delay and should trigger a TCP timeout.

Figure 4.11 shows the packet trace for a single-hop TCP flow encountering this error pattern. The test program reported a throughput of 512 Kbit/s, about 80% of the link capacity available after packet loss (640 Kbit/s). The trace shows that between the error bursts TCP performs well. Figure 4.12 shows in more detail what happens during an error burst. We observe two types of redundant packet: "probe" packets used by the sender to reestablish contact, and a sequence of packets retransmitted after the pause, which TCP incorrectly believes were lost. While we do have redundant retransmissions, the amount of overlap is only about 3%. This overlap can be further reduced by the Eifel extension to TCP [54] (see Section 4.7.2).

For local error control to perform well, end-to-end retransmission timeouts should be substantially longer than the single-hop round-trip time. Many TCP implementations are derived from the Berkeley "Net-2" implementation, in which retransmissions are scheduled by an operating system task which runs periodically, often every 500 milliseconds (the "slow timeout" interval). This creates a floor on the end-to-end timeout that gives link-level error control substantial time to hide error bursts. While TCP implementors might choose a finer grain or impose a less conservative minimum timeout (typically two slow-timeout "ticks"), this relatively coarse timeout is supported by RFC 793 [102], which suggests a minimum value of one second for the retransmission timeout. Furthermore, TCP trace data suggests that long-haul Internet connections may avoid many false timeouts by maintaining a minimum retransmission timeout of at least one second [2]. As a result, long timeouts should be considered more as protocol feature than implementation artifact.

It is also interesting to note that, regardless of the resolution of the timers, the values they take on allow for some variation. For example, systems compliant with RFC 2988 [97] set the retransmission timer to an estimate of the round-trip time plus four times the mean deviation from that estimate, as suggested by Jacobson and Karels [60]. This means that, if a variable number of wireless link-level retransmissions caused fluctuations in the end-to-end round-trip time, TCP would tend to react by retransmitting more cautiously. Two simulation studies [83, 31] have observed this positive interaction.

In summary, excessive retransmission persistence, such as the perpetual retransmission implemented in our simple LLC, poses a risk of reduced performance due to inter-layer duplicate retransmissions. However, factors such as TCP's adaptation to delay variation, cautious minimum timeout, and slow-start probing seem to allow persistent retransmission with only a small efficiency loss. Meanwhile, giving up after only a few transmissions has the potential to seriously compromise TCP performance. We conclude that a high degree of link-layer retransmission persistence is well worth the minor negative side effects.

Similar results were obtained by Reiner Ludwig [79, 78]. He observes that persistent retransmission is critical to TCP performance and that competing retransmissions are rare and innocuous. He proposes and investigates a system that allows end-system control over the degree of link-level retransmission persistence.

4.5.4 Summary of local retransmission experiments

We believe these experiments demonstrate the potential for a "pure link-layer" retransmission strategy to use local knowledge to significantly and transparently enhance the throughput of unmodified TCP in many difficult situations. The main limitation of this approach seems to be that it cannot hide pathologically long delay bursts from TCP, which may respond with a small amount of duplicate retransmission. On the other hand, when the delay period is over, the order-maintaining queue structure allows TCP to recover smoothly. Note that, while in our experiments the wireless link was the first or last segment in the path, the analysis suggests this is not critical: what matters is that the conditions listed in Section 4.5.3 are met.

While we have encouraging results for TCP, which is by far the dominant reliable data transport protocol, it seems clear that other applications, such as streaming video, could require different low-level retransmission policies. Luckily, the Internet seems to be evolving in the direction of making flow-type information available to link-layer elements via Differentiated Services marking [91] or RSVP-like protocols [26]. Different low-level error control policies could be implemented for packets belonging to flows of different types. For delay-sensitive traffic such as interactive video or audio transfers, local retransmission could be less persistent since packets "expire" after a certain time interval.

4.6 Trace-based evaluation of adaptive error control

The error environments we have investigated so far are unrealistic in two ways. First, the packet loss patterns we used were synthetic and were not representative of any known natural error patterns. Second, packet-loss error traces poorly describe actual error environments, in which "packet loss" is due to a variety of underlying error events, detected by different hardware or software layers. Packets could be "lost" due to capture effects, initial synchronization failures, errors in recipient MAC addresses, synchronization loss due to clock drift, checksum failures due to bit errors, etc. The techniques that defend against each threat differ, and, since each one consumes air time and/or energy, it is counterproductive to try to solve "packet loss" without some indication of which underlying process is responsible.

Therefore, in this section we will reprise the detailed trace-replay experiments of Chapter 3, replacing the abstract throughput simulations used there with the throughput obtained by actual kernel implementations of TCP and UDP. After describing our adaptive error control implementation, we will present the performance of various error control policy modules in the presence of different error environments. We will investigate whether the rapidly-varying throughput made available by an adaptive link layer is readily usable by TCP, evaluate the utility of adaptive error control policies and the error control techniques they use, and compare the results of the previous chapter's abstract throughput simulator with the real-world TCP implementation measured herein.

4.6.1 LLC infrastructure

In Section 3.2 we observed that both packet length adjustment ("packet shrinking") and error coding can significantly reduce the packet error rates caused by the error environments recorded in our trace data. When we wish to move beyond simple retransmission to employ these techniques, several practical issues arise. First, since each of these techniques causes the size of each transmisison's user data payload to vary, we need some way of transmitting end-to-end IP packets inside these small and variable payloads. Second, we need to collect data about the current error environment so that adaptive policy code can control the mechanisms we provide.

Three options for handling the variable-length payload problem are varying the IP MTU, relying on IP datagram fragmentation, or providing local fragmentation and reassembly.

While it is possible to do end-to-end management of packet shrinking by adjusting the IP maximum transmission unit (MTU) of the wireless link and propagating this information to end systems, or by employing IP fragmentation, both approaches have substantial per-packet overheads for even medium-sized packets. For example, if we reduce our MTU from a typical 1000 bytes to 200 bytes, TCP is required to transmit the IP and TCP headers (typically a total of 40 bytes) six times instead of once for a large packet, raising the header overhead from roughly 4% to roughly 20%. While IP fragmentation would replicate only the IP header, resulting in less throughput inefficiency, it would introduce other system overheads. Generating each fragment would require IP header manipulations, including a checksum recomputation. In addition, IP fragmentation results in non-trivial work for the receiving end-system, such as maintaining and garbage-collecting a potentially arbitrary number of reassembly queues.

Propagating link MTU values across the network is a slow and costly process, whether this is done via routing protocols or per-flow MTU discovery, so it would be impractical to adjust the value to prevailing error conditions.

Finally, both TCP segmentation and IP fragmentation would limit adaptation by breaking a packet once into fixed-sized sub-packets, thus committing the link to a fragmentation policy on a coarse grain.

For these reasons, we implement packet shrinking through link-level packet segmentation and reassembly over the wireless link.

This requires segmentation and reassembly buffers on each host and some changes to the packet headers previously described in Section 4.3.1. Every DATA transmission adds to the existing packet sequence number a starting byte offset, a byte count, and a "packet complete" bit, while acknowledgements consist of not only a packet sequence number but also a cumulative length indicating correctly received bytes. A sender can signal that it has abandoned a packet transmission partway through by incrementing its sequence number before its peer has acknowledged the previous packet; at this point, the recipient knows to flush its reassembly buffer. This premature abandonment was used to disable link-level retransmissions for the experiment in Section 4.5.1. The resulting LLC protocol, summarized in Table 4.2, can execute data transactions for variable-sized segments instead of complete IP packets.

The other job of our LLC protocol is carrying adaptation strategy information between the cell master and slave stations. If a master's POLL reaches a slave, the slave's response will describe any observed packet truncation and/or decoder failures. If the slave's response reaches the master, the master will likewise note damage. Adaptation policy modules (described in Section 3.3.2) use these observations to estimate the appropriate transmission size and error coding level for the next transmission. This model assumes a link capable of almost always transmitting a small amount of metadata even if the main body of a packet is severely damaged. In practice, many wireless links transmit certain information, such as synchronization or training sequences and station addresses using redundancy or robust coding, making this assumption realistic.

4.6.2 Performance of TCP and UDP

To evaluate the performance of an error control policy operating in a particular error environment, we measured the throughput across the link of a single TCP stream. In addition,

Command	Parameter	Meaning
POLL	Slave	specifies slave station
	Channel	channel being acknowledged and polled
	Slave-seqno	last packet received on this channel
	Slave-bytecount	last byte received in that packet
DATA	Channel	channel number
	Seqno	number of packet in progress
	Offset	first byte contained in this segment
	Length	count of bytes in this segment
	Done	segment contains final byte (boolean)
ACK	Channel	channel number
	Master-seqno	last packet received on this channel
	Master-bytecount	last byte received in that packet

Table 4.2: LLC packet summary. In the general case, the master transmits POLL and DATA together, to which the slave will respond with DATA and ACK, also in a single transmission.

we measured the throughput obtained by a program that sprays UDP packets as fast as possible, ignoring send queue overflows and losses, reporting only the number of packets received. Since this program counts packets *received*, it estimates the link throughput after error control.

Table 4.3 reports the results achieved by the adaptation policies from Section 3.3.2 when confronted with the error traces presented in Section 3.1. The relative throughput achieved by TCP and UDP for each combination of trace and policy is displayed in Figure 4.13. This clearly indicates that, regardless of adaptation policy, the resulting link was as usable by TCP as by loss-insensitive UDP: our adaptive error control system is not causing TCP to lose performance due to false triggering of its congestion avoidance mechanism.

Trace	Bold		Light		Robust		Bimodal		Bi-code		Bi-size		Flex	
	TCP	UDP	TCP	UDP	TCP	UDP	TCP	UDP	TCP	UDP	TCP	UDP	TCP	UDP
Office	90.1	91.0	84.8	85.4	32.0	32.2	89.9	90.3	89.6	90.2	89.6	90.6	89.7	89.9
Walking	84.5	85.2	81.3	82.4	32.0	31.6	84.6	84.9	84.2	85.1	84.2	84.7	83.2	83.8
Adjacent	15.0	15.9	14.8	15.4	5.4	5.3	15.1	15.0	14.1	15.0	14.6	15.1	14.7	15.2
Table	0	0	5.4	5.4	19.3	19.2	12.8	12.6	37.9	37.9	0	0	45.8	46.2
Walls	64.6	65.9	75.8	77.2	30.2	29.8	64.1	64.9	64.7	65.7	64.9	65.6	72.6	72.9

Table 4.3: Performance of different error control policies: averages, in kilobytes/second, of five runs.

4.6.3 Utility of adaptive error control

Examining Table 4.3 more closely allows us to determine the utility of error coding and packet length adjustment and the importance of employing each adaptively. Figure 4.14

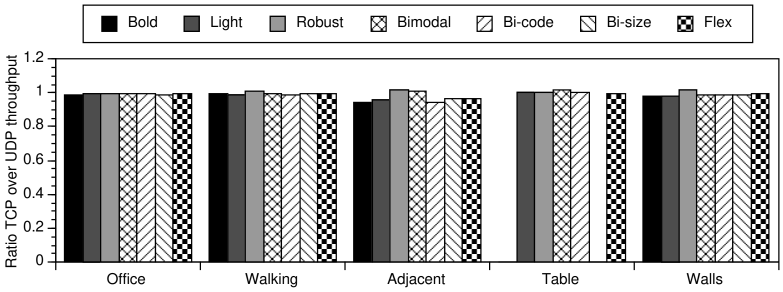

Figure 4.13: Comparison of TCP and UDP throughputs for the experimental scenarios. The similarity of these throughput figures indicates that TCP is not falsely assuming link congestion.

presents the TCP numbers graphically. For each error trace, the throughputs obtained by the various adaptation policies are normalized to the throughput of the top performer. The data are then transposed so that the figure displays, for each policy, how competitive it was across error traces. This makes it clear that, for example, in every error environment ROBUST achieves approximately half the throughput of whichever other policy was the best for that environment.

The performance of the static policies is fairly predictable. BOLD (retransmission, or pure ARQ) does very well in low-error situations because it doesn't spend any bandwidth on shrinking or coding overhead. Unfortunately, it is unable to transfer any data in the "table" case, which has a packet corruption rate of 94%, since TCP cannot complete its inital three-way handshake before timing out. LIGHT fares slightly worse in low-error situations but manages to achieve some minimal throughput in the harsh "table" case. It dominates the other policies in the "walls" scenario, in which 27% of the packets are corrupted, but only lightly. ROBUST does poorly in essentially every case: in addition to 30% overhead lost to coding, it loses substantial throughput due to the high cost of sending small packets.

The adaptive policies all perform generally well, with the exception of BI-SIZE, which does not perform coding–like BOLD, it cannot operate in the "table" case. Comparing the three two-mode policies allows some evaluation of the utility of packet shrinking. If we examine the performance of BIMODAL and BI-CODE in the "table" case, which has 94% corruption but only 2% truncation, we can verify that linking packet shrinking to error coding level is probably counterproductive, since at least in this case decoder failures are a poor predictor of truncation and the performance reduction is substantial. The case for shrinking is better made by comparing the two policies in the "adjacent" case, which has almost no corruption and 23% truncation. In this situation packet shrinking provides a modest performance increase. Finally, the FLEX policy, which can independently adjust shrinking and coding, and can do so with finer granularity, does well overall and particularly well in the challenging "table" and "walls" cases.

In summary, BOLD demonstrates that error coding can be either useful or necessary depending on error conditions ("walls" or "table"). The overall performance of the static policies compared to the dynamic policies argues for adaptation. Finally, the strong perfor-

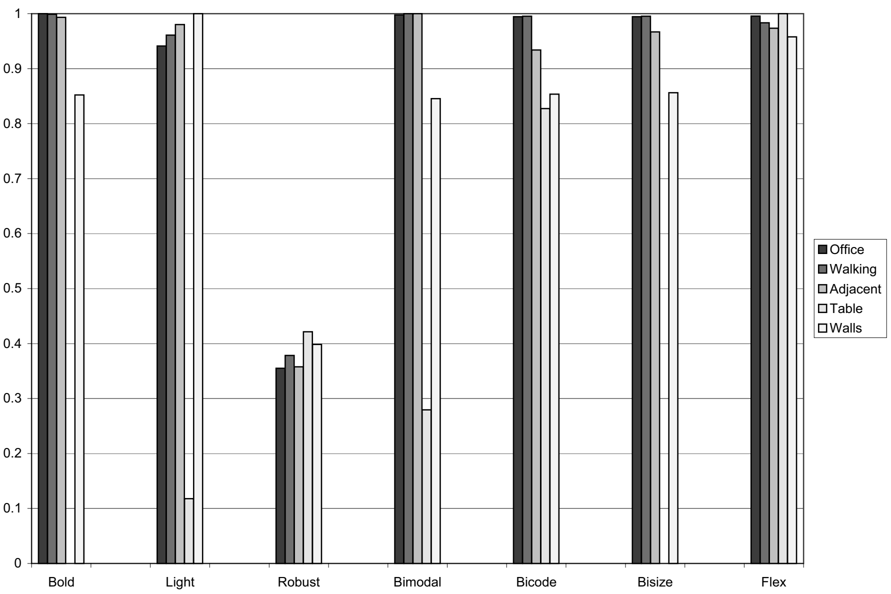

Figure 4.14: Comparison of normalized performance for each policy. For each policy, each bar represents that policy's performance in one scenario (office, walking, adjacent, table, or walls) normalized to the best performance in that scenario.

mance of FLEX regardless of error environment suggests that a simple adaptive policy can be very effective.

4.6.4 Comparison to simulator results

Comparing the TCP throughput results in Figure 4.14 to the throughput simulator numbers in Figure 3.9 produces some confidence in both the simulator and the result of trace replay in the running system.

The first thing to observe is that the results are similar—both measurement environments find the same traces to be "hard" across policies and each policy finds the same traces to be hard or easy regardless of the measurement platform.

Some of the differences between the figures are accounted for by two factors. First, the simulator can eke out throughput in situations where real TCP cannot. This is clear if we compare the performance of the BI-SIZE policy in the "table" error environment—TCP cannot establish a connection even though there is *some* throughput available. Second, the "robust" policy, which achieves throughputs as high as 70% of the oracle in the simulator, never achieves much better than 40% of the best competing policy in the running system. We attribute this to the high per-packet cost of our software implementation (see Section 4.3.3).

4.6.5 Summary

The above results suggest that error coding and packet length adjustment can build on top of local retransmission to further increase throughput in interference and attenuation environments. Adaptation appears to be vital to achieve good performance in the variety of error environments that a single mobile host will plausibly encounter. Figure 4.13 indicates that, regardless of adaptation policy, the resulting link was as usable by TCP as by loss-insensitive UDP: our adaptive error control system is not causing TCP to lose performance due to false triggering of its congestion avoidance mechanism.

4.7 Related work

4.7.1 TCP in Wired-cum-wireless Environments

An excellent survey paper, titled *TCP in Wired-cum-wireless Environments* [98], presents a clear overview of the problems that wireless environments pose for TCP and discusses a wide variety of research related to running TCP over wireless links. The paper includes background information on various wireless network technologies and a concise description of the WAP platform (widely known as "wireless web" in the cellular phone world).

4.7.2 Reiner Ludwig's Cross-Layer study

.

A higher-level overview of the relationship between that dissertation [78] and this one appears in Section 1.5.4. The author proposes modifications to the typical configuration of the RLP radio link protocol and modifications to TCP's congestion control and retransmission behavior to better handle data link errors and longer handoff-induced outages. We will consider the link-level and TCP issues in turn.

Ludwig found that the typical configuration of the GSM RLP link protocol was suboptimal in three ways. First, the degree of forward error coding, which is fixed rather than variable, was frequently stronger than necessary. Second, the frame size chosen by the RLP designers is too small for many error environments, resulting in noticeable throughput reduction. Third, link-level retransmission was not sufficiently persistent to support TCP. These results are all in agreement with our observations: error coding and frame length should both be dynamically adjusted, and TCP requires link-level retransmission with substantial persistence. In addition, we both observed that inter-layer retransmission effects were typically not significant.

Our investigation of link-level error control was not constrained by a pre-existing link protocol, so we had more freedom to experiment with error control mechanisms such as variable rate error coding and packet length adjustment and to investigate multiple adaptation policies. On the other hand, Ludwig investigated mechanisms for varying the reliability of a flow's traffic according to "flow type" information which can be provided by communicating end-systems.

Ludwig then proceeded to describe and analyze problems related to TCP's handling of retransmissions and timeouts. The relatively low throughput of the GSM radio channel means that round-trip delays due to link-level retransmissions are frequently visible on the timescale used by TCP's round-trip estimator. When TCP's estimate of the round-trip time is substantially incorrect, the result may be spurious retransmissions, congestion avoidance behavior, and timeouts. Ludwig observed that much of this trouble is due to the TCP transmitter's inability to distinguish between acknowledgements of different end-to-end transmissions of the same packet. He resolves the acknowledgement ambiguity by combining use of the existing TCP timestamp option with modest code changes to TCP transmitter behavior. While acknowledgement ambiguity in an end-to-end protocol merits a remedy, this represents an argument for good design of protocol layers rather than for re-working the Internet architecture (as suggested by others, see below).

A recent simulation study of this "Eifel Algorithm" for resolving transmission ambiguity [54] observes substantial improvements in both throughput and download latency when Eifel is applied to various common TCP implementations running over GSM's General Packet Radio Service (GPRS [30]).

4.7.3 Other Inter-layer Studies

An early simulation-based study of interactions between link-level retransmission and a TCP-like protocol [36] suggested that, unless link errors are very common, link-layer retransmissions would frequently duplicate end-to-end retransmissions. Later studies involving real TCP have, to a large extent, not confirmed this concern.

A paper on Channel State Dependent Packet Scheduling, (which is discussed further in Section 5.8.1) observes the importance of link-level retransmission for preserving TCP throughput in the face of burst errors; coarse-grained TCP timeouts allowing multiple link-level retransmissions; and the desirability of link-level error control being independent of end-to-end transport protocols. Evaluation is via simulation with a Markov model of burst packet loss.

An ns2 simulation study of TCP running over GPRS [83] observes TCP obtaining reasonable performance over GPRS RLC, even in the face of harsh link-level conditions. In particular, link-level retransmission is fast enough to operate beneath TCP, and TCP's retransmission timeout is observed to back off in response to round-trip variance induced by the low-level retransmissions.

An analytical study of TCP running over the link layer retransmission provided by the new UMTS cellular telephone standard is analyzed in [31]. The study considers the effects of fading (i.e., not interference), uses a Markov-model error environment, and does not consider the effects of error coding. The primary results are that TCP performance is dramatically improved by the presence of link-level retransmission and that “silent ARQ” (absence of inefficient inter-layer retransmissions) is typical.

4.7.4 TULIP

TULIP (“transport unaware link improvement protocol“) [96] is a mostly-protocol-blind link-layer retransmission scheme. While it does not attempt to locally compute end-to-end TCP state or modify the TCP data stream, it differentiates between TCP packets bearing data, which are transmitted reliably and in-order, and packets bearing only acknowledgement information, which are not. UDP traffic is not transmitted reliably. When possible, TULIP piggybacks TCP acknowledgements on TULIP acknowledgements which are in turn piggybacked on MAC-level acknowledgements. The protocol uses a sliding window and selective acknowledgements; timeouts are based on maximum link-level round trip time. Simulation results show that TULIP allows TCP to perform well across a wide range of error rates and distributions. In addition, TULIP outperforms the “Snoop” TCP accelerator (see below) when error rates are high, largely because Snoop’s retransmission approach is similar to that of TCP, which does not perform well in the face of high error rates.

The authors observe that variance in end-to-end round-trip time induced by link-level retransmissions causes TCP to compute conservative end-to-end timeout values, as we suggested in Section 4.5.3.

The TULIP results provide a third demonstration, after this research [39] and that of Ludwig [79], that link-level retransmission is vital and effective for improving TCP performance. Competing inter-layer retransmissions are not a relevant factor across a wide range of error rates.

The published results do not clearly specify why TULIP inspects TCP packets to specifically avoid reliable transmission of TCP acknowledgements. That is, no comparison of TULIP running with and without reliable TCP acknowledgements is presented. Furthermore, it is not clear that all UDP traffic should be unreliable. Certain important applications, such as the Domain Name System [84, 85], rely on UDP and perform poorly when multiple packets are lost.

4.7.5 Historical Wireless LAN retransmission efforts

Several high-speed wireless LANs include both link-level retransmission and packet fragmentation. The IBM Wireless RF LAN [14] performs fixed-size fragmentation primarily to facilitate time scheduling and comply with frequency-hopping dwell time limits.

Though the MACA [63] and MACAW [20] protocols are designed for distributed multiple access rather than the centralized control our MAC employs, we believe that their approach to local retransmission is similar in spirit to ours. The MACAW protocol design considers multiple flows per device and presents evidence that a post-packet acknowledgement increases TCP throughput.

The Arlan wireless LAN includes medium-level retransmission; a comparative study of WaveLAN and Arlan [114] observes noticeable TCP performance improvements as a result.

4.7.6 IEEE 802.11 wireless LAN

The IEEE 802.11 wireless LAN standard [70] may optionally be configured to perform fragmentation into fixed-sized chunks, and retransmits packets a (configurable) fixed number of times according to a length threshold. Our experience suggests that the relatively small default configuration constants may leave enough residual errors to disturb TCP and that adaptive packet coding and shrinking merit consideration.

Integrating adaptive local error control with the 802.11 contention-based Distributed Coordination Function (DCF), a descendant of MACA and MACAW, faces no obvious major obstacles. One challenge is that, unlike our prototype MAC and the 802.11 Point Coordination Function (PCF), DCF round-trip times may be very large in periods of heavy contention. This poses two challenges: the control loop for the adaptive link protocol may be too long, and long link-level delays may cause competing inter-layer retransmissions. The control loop problem may be less difficult than it seems. Our approach is based on the assumption that, once a packet is transmitted, the sender can quickly determine whether it must be retransmitted and whether any coding or sizing adjustments are necessary. While DCF round-trip times may be long, they are nonetheless bimodal: if a transmission is successful, it will take place, and be acknowledged, within the strict bounds of the RTS-CTS-DATA-ACK cycle. Truncation and decoder failure information could be added to the medium-level ACK. Otherwise, some part of the cycle failed, which is analogous to a single packet loss in the polled/PCF case, and the entire cycle will need to be repeated. The inter-layer competition problem is straightforwardly parallel: if the round-trip time is always large, the end-to-end transport protocol will back off to match; if it is typically small but occasionally becomes very large, there may be an occasional competing retransmission, but this will probably be unimportant.

The IEEE 802.11e draft standard [59] includes optional per-flow Reed-Solomon block codes. The potential of this technique may be limited because the coding level employed is static. We have observed that it is difficult for static error coding decisions to perform well across a variety of error environments.

4.7.7 Other Link-Level Retransmission Systems

All three "second-generation" or "2G" cellular phone systems, the IS-54/IS-136 TDMA system [87] the IS-95 Qualcomm CDMA system [64], and the European GSM standard [44], provide link-level retransmissions for packet data. The AIRMAIL design [7] adds bit-, byte-, and packet-level adaptive forward error correction to yield a protocol for power- and processing-constrained access devices.

A WaveLAN-based evaluation of packet shrinking [72] also suggests an architecture including link-level fragmentation and adaptive error coding.

4.7.8 Wireless-aware TCP

Researchers have proposed various modifications to TCP to improve performance over wireless links. These suggestions face two general obstacles. First, it is difficult for end-system hosts to determine which negative packet outcomes are due to which link along a network path. If multiple links exhibit different error modes, this problem is very challenging. Second, deployment of modifications to TCP is traditionally very slow. Our position is that it is unwise to deploy network links that *typically* re-order packets or frequently lose them in the absence of congestion.

One approach is to modify the underlying network, or at least the wireless links, to explicitly signal packet errors to TCP [52, 68]. This frees TCP from its assumption that losses signal congestion and allows it to retransmit aggressively. Aside from various technical issues related to the mechanisms and reliability of this signalling, there is a significant deployment hurdle, in that every wireless link must generate the notifications and only wired TCPs that understand them can benefit.

Wang and Kung [132] suggest an approach by which end-system hosts can distinguish network congestion from packet loss by comparing the fates of two TCP streams, one carrying bulk data and another carrying only token amounts.

The authors of [129] propose modifying TCP so that it can run more efficiently above a protocol-blind link-layer retransmission scheme that does not preserve packet ordering. Unmodified TCP receivers generate duplicate acknowledgements in response to mis-ordering, which the TCP senders interpret as an indication of loss and congestion; the modified TCP receiver artificially delays duplicate acknowledgements for a period of time in the hope that the link layer can provide the missing packet. Simulation results indicate that this change improves performance in the target environment but can degrade throughput noticeably in the presence of genuine congestion-induced losses.

TCP modifications to allow senders to un-do congestion avoidance actions induced by packet reordering are discussed in [23]. This work depends on TCP using a mechanism such as Eifel [78] or DSACK [48] to detect spurious retransmissions.

Mobile TCP [123] attempts to improve TCP performance during handoffs. When the mobile host senses a handoff, it notifies its peer, which temporarily attributes packet losses to the handoff rather than to congestion. In addition, when a mobile host switches from one network provider to another, the wired peer resets its network estimation state (round-trip time, congestion window, etc.) to the same default values used to establish new connections. These modifications result in noticeable performance improvements.

Freeze-TCP [53] proposes modifying the TCP receiver code to warn the remote TCP sender of an impending link disconnection by closing the TCP receive window. Freeze-TCP will later advise the sender of a reconnection by transmitting duplicate acknowledgements. This allows the sender to freeze its connection state during a link outage or handoff so that it doesn't incorrectly infer congestion from the timeouts that it will observe. Assuming advance notice of a link outage is available, this approach provides noticeable performance improvements. The paper also presents an excellent overview of issues related to TCP gateways and accelerators (discussed below).

Finally, WTCP [122] proposes replacing TCP outright with a rate-based protocol in wide-area wireless networks.

4.7.9 TCP accelerators

A comparison [11, 12] among the Berkeley LL family of link-layer retransmission protocols demonstrated a noticeable performance enhancement obtained by retransmitting and filtering TCP packets and acknowledgements. This implementation is specific to TCP and the general approach depends on transport protocols generating frequent acknowledgements (as opposed to, e.g., periodic summaries). Our situation differs from theirs in that our timeout interval is smaller by roughly an order of magnitude and our retransmission scheme does not re-order packets. This allows us to achieve good performance without depending on TCP-specific protocol information. Such a dependence will require TCP accelerators to be upgraded as new TCP options become popular.

Protocol independence allows us to provide good service for non-TCP applications. One example is UDP-based request/response protocols such as Domain Name System (DNS) lookups, where throughput needs are minimal but human intolerance of latency creates a dependence on low end-to-end packet error rates.

4.7.10 TCP gateways

Several groups have examined "indirect" or "split" approaches to improving TCP performance [8, 135, 9, 29, 10, 28], which rely on transparent protocol translators. For example, in an indirect implementation of a transport protocol, a fixed host communicates with a mobile host via a proxy that transparently terminates the transport-level connection and uses another transport-level connection (and potentially another protocol) to reach the mobile host.

Both TCP accelerators and TCP gateways fail to address non-TCP traffic and depend on an ability to understand TCP packets.

However, TCP gateways face additional challenges. It is tricky to maintain the end-to-end semantics of acknowledgements unless the gateway delays issuing its own acknowledgements until hearing from the mobile host; being conservative in this fashion can reduce throughput. When a mobile hosts moves from one TCP gateway coverage area to another, the two gateways must migrate connection state information to avoid breaking the end-to-end connection.

In light of these complexities, it is fortunate that our experience suggests TCP, and other transport protocols, may benefit from simpler and protocol-blind link-level approach.

4.8 Conclusion

The results presented in this chapter indicate that "pure" link-layer local error control mechanisms can greatly increase the efficiency of data transfer in wireless LANs. When designed correctly, these mechanisms can mask the worst effects of the high and dynamic error rates often found in wireless networks. Our experiments show that non-snooping local retransmission can deliver vast improvements in TCP performance over lossy wireless LAN links, allowing TCP to use a high fraction of the available link bandwidth. An analysis of our experimental results shows that, to effectively support data transfers using TCP, local error control should meet the following conditions: it should be persistent enough to virtually eliminate non-congestion losses, should avoid packet reordering, and should not delay packets beyond the typical TCP end-to-end retransmission timeout. These conditions avoid false triggering of TCP's end-to-end error control mechanisms, such as congestion avoidance, fast retransmission, and timeouts. Our experiments suggest that violating the third condition may not be catastrophic, causing only a modest duplication of effort.

While these conditions are partly driven by properties of TCP's error control mechanisms, they are not specific to TCP itself. The local error recovery mechanisms do not depend on TCP's header format or acknowledgement generation rules, but instead rely on high-level properties of the end-to-end congestion avoidance and error control mechanisms, which are regarded as appropriate for reliable byte stream transport in general. Other flow types, such as delay-sensitive streaming media, might prefer different error control policies, such as reduced persistence. Flow-type-specific approaches are attractive alternatives to protocol-specific approaches, and can be supported based on flow information obtained from Differentiated Services marking [91] or RSVP-like protocols.

The above results were demonstrated using two local error control implementations. First we implemented a simple, generally applicable design that uses only retransmission to recover from errors. This allowed us to analyze error recovery behavior and to identify conditions under which purely local error recovery is effective. Then we broke the "packet error rate" abstraction barrier and applied our trace data, which faithfully represent loss, truncation, and bit corruption. This allowed us to employ a more sophisticated adaptive local error control implementation designed for WaveLAN hardware that includes local retransmission, adaptive packet shrinking, and FEC coding. Simple adaptation policies outperformed static policies across a range of error environments while allowing effective end-to-end communication using both TCP and UDP.

Finally, the results obtained by TCP running over our adaptive error control policies strikingly parallel the results obtained by last chapter's abstract throughput simulator. This increases the trust we can place in both measurement frameworks.

Chapter 5

Effort-limited Fair (ELF) Wireless Link Scheduling

5.1 Introduction

While the previous chapter demonstrated that adaptive link-level error control enables higher-level protocols to perform well in the presence of even harsh error conditions, our demonstrations were in the context of a single transport-level flow. That is, the throughput achieved by a particular flow was solely a function of the error environment it encountered. When we broaden our examination to include multiple flows (which may be running different protocols), multiple machines sharing a wireless link, and location-dependent errors, it is clear that we must address not only the question of the best transmission strategy for a given flow at one instant, but which flow should be transmitting at that time.

Scheduling algorithms for wired links are defined according to specifications which map offered traffic load to the performance that each flow "deserves". Schedulers employ various fairness notions, such as priority classes, packet deadlines, or specified percentages of link throughput. They may be purely reactive, or may require flows to negotiate via an admission control process before sending. Regardless of these design decisions, information flows in one direction, from offered load to desired outcomes.

However, wireless link errors affect flows in an dynamic and unequal fashion. Flows that "deserve" the same throughput in the absence of errors may differ greatly in their sensitivity to throughput reductions. For example, a file download is injured less by reduced throughput than a videoconference would be. Throughput sensitivity is a function of non-technical considerations as well: on a corporate network, a video feed of a shareholder meeting might be considered more deserving than an entertainment news show. Therefore, when the capacity of a link drops significantly due to errors, a scheduler must feed this information back and somehow adjust the throughput target for each flow. We believe that this adjustment process cannot be automated without external information that specifies not only how to distribute link time in the absence of errors, but also how to distribute link errors when they arise.

This information will allow the scheduler to balance the need for certain flows to meet their throughput targets even in the presence of link errors against the need for to preserve overall network efficiency. If one wireless station is experiencing a much higher error rate than others, we might wish to compensate for that by giving it extra air time for retransmissions. Alternatively, we might want to limit its air time since much of it would be better used on other stations capable of error-free communication. In either case, the function of error-sensitive link scheduling is to implement end-user decisions about which flows most require stable throughput despite a dynamic error environment.

5.1.1 Chapter Outline

The remainder of this chapter is organized as follows. In the next section we list and discuss the principles we used to guide our approach to the wireless link scheduling problem. Then, in Section 5.3, we examine how various granularities of link errors affect link scheduling. Then we use several detailed examples to illustrate the need for error-sensitive link scheduling and the interplay between the competing notions of *effort fairness*, *outcome fairness*, and link efficiency. In Section 5.4 we describe the *power factor* administrative control which defines how a scheduler should respond to capacity loss. Section 5.5 describes a scheduler implementing the power factor. In Section 5.6 we examine the behavior of this algorithm via trace-driven simulation, and Section 5.7 reports on experience with our prototype implementation. In Section 5.8 we place our work in the context of related wireless packet scheduling efforts.

5.2 Design Principles

The results in this chapter are guided by the following list of principles, which formalize the concerns raised above:

1. in an error-free environment, the outcome achieved by a wireless scheduler should be identical to that of an equivalent wireline scheduler;

2. the amount of capacity loss suffered by a flow should not be proportional to its transmission rate or its error rate, but should be configurable through administrative controls;

3. it must be possible to administratively bound the amount of capacity that is lost due to location-dependent errors;

4. in the absence of information to the contrary, flows experiencing equal error rates should experience the same capacity loss; and

5. capacity unused by one flow should be distributed "fairly" among other flows.

While several of our principles (1, 4, and 5) are similar to those used in other projects [76, 89], the principles dealing with capacity loss (2 and 3) are unique. These two principles will guide our design of a new scheduler model that will allow intuitive control over the effect of capacity loss.

The complexity of the problem can be illustrated by considering a "straw man" solution, overprovisioning. Some fraction of the link, e.g., 20%, would be set aside to recover from errors, while the remainder would be reserved or otherwise managed by a traditional wireline scheduler. During low-error-rate periods, leftover "error control" air time would be distributed among best-effort flows. This approach is not practical. For a variety of technological and regulatory reasons, wireless links often have a limited, fixed capacity, making this type of overprovisioning undesirable. Furthermore, even reserving a large fraction of the link for error control would not be adequate in all circumstances, since it is not possible to bound the error rate in advance. Another possible approach would be to regularly reset the relative weights or priorities of the flows, taking into account their current error rates. This is not practical for a variety of reasons. As we saw in Chapter 3, error rates change very quickly, so frequent manipulation would be required. Also, as we will discuss in Section 5.5.2, determining per-flow error rates can be very difficult.

The scheduler model we propose focuses on the insight that, in a wireless environment, we must distinguish between "effort" (air time spent on a flow) and "outcome" (actual useful throughput achieved by the flow). While effort equals outcome in a wireline environment, they can be substantially different in a wireless environment. Imagine that a wireline scheduler evaluates a set of flow requirements and determines that a certain flow would be satisfied by 10% of the error-free link capacity. If that flow belongs to a station experiencing a 50% packet loss rate, spending 10% of the link's *effort* on that flow will result in it achieving *outcome* equivalent to 5% of the link's error-free throughput. But if this flow is critically important to some application, it might make more sense to spend 20% of the link's effort on the flow so that it would achieve its expected outcome, 10% of the error-free link throughput. To address this effort-outcome disconnection, we propose an "effort-limited fair" (ELF) scheduling approach. An ELF scheduler strives to achieve per-flow outcomes envisioned by users (such as a specific throughput for constant-bitrate flows or a specific link share for best-effort flows), subject to limits on the effort (transmission time) spent on each flow, as specified by a per-flow *power factor* setting. The power factor is a control knob that can be used to administratively implement a variety of fairness and efficiency policies.

5.3 Justification of Principles

In this section we will analyze the hazards posed by a wireless error environment in order to guide our design process. In particular, we will discuss how link errors of varying granularity may be addressed, including the utility and limitations of time-slot swapping, present differing notions of how a wireless link scheduler should behave in the face of link capacity loss, and discuss location-dependent errors in particular. The insight we gain from this analysis will clarify and support our second and third design principles.

5.3.1 Network model

We assume that a wireless network consists of one or more cells and that the bandwidth in each cell is managed by a centralized packet scheduler. One example of a such a cell is an IEEE 802.11 LAN [70] in which a base station is configured to manage all bandwidth via the Point Control Facility.

We assume that the scheduler views traffic as a set of flows. Flows can be defined in a variety of ways. One simplistic approach would be to define each mobile station's traffic with a cellular base station as a single flow; alternatively, each TCP or UDP connection could be treated as a flow, or flows could be defined by an end-to-end state establishment mechanism such as RSVP. This work is based on the Weighted-Fair Queueing (WFQ) wired-link scheduling approach [35]. In a WFQ scheduler, link time is distributed over the flows according to a set of weights. A WFQ scheduler can be used to implement throughput reservations; in that case, the weights will be adjusted as flows enter and leave the network so that reserved flows maintain their reserved rates. We have chosen WFQ as a basis in part because its conceptual simplicity has resulted in substantial analysis and development [33, 15, 124, 89].

5.3.2 Scheduling implications of link error patterns

Wireless networks experience errors on different time granularities, and network designers have devised a variety of error control mechanisms to battle these errors. Error bursts at the bit level are typically addressed by a mixture of error coding and interleaving (Chapter 2), which reduces but does not eliminate link capacity variation. At a coarser time granularity, some packet errors can be predicted based on packet burst error patterns, and a wireless link scheduler may attempt to reduce the overall packet error rate by swapping transmission slots between stations believed to be experiencing a multi-packet error burst and stations believed to be currently error-free [76, 89]. Again, this approach does not eliminate all capacity loss, because of the imperfect predictability of error bursts and because some interference sources may affect multiple stations or even the base station. We conclude that *capacity loss is a fundamental reality for wireless schedulers.*

To discuss the impact of variable link capacity we will use an example of a wireless cell with a capacity of 800 Kbit/s employing a WFQ scheduler to serve two guaranteed flows and two best-effort flows. We will first assume that all errors are location independent, so that all flows experience the same error rate. The flow properties and weights are summarized in Table 5.1. We will use the term *fidelity* to denote the degree to which a scheduler is "faithful" to a particular flow, namely the percentage of the flow's expected throughput on an error-free link that it actually achieves.

Let us first consider what happens if 50% of all packets are lost. Since all flows are experiencing errors, most wireless fair queueing schedulers would assign transmission slots to the flows according to their weights in Table 5.1. With this *effort-fair* approach, each flow receives 100% of its expected *effort*, but this is degraded to yield only 50% of its expected *outcome* (Table 5.2), so both rate-sensitive flows fail to achieve their desired rates.

Client Flow	Target rate (kilobits/second)	WFQ Weight
Audio	reserved 8	1.0
Video	reserved 350	44.0
FTP1	available	27.5
FTP2	available	27.5

Table 5.1: Client example

Client	Expected rate (Kbit/sec)	Effort-fair rate (Kbit/sec)	Preferable rate (Kbit/sec)
Audio	8	4 $\otimes$	8 $\surd$
Video	350	175 $\otimes$	350 $\surd$
FTP1	221	110 $\approx$	21 $\approx$
FTP2	221	110 $\approx$	21 $\approx$

Table 5.2: Possible results of a 50% packet error rate. Results marked with "$\surd$" represent flows meeting their needs; "$\otimes$" represents un-met needs; "$\approx$" represents flows without specific requirements.

An alternative would be a priority-based scheduler which would allot enough extra effort to the audio and video flows that they would achieve their target rates at the expense of reducing the effort spent on the best-effort FTP flows. This would produce the result shown in the last column of Table 5.2 which, given the target rates shown in Table 5.1, is the correct outcome.

While the priority-based scheduler achieves high fidelity in the above scenario, it is too simplistic to be practical. For example, with a 50% error rate, the priority-based scheduler allocates about 89% of the useful bandwidth to reserved flows ($358kbs/400kbs$). Many network managers would argue that this is unfair to the best effort-flows. Things get even worse when the error rate increases. As soon as the error rate is more than 55%, the FTP flows will receive no throughput even though both the audio and video flows will fail to meet their reservations. Since no flow will be satisfied, this is an unreasonable use of the bandwidth. A better allocation would be to have the audio stream meets its reservation while the other three flows split the remaining bandwidth.

Each of these approaches performs well at one extreme of the error rate spectrum and fails at the other extreme. The effort-fair scheduler ensures that a reserved flow experiencing *any* errors will fail to meet its reservation, even if repairing the errors would require only a trivial amount of unfairness to some other flow; the priority-based scheduler performs unacceptably if *any* high-priority flow experiences a high error rate. An important insight of this example is that, while we *should* help the video flow more than the FTP flows because of its importance, we *can* help the audio flow more than any of the others: multiplying its modest throughput needs by even a large factor to compensate for a high error rate won't drive the link to starvation. This sort of decision calls for a control mechanism which admission control can use to map flow rate and importance into a procedure for distributing link capacity loss among flows. In summary, *capacity loss should result in fidelity loss according to administrative controls.*

5.3.3 Location-dependent errors

Let us now consider the impact of location-dependent errors using an example with two stations, one of which is error-free while the other experiences a 50% error rate. Each station owns one "thumbnail" compressed video flow (100 Kbit/sec) and one FTP flow (best-effort). Table 5.3 summarizes the results for four possible schedulers.

Flow	Error rate (%)	Effort rate (Kbit/sec)	Priority rate (Kbit/sec)	Outcome rate (Kbit/sec)	Desirable rate (Kbit/sec)
Video1	0%	100 $\surd$	100 $\surd$	67 $\otimes$	100 $\surd$
FTP1	0%	300 $\approx$	250 $\approx$	200 $\approx$	167 $\approx$
Video2	50%	50 $\otimes$	100 $\surd$	67 $\otimes$	100 $\surd$
FTP2	50%	150 $\approx$	125 $\approx$	200 $\approx$	167 $\approx$
Throughput		600	575	534	534
Efficiency		75%	72%	67%	67%

Table 5.3: Location-dependent errors. Results marked with "$\surd$" represent flows meeting their needs; "$\otimes$" represents un-met needs; "$\approx$" represents flows without specific requirements.

With effort-based scheduling, the lost capacity is distributed proportional to the error rate experienced by each flow, i.e., only the second station will suffer capacity loss, and its video flow will not meet its throughput expectation. As we saw above, the effort-based scheduler ensures that every reserved flow experiencing errors fails to meet its reservation.

A priority-based scheduler could ensure that both reserved flows are satisfied. We would then need to define how it should divide the remaining link capacity among the (equal-priority) best-effort flows; Table 5.3 assumes this division would be effort-fair. Once again, we cannot actually deploy a priority-based scheduler, since a single high-priority flow experiencing serious errors would starve every flow. Also, the throughput discrepancy between the two FTP flows is very large, and a case could be made that it would be more fair for them to receive the equal throughput.

To address the best-effort fairness issue, we could employ an "outcome-fair" scheduler, which would allocate effort to flows in such a way that all flows would achieve the same percentage of their expected throughput. In this case, an outcome-fair scheduler would ensure that every flow achieves 67% of the throughput it would achieve on an error-free link. In essence, the air time is divided equally among packets sent to the error-free station, lost packets sent to the error-prone station, and packets that successfully reach the error-prone station; each station receives equal throughput and 33% of the air time is wasted. While this would improve fairness among the FTP flows, *neither* video flow would achieve its desired throughput. The outcome-fair scheduler, like the priority scheduler, behaves poorly when any flow experiences a high error rate.

Finally, we will consider a more desirable outcome, in which both video flows meet their needs and both FTP flows receive equal throughput. While this outcome could be accomplished by a hybrid priority/outcome scheduler, this scheduler would become unusable when confronted with any high-error-rate flow.

Notice that the presence of location-dependent errors adds an efficiency dimension to the space of policy options. If all stations experience the same error rate, moving air time from one flow to another does not effect the useful throughput of the cell. However, with different error rates, moving air time from low-error flow to a high-error flow reduces the useful throughput of the cell. This is clearly illustrated by the efficiency row in Table 5.3: outcome-based scheduling results in markedly lower efficiency than effort-based scheduling. All of these considerations call for a scheduler that employs outcome fairness for best-effort flows when error rates are low, uses priority to support reserved flows as long as error rates are moderate, and falls back to effort fairness in the face of very high error rates so that the link will still provide some useful throughput.

This example illustrates two important concepts: *Sometimes flows with different error rates should experience the same fidelity,* but meanwhile *it must be possible to control cell-wide efficiency in the presence of station-dependent errors by administratively bounding the amount of link effort devoted to "fairness."*

5.3.4 Summary

The examples we have presented above argue that setting goals for wireless link schedulers is challenging. Different mixes of flows, considered in terms of their relative throughput requirements and sensitivity to throughput reductions, and different error rates, whether shared or varying per mobile host, seem to argue for different approaches to dividing up air time. We saw that it often makes sense to deliver a flow's expected throughput even in the face of mild errors but abandon it to its fate when its error rate becomes excessive. Below we will investigate a notion of scheduler "fairness" that encompasses both behaviors.

5.4 The power factor

In this section we will present a model of how a wireless link scheduler should adjust flow weights in response to errors in order to create a hybrid between effort fairness and outcome fairness which is parameterized by a single administrative control, the "power factor." We will state a simple weight adjustment criterion, describe the throughput flows achieve as a function of their error rates, show how this weight adjustment can apply to both constant-rate and best-effort flows, and discuss the applicability of this approach. An example will demonstrate that this scheduling approach achieves the sort of per-flow outcomes we sought in the previous section.

5.4.1 Weighted fair queueing with adjustable weights

We will begin with a weighted fair queueing (WFQ) scheduler that distributes effort (air time) according to weights provided by an admission control module. The scheduler will adjust each flow's weight in response to the error rate of that flow, up to a maximum weight defined by that flow's power factor, also provided by the admission control module (see Figure 5.1).

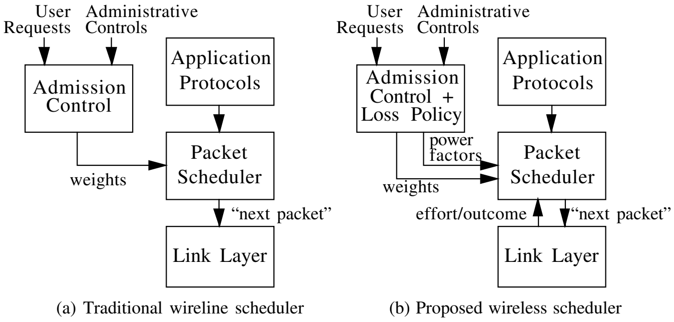

Figure 5.1: Scheduler models

For example, a power factor of 200% indicates that a flow's weight should be doubled in a high error environment, which means that it would gain weight relative to flows with a lower power factor, but lose weight relative to flows with a higher power factor. This makes it possible to, for example, increase the link share of voice and web traffic relative to video traffic in a high error environment. We call this type of scheduler *effort-limited fair* (ELF) because it manipulates flow weights to achieve *outcome fairness* subject to a limit on each flow's *effort*.

5.4.2 The power factor

In order to characterize the behavior of the ELF scheduler across the entire spectrum of error rates, we introduce the following notation. Let us assume we have N flows sharing a link with (error-free) bandwidth B. Each flow has a weight W_i, a power factor P_i, and experiences an error rate E_i. We can now define the adjusted weight of flow i as

$$A_i = min(\frac{W_i}{1 - E_i}, P_i \times W_i) \tag{5.1}$$

That is, for low error rates we scale the weight W_i to make up for the link errors, but we limit the adjustment to a factor P_i. The crossover point is at error rate $E_i^c = \frac{P_i - 1}{P_i}$. We will refer to the error range $E_i < E_i^c$ as the outcome region and the error range $E_i > E_i^c$ as the effort region. The throughput T_i for flow i is given by the product of the transmission time it receives and its success rate,

$$T_i = (\frac{A_i}{\sum_j A_j} \times B) \times (1 - E_i)$$

To justify this approach, we will look at the behavior of the scheduler under some specific conditions.

First, in an error-free environment ($E_i = 0, \forall i$), the scheduler is equivalent to a traditional WFQ scheduler with weights W_i.

As long as a flow is in its outcome region, A_i adjusts to exactly cancel the flow's reduced success rate $(1 - E_i)$, yielding a throughput of

$$T_i = \frac{W_i}{\sum_j (A_j)} \times B$$

That is, the effective weight of the flow is corrected back to W_i, although that is relative to the adjusted weights of the whole link. If all flows are in their outcome regions and they all experience the same error rate E the throughput of flow i becomes

$$T_i = \frac{W_i}{\sum_j \frac{W_j}{1-E_j}} \times B = \frac{W_i}{\sum_j W_j} \times (B \times (1 - E))$$

Thus the scheduler is equivalent to a WFQ scheduler with the original weights W_i running on a E-degraded link, which is exactly outcome-fair.

At the other end of the spectrum, if all flows are in their effort regions, i.e., $E_i > E_i^c, \forall i$, the throughput becomes

$$T_i = \frac{P_i \times W_i}{\sum_j (P_j \times W_j)} \times (B \times (1 - E_i))$$

This means that the scheduler distributes transmission time to the flows in WFQ fashion and the scheduler is "effort fair" (with adjusted weights).

Finally, one of the motivations for introducing the ELF scheduler approach was to limit how much effort (transmission time) is given to any specific flow, so that one flow experiencing very high error rates cannot degrade the performance of the entire link. The highest fraction of the link time that flow i can take is given by

$$\frac{P_i \times W_i}{(P_i \times W_i) + \sum_{j \neq i} (W_j)}$$

when flow i is in its effort region and all other flows are error-free.

5.4.3 Fixed-rate reservations

A traditional WFQ scheduler can provide not only guaranteed link shares but also absolute throughput guarantees. To guarantee a throughput of T_i to flow i, weights must be assigned such that $T_i = \frac{W_i}{\sum_j W_j} \times B$ (and updated when flows enter or leave the system). Providing absolute bandwidth reservations requires explicit support in ELF. The reason is that, while ELF weight adjustment maintains each flow's link *share*, errors decrease the *size* of the link. To obtain absolute throughput reservations ELF will adjust the weights of guaranteed flows upward according to Equation 5.1 while simultaneously *reducing* the weights of best-effort flows in a straightforward way.

To support throughput guarantees, we will define G to be the set of guaranteed flows and B_i to be the bandwidth allocated to each flow i in G. Next, we will use fractions of the error-free link as weights, i.e., $W_i = \frac{B_i}{B}, \forall i \in G$. Admission control will be responsible for ensuring that the link is not overcommitted in both the error-free case ($\sum_{i \in G} W_i \leq 1$) and when all guaranteed flows are error-limited ($B_{G_{max}} = \sum_{i \in G} W_i \times P_i \leq 1$). Next, we aggregate all best-effort flows into one virtual flow with a special weight-adjustment function

$$A_{BE} = 1 - \sum_{i \in G} A_i \tag{5.2}$$

which ensures that the best-effort aggregate flow will consume only whatever link time is left over after every guaranteed flow has either achieved its outcome or has reached its crossover error rate E_i^c.

For any guaranteed flow with $E_i < E_i^c$, $A_i = \frac{B_i/B}{1-E_i}$, so its expected throughput

$$T_i = \frac{A_i}{\sum A_j} \times B \times (1 - E_i)$$

becomes the correct value,

$$T_i = \frac{\frac{B_i/B}{1-E_i}}{1} \times B \times (1 - E_i) = B_i$$

The best-effort flows will avoid starvation if $B_{G_{max}} < B$, in which case an error-dependent amount of transmission time will be allocated to the best-effort aggregate flow, which will distribute it among the best-effort flows using exactly the approach of Section 5.4.2.

5.4.4 Example

This scheduler can support a variety of policies. For example, the hybrid outcome/effort scheduler described in Section 5.3.3, which produced the "Desirable rate" column of Table 5.3, can be implemented by setting the power factors of each flow to at least 200%. The weight adjustments and resulting throughputs are shown in Table 5.4. The top section of the table demonstrates how the scheduler adjusts the weight of the "Video2" flow in response to errors. The center section shows how the errors experienced by "Video2" cause the weight of the best-effort virtual flow to be adjusted *downward* in accordance with Equation 5.2. The efficiency, and hence throughput, of the best-effort virtual flow depends on the mixture of best-effort weights, power factors, and error rates. Finally, note that the base and adjusted weights of the two best-effort FTP flows as displayed in the third section specify how they share the throughput of their virtual sub-link, not the entire link.

In general, setting a flow's power factor to 100% will cause it to be scheduled in an effort-fair fashion, and raising its power factor will cause it to experience outcome fairness over a wider range of error rates. In particular, it is feasible, albeit probably undesirable, to obtain pure outcome fairness for all best-effort flows by setting their power factors to infinity.

Flow	Base Weight	Error rate (%)	Adjusted Airtime Weight	Desirable throughput (kbit/sec)
Video1	$^1/_8$	0%	$^1/_8$	100
Video2	$^1/_8$	50%	$^2/_8$	100
(Best-Effort)	($^6/_8$)	(irrelevant)	($^5/_8$)	(dependent)
FTP1	$^1/_2$	0%	$^1/_3$	167
FTP2	$^1/_2$	50%	$^2/_3$	167

Table 5.4: Achieving the "Desirable" outcomes in Table 5.3.

5.4.5 Choosing power factors

So far we have assumed that an admission control module can set appropriate per-flow power factors though we have not specified how they might be chosen. A few brief examples should suggest the range of plausible options.

Since the vast majority of existing Internet traffic is best-effort and has no end-to-end flow state, it is important to handle this case well. Luckily, reasonable results can be obtained by a local administrative decision to award each best-effort flow a "reasonable" power factor, such as 110% or 120%. The result is that many fluctuations in the local error rate will result in only a mild degradation in throughput, which will be shared among all the best-effort flows. This is more "friendly" to end-to-end protocols, which try to estimate and adapt to throughput variations, than allowing each link-level error burst to stall one or more flows.

An extension to this approach would allow end-to-end protocols to signal, via Differentiated Services marking [91], how to divide their air time between reliability and throughput. That is, a flow selecting reliability might be assigned a weight of 1 and a power factor of 200% while a flow selecting throughput would be assigned a weight of 2 and a power factor of 100%. The first flow would trade away best-case throughput in return for throughput stability in the face of errors. From the point of view of the link, both flows would consume the same amount of air time in the worst case. If this approach were coupled with the per-flow persistence setting previously discussed in Section 4.5.4, the result would be potentially useful for a wide variety of protocols: a single mechanism would allow them to affect the behavior of both wired and wireless link scheduling.

For flows that signal a need for a particular throughput (via, e.g., RSVP), one possibility would be to adapt an existing wireline admission control module in a straightforward fashion. If the link error rate is expected to rarely exceed a certain critical value E^c, assign every admitted reserved flow a power factor of $\frac{1}{1-E^c}$ and stop admitting new reserved flows when they occupy $1 - E^c$ of the error-free link rate, which is when their worst-case air-time requirements would consume the entire link.

Another possibility would be to assign power factors according to flow classes. For example, flows requesting 8 Kbit/s or 64 Kbit/s could be categorized as voice flows and assigned a power factor of 300%; a second throughput range, appropriate for compressed video, could be assigned a power factor of 150%. Such an admission control module would need to avoid overcommitting the link, and would need some policy to determine how much of the link could be assigned to each class.

Our goal is not to advance the power factor as a new end-to-end signalled flow parameter. Rather, we expect that ELF will be a tool used to implement locally-desirable policies.

5.4.6 Discussion

The proposed "power factor" scheduler model meets the requirements outlined in the introduction. By setting the power factor appropriately, administrators can control the degree to which the fidelity of a flow will be maintained in the presence of errors. Selection of the power factor should consider the relative importance and demands of flows (e.g., audio is typically more valuable than video while requiring less bandwidth), and should also consider fairness issues across classes (e.g., reserved traffic should not be able to starve best-effort traffic). The power factor can also be used to control efficiency. For example, by keeping all power factors below 200%, we can maintain whole-link efficiency over 50% (unless that is impossible, i.e., every flow experiences an error rate above 50%).

In this section we demonstrated how adding a per-flow power factor setting transforms a wireline WFQ scheduler into a hybrid effort/outcome WFQ scheduler. We believe that the power factor is also a useful abstraction for other schedulers. One example is a priority scheduler. We could associate power factors with the different priority levels to avoid high priority traffic completely starving low priority traffic or flows suffering from location-dependent errors consuming a disproportionate fraction of the link capacity in a futile attempt to obtain their expectations. Schedulers which offer more complex service characterizations will require additional work. A deadline-based scheduler or any scheduler attempting to meet delay or jitter guarantees will clearly need to track outcomes for individual packets rather than entire flows.

5.5 Scheduler Algorithm

Transforming ELF from a fairness specification appropriate for wireless links to an actual error-sensitive wireless link scheduler requires several steps. Features of our prototype network will suggest a particular base scheduler to which we will add ELF features. To cast ELF into an algorithm we must address the difficulty of measuring error rates. After we sketch the operation of our ELF scheduler with reference to pseudo-code we will discuss some still-open design issues.

5.5.1 Scheduler model

In this section we will present an algorithm for a particular ELF scheduler based on weighted round-robin (WRR). This scheduler will support reserved flows with absolute-rate reservations and best-effort flows which distribute the remaining capacity via weights. Flows in both classes will be bounded by effort limits. We use packet weighted round-robin (WRR) instead of weighted fair queueing, measuring throughput in packet slots. One reason for using WRR instead of WFQ is simplicity. Also, wireless networks are often slot-based for synchronization and power-management purposes. Finally, the prototype network we use for experiments in (Section 5.7) has a relatively high per-packet cost, so charging on a per-packet instead of a per-byte basis is actually quite reasonable.

5.5.2 The difficulty of measuring error rates

The automatic weight adjustment approach of Section 5.4 assumes that a scheduler can easily measure per-flow error rates. However, measuring error rates is far from straightforward.

One problem is that it is inherently difficult to obtain timely information about a flow's current error rate. This is particularly true for low-throughput flows. Since they transmit infrequently, the standard procedure of using multiple observations to filter out measurement noise would base the error rate estimate on a long period of time. If this timespan is longer than the typical amount of time an error rate remains stable, the scheduler's error estimate will be essentially random.

A second problem, measurement timing, affects both low-throughput and high-throughput flows. Transmission timing can greatly affect observed per-flow error rates. Table 5.5 demonstrates how both per-flow and whole-link error rates depend as much on timing as on actual link conditions. In the table, `O` represents an error-free time slot for a particular flow and `X` represents a time slot which will cause a packet loss; $\boxed{\texttt{O}}$ and $\boxed{\texttt{X}}$ represent time slots when the scheduler actually chooses to transmit for that flow. If we consider air time, Flow 1's "true" error rate is 0% and Flow 2's is 50%, but different packet transmission timings vary Flow 2's *observed* error rate from 0% to 100% and the link's error rate from 0% to 50%.

This example argues that, even if a scheduler can accurately predict a flow's true error rate, this is of limited use without very accurate prediction of error *timing*. For example, a scheduler seeking to meet a short-term whole-link efficiency target might be forced to avoid a specific flow because unfortunate timing effects were currently "artificially" raising its error rate above the "true" value.

A pathological sub-case of the timing problem is posed by error bursts. If a flow's errors are typically bursty rather than uniformly distributed, a rapid retransmission is likely to fall within an error burst. Thus, increasing a flow's weight and giving it extra air time can actually increase its error rate.

Finally, the whole-link error rate is a function of not only timing but also location-dependent errors and power factors. Transmitting on behalf of different flows yields different link-capacity measurements. Consider two flows with weight 50% and power factor 100%. If one flow encounters a 50% error rate, the link capacity will be 75%. If that flow's power factor were 200% rather than 100%, it would use two thirds of the air time, wasting half of that, and the link capacity would drop to 67%.

To the extent that per-flow and whole-link error rates are either not measurable or even not clearly defined, the situation calls for a scheduling algorithm that approximates ELF airtime allocation without relying on explicit computations of per-flow error rates or whole-link capacity. Such an approximation will be presented in the next section.

Flow 1	O	$\boxed{\texttt{O}}$	O	$\boxed{\texttt{O}}$	O	O	$\boxed{\texttt{O}}$	$\boxed{\texttt{O}}$	$\boxed{\texttt{O}}$	O	$\boxed{\texttt{O}}$	O
Flow 2	$\boxed{\texttt{O}}$	X	$\boxed{\texttt{O}}$	X	$\boxed{\texttt{O}}$	$\boxed{\texttt{X}}$	O	X	O	$\boxed{\texttt{X}}$	O	$\boxed{\texttt{X}}$
	$E_1 = 0\%$				$E_1 = 0\%$				$E_1 = 0\%$			
	$E_2 = 0\%$				$E_2 = 50\%$				$E_2 = 100\%$			
	$E_{tot} = 0\%$				$E_{tot} = 25\%$				$E_{tot} = 50\%$			

Table 5.5: Error rate as a function of transmission timing.

5.5.3 Algorithm Overview–tracking deserved throughput, effort, and outcome

We will explain our scheduling algorithm using the pseudo-code in Figures 5.2 and 5.3. The code embodies several simplifying assumptions. All flows have data queued at all times. Transmission time is slotted into fixed-size packets, though an adaptive link layer may choose to send shorter packets to avoid interference-related truncation. Each packet transmission fully succeeds or fails (there are no sub-packet acknowledgements).

Admission control converts each flow's weight into its expected inter-transmission interval (a flow expecting 50% of the link would transmit every two time slots; this is referred to as "WRR with WFQ-like spreading" in [88]). The transmission intervals of best-effort flows apply not to actual link time but to a dilated virtual time (see below). Implementing the power factor policy requires us to consider, for each flow, the amount of throughput it should have experienced in the absence of link errors, the amount of throughput it has actually achieved, and how much air time we have spent trying to reduce the gap between these values. The scheduler keeps tracks this information via two per-flow variables, `deserve`, which stores the number of packets by which this flow's achievement lags the ideal situation, and `effort`, which contains the amount of air time allocated to this flow but not yet expended. The per-flow scheduler state is summarized in Table 5.6.

Flow parameter	Meaning
Inter	Flow's transmission rate (expressed as interval, in slots; equivalent to $^1/_{\text{rate}}$)
Power	Power factor (fixed point: 100% represented as 100)
Last	Timestamp: start of flow's most recent interval
Deserve	Number of packets of throughput deserved (fixed point)
Effort	Number of airtime slots banked (fixed point)

Table 5.6: Per-flow scheduler state. The first two fields are set by the admission control mechanism and the other three constitute the scheduler's internal state.

Within each flow group (guaranteed or best-effort), the scheduler follows the same two steps. First, each time the flow group's clock advances, `tick` scans for flows whose inter-transmission time period has passed, allocating each one a single packet's worth of deserved throughput and the amount of effort specified by its power factor. The flow will remain eligible to transmit until it either achieves that throughput or exhausts that effort.

Second, `choose` is used to determine which flow, if any, should use the next transmission slot. In this presentation `choose` will consider every eligible flow though other mechanisms, such as burst error prediction (see below), could filter some flows out. When multiple flows are eligible for transmission, `choose` selects the most urgent. We will choose the flow whose outcome is currently lagging its deserved throughput by the greatest percentage, breaking ties by serving low-throughput flows first. We prefer to serve low-throughput flows on the grounds that jitter reduction might be more important to them. If `choose` detects no eligible flow it indicates this to the caller.

The main scheduling function `schedule`, is called by the link layer when it is ready to send a packet (Figure 5.3). It will first try to select an eligible guaranteed flow before moving on to best-effort flows. That is, guaranteed flows are served until each is either satisfied with its outcome or limited by expended effort, and the best-effort flows share the leftover bandwidth. The main scheduler decision procedure operates as follows. First it uses `tick` to inform guaranteed flows of the passage of time, and then it asks `choose` whether any guaranteed flow should currently transmit. If not, it moves on to the best-effort class, calling `tick` with the best-effort virtual time and then invoking `choose`.

While reserved time increases for each link time slot, best-effort time increases only when no best-effort flow is eligible. To see why this is correct, recall that the basic ELF scheduler provides flows in their outcome regions with the link *share* they expect, but that these shares are fractions of the error-degraded link throughput. Thus, any errors that are encountered must slow down the virtual time experienced by best-effort flows. For example, if every best-effort flow is experiencing an error rate of 50%, every two packets of deserved throughput will require three airtime slots. Since on average each best-effort flow's eligibility will last 1.5 slot times, ticking the best-effort virtual clock only at the end of eligibility provides the correct timing. In the other direction, sometimes no flow, guaranteed or best-effort, is eligible. In that case the best-effort clock is advanced until a flow is eligible (or, if there are no best-effort flows, a reserved flow will be served early).

At this time the link layer transmits a packet for the designated flow, employing whatever combination of packet length and error coding recently history suggests is prudent for successful communication with that flow's peer machine. Before the next scheduling decision is made, the link layer will inform us of the transmission outcome using the function `update`, which will bill the designated flow for the effort expended and update its achievement value appropriately. For example, if the error control implementation believes the flow is likely to experience packet truncation, it might choose to send a short packet (effort = 50%); the flow's "deserve" will be decremented by 50% if the transmission succeeds and will be unchanged otherwise. This assumes the existence of a fast link-level acknowledgement, a feature common to many wireless LAN MAC protocols [70, 63, 20]. The necessity for this

```
/* per flow state kept by scheduler:                */
/*  inter - inter-transmission interval, 1/rate      */
/*  last  - timestamp of last eligibility to send    */
/*  power - power factor (as percentage)             */
/*  deserve - amount of *outcome* flow deserves      */
/*  effort  - amount of *effort* flow is entitled to */

/* "tick": update the deserve and effort of each flow  */
tick(flow-list, now)
{
   foreach flow (flow-list)
      if (flow.last + flow.inter <= now)
         shouldserve(flow, now);
}

shouldserve(flow, now)
{
   ++flow.deserve;
   flow.effort += flow.power / 100;
   cap_effort(flow);
   flow.last = now;
}

/* "choose" next flow from a class of flows */
choose(flows, &resultflow)
{
   int bestwlag = 0;  /* best weighted lag */
   int bestinter = 0;
   int bestdeserve = 0;
   flow bestflow = NULL;

   foreach flow (flows) {
      /* weighted lag: deserve / rate */
      int wlag = flow.deserve * flow.inter;

      /* must have effort */
      if (flow.effort < 1)
         continue;

      /* most deserving flow has highest % lag */
      /* tie breakers: low-rate flows first;   */
      /* then flows with more absolute lag    */

      if ((wlag < bestwlag) ||
         ((wlag == bestwlag) && (flow.inter > bestinter)) ||
         ((wlag == bestwlag) && (flow.inter == bestinter)
            && (flow.deserve < bestdeserve)) {
         continue; /* not the most deserving */
      }
      /* becomes the most deserving */
      bestwlag = wlag;
      bestinter = flow.inter;
      bestdeserve = flow.deserve;
      bestflow = flow;
   }
   *resultflow = bestflow;
}
```

Figure 5.2: Pseudo-code for core of the scheduler

explicit two-stage transmission and billing feedback is unique to error-prone links, and the

```
/* "schedule" picks next flow to be served */
schedule(&selectedflow)
{
    static int now, dilatednow;
    flow flow;

    ++now;
    tick(protected-flows, now);
    choose(protected-flows, &flow);
    if (!flow && length (unprotected-flows) > 0) {
        choose(unprotected-flows, &flow);
        while (!flow) {
            ++dilatednow;
            tick(unprotected-flows, dilatednow);
            choose(unprotected-flows, &flow);
        }
    }
    if (!flow} {
        /* advance schedule */
        nextflow(protected-flows, &flow);
        shouldserve(flow, now);
    }
    *selectedflow = flow;
}

update(flow,outcome) /* call-back from link layer */
{
    --flow.effort;
    if (outcome == SUCCESS) {
        --flow.deserve;
        cap_effort(flow);
    }
}

cap_effort(flow) /* limit accumulated effort  */
{
    /* retain no more than 4 packets' effort */
    int slack = 4;
    int cap = ((flow.deserve/100)+slack)*flow.power;
    flow.effort = max(flow.effort, cap);
}

/* "nextflow" picks the flow that will fire next */
nextflow(flows, &flow)
{
    flow nextflow;
    int nextfire = INFINITY;

    foreach flow (flows)
        int fire = flow.last + flow.inter;

        if (fire < nextfire)
            nextflow = flow;
            nextfire = fire;
    *flow = nextflow;
}
```

Figure 5.3: Pseudo-code for core of the scheduler - part 2

interface between the scheduler and the error control layer allows separate decision-making about these orthogonal issues.

Finally, whenever we update the status of a flow (functions `tick` and `update`) we check whether the stored effort exceeds a threshold using the function `cap_effort`. This is so that a flow cannot accumulate a large effort bank while its error rate is low and seize the link for an excessive period, starving other flows, when it encounters errors. Since the scheduler always tracks deserved-but-not-attained throughput, capping the effort bank costs a flow only latency, not throughput. We believe `cap_effort` contains a plausible heuristic; one more rigorous approach would be service curves [124].

Table 5.7 briefly illustrates the operation of this scheduler on behalf of two best-effort flows. For the sake of brevity, Flow 2's 50% error rate is in the form of a strict alternation between success and failure, independent of transmission timing (i.e., we counterfactually assume the opposite of reality as described in Section 5.5.2).

Air time	0	1	2	3	4	5	6	final
Virtual time	0	1,2			3,4			
Flow 1								
deserve	0	100	100	100	100	100	100	0
effort	0	200	200	200	300	300	300	200
outcome	0	0	0	0	100	100	100	200
Flow 2								
deserve	0	100	100	0	100	100	0	0
effort	0	200	100	0	200	100	0	0
outcome	0	0	0	100	100	100	200	200
Transmission choice		F2 (fail)	F2 (ok)	F1 (ok)	F2 (fail)	F2 (ok)	F1 (ok)	

Table 5.7: Blow-by-blow trace of ELF scheduling. Reading a column downward from the top yields the scheduler state producing the decision indicated at the bottom

5.5.4 Possible extensions

A question ignored so far is to what extent best-effort flows should experience "fairness" over a long term. That is, if a flow experiences errors for a period and loses throughput as a result, is it entitled to regain that lost throughput later? This corresponds to the question of whether there is a cap on each flow's `deserve` value. If this value is allowed to grow without bound, as in the current implementation, a flow will eventually reclaim the lost throughput, at the expense of other flows in its class, when it experiences a low-error period. Alternatively, this value could be capped or aged.

A related issue is what should happen to a flow that becomes idle. The policy of the in-kernel implementation is that the flow's `deserve` and `effort` would cease to increase (the simulator, lacking idle flows, ignores this issue). Instead, we could allow an idle flow to accumulate both deserve and effort up to some threshold; this would then allow the flow to burst for a short while when it becomes active. Note that deciding whether a flow is idle may be difficult for a centralized scheduler when error rates are high since queue length information must be transmitted to the scheduler by mobile hosts.

This scheduler implementation is orthogonal to swapping in the sense that, if there is some good predictor of when a station is unreachable, `choose` can easily skip known-unreachable stations.

The ELF approach could be used as the policy for an 802.11 Point Control Function (PCF) [82]. If we assume that one station is in a position to discover the outcomes of most packet transmissions, the LAN could operate according to the more-efficient Distributed Control Function most of the time, and the controlling station could then use the PCF period to allocate extra effort to stations according to their power factors.

5.6 Simulation

To evaluate our scheduling algorithm in a repeatable fashion we subjected it to a simple trace-based simulation. The simulator assumes all flows are continuously busy, and records throughput allocation decisions made by the scheduler (ignoring how real transport protocols might react to allocation variations). Once a time slot is allocated to a given flow, the FLEX adaptive error control policy (Section 3.3.2) decides which error control techniques to use and the simulator applies the next event in the trace stream to determine success or failure.

We will plot the error-free link-level throughput each flow achieves, as a percentage of what it would expect in the absence of errors, on its own independent vertical axis. Each plot will include a "Link" pseudo-flow which represents the total link throughput as a fraction of the error-free link throughput. Each flow will be represented by a moving-window average throughput, sampled at the same rate as its expected inter-transmission interval (that is, a flow expecting 50% of the link is sampled five times as frequently as a flow expecting 10%). Each point on the graph is the mean of the 10 most recent samples.

We will present the results of three simulator runs. In the first, we evaluate the scheduler's performance in the motivating scenario of Table 5.2. Two protected flows, audio and video, expect 1% and 44% of the link, while two unprotected flows will share the remainder. Throughout the trace, the link will experience a uniformly distributed 50% packet loss rate which is independent of which flow's packet is being transmitted. We would like the reserved audio and video flows to experience little or no disruption while the two best-effort flows should equally divide the remainder of the link.

Figure 5.4 shows the results of an effort-fair scheduler: each flow experiences a bursty 50% throughput reduction. While this is an acceptable outcome for the best-effort FTP flows, it is disastrous for the video and audio flows. This is especially unfortunate given the trivial amount of extra air time which would be required to give the audio flow full fidelity.

The effects of ELF scheduling can be observed by comparison with Figure 5.5. The audio flow has a power factor of 300% in an attempt to ensure it will survive most plausible link errors. The video flow has a power factor of 223%, a value chosen so that it and the audio flow will consume the entire link in periods when the error rate is 55% or more. The two best-effort flows each have a power factor of 120%, which will enable outcome-based fairness between them in the face of light errors. The fidelity achieved by the FTP flows drops dramatically from 50% to approximately 8%. However, the audio and video flows

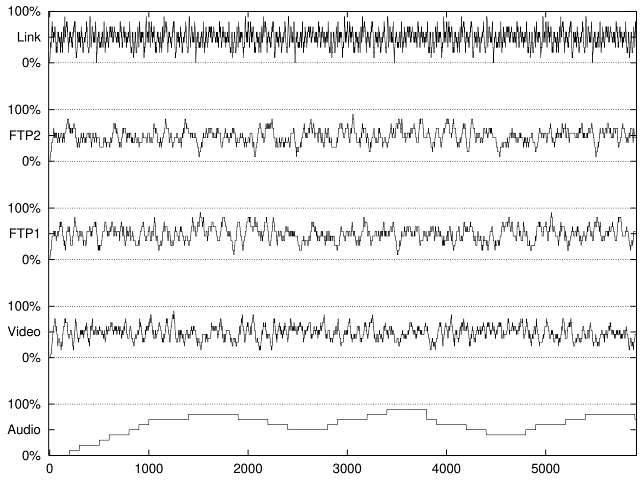

Figure 5.4: Effort-fair scheduling in a 50% loss scenario suggested by Table 5.2.

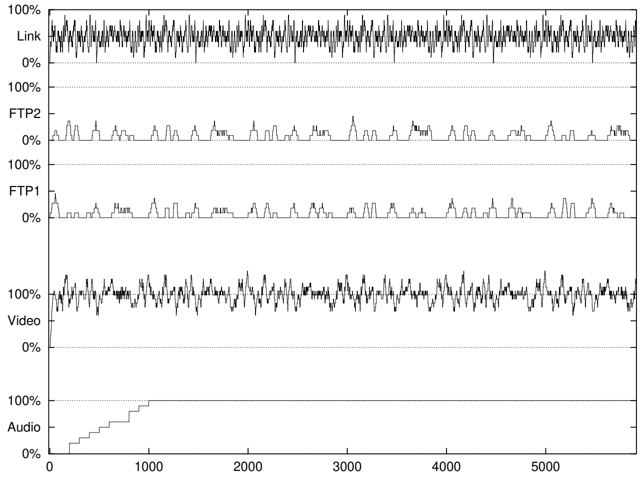

Figure 5.5: ELF scheduling in a 50% loss scenario suggested by Table 5.2.

both achieve 100% fidelity. In the case of the video flow, which is more demanding, this throughput is bursty, with intervals of reduced fidelity balanced by catch-up intervals. It is feasible for the ELF scheduler to provide the low-throughput audio flow with essentially perfect service.

Figures 5.6 and 5.7 demonstrate how the same flows would fare in a more interesting error environment. In the "walls" error trace (Section 3.1.3), both packet losses and bit corruptions vary in severity due to a person moving through the main signal path.

At first glance the two plots appear similar: error bursts, visible as sharp throughput drops on the "Link" plot, immediately and briefly affect every flow. The differences, while subtle, are significant. The video and audio flows receive an upward compensatory throughput spike after each dropout. The cost is that each dropout is slightly worse for the two FTP flows. In effect the ELF scheduler has shifted much of the end-to-end throughput adaptation

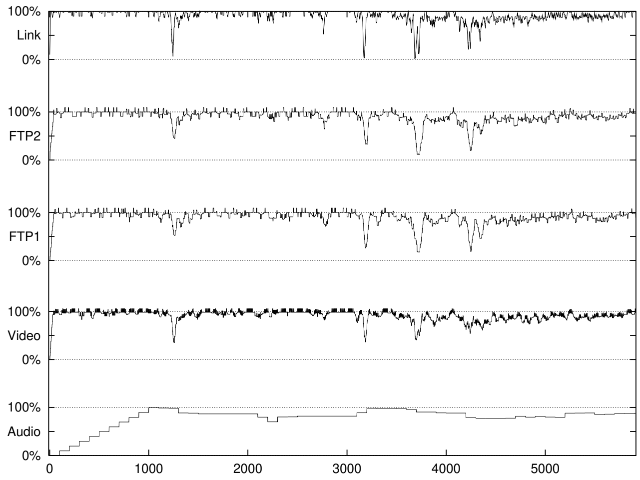

Figure 5.6: Effort-fair response to a dynamic real-world error trace.

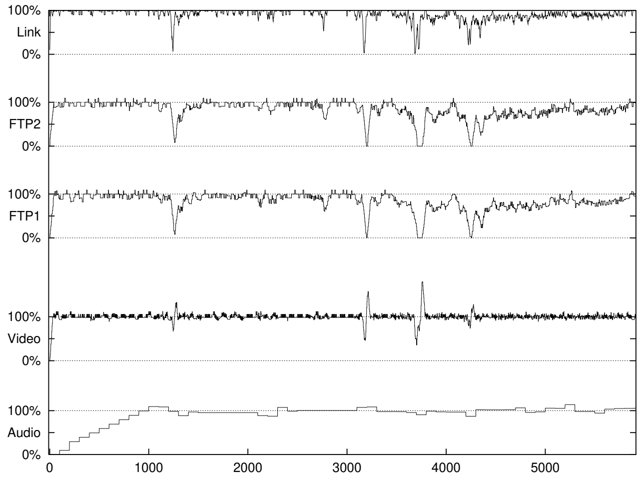

Figure 5.7: ELF response to a dynamic real-world error trace.

burden from the rate-protected flows to the best-effort flows.

ELF can be used to mitigate the effects of location-dependent errors, as demonstrated in Figures 5.8 and 5.9. The video flow and the "FTP1" best-effort flow belong to a station experiencing significant errors, while the other two flows belong to a station encountering none. The error-prone station observes a smoothly increasing frequency of packet loss, ranging from 10% to 90%, surrounded by loss-free periods. Without ELF scheduling, the errors directly affect FTP1 and the video flow.

If we assign the video and audio flows power factors of 140% and 300%, and employ ELF scheduling, we improve fairness between the two FTP flows and provide better service to the video flow as follows.

From packet 1000 to packet 2000, the scheduler shifts all of the video flow's error burden and some of FTP1's burden to FTP2 so the best-effort flows experience equal throughput.

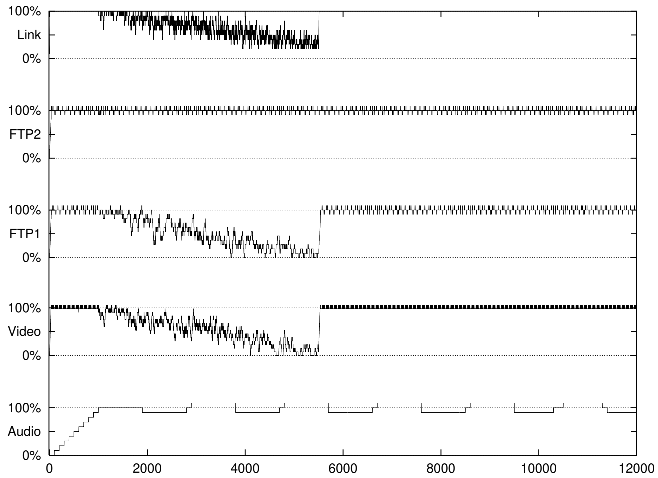

Figure 5.8: Location-dependent error trace, effort-fair scheduling.

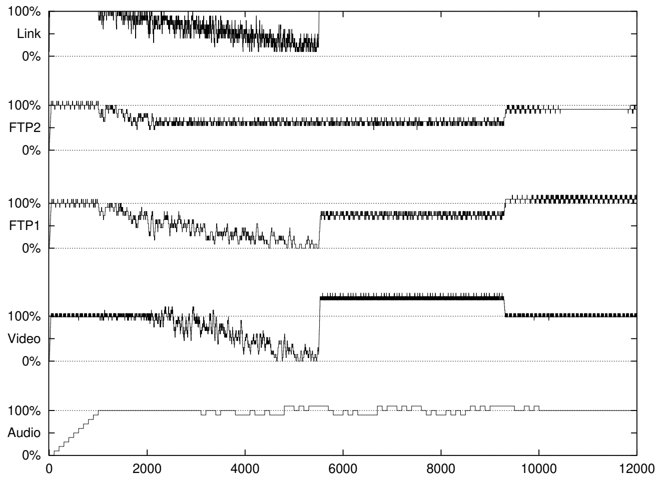

Figure 5.9: Location-dependent error trace, ELF scheduling.

Then, however, flow "FTP1" exhausts its outcome region and begins to lose throughput relative to "FTP2." Next, errors begin to overwhelm the video flow's ability to demand extra link time. From this point on, any increase in the error rate is shared by "Video" and "FTP1". When the link no longer experiences errors, we observe two interesting phenomena. First, the video flow receives throughput beyond its reservation so it can clear its lag, but its power factor still limits its link time so that the FTP flows are not starved. Likewise, the ill-fated "FTP1" flow receives more throughput than "FTP2" so it can achieve long-term fairness with "FTP2." Again, this short-term unfairness is limited by its power factor.

These scenarios illustrate the main goals of our scheduler. We have observed prioritization of reserved flows over best-effort flows coupled with effort-limited outcome-based fairness. The result is that reserved flows meet their throughput needs as long as their error rates are lower than the error allowance specified by the power factor. The scheduler performs intuitively across a wide range of desired flow throughputs (1% to 44% of the link) and error rates (10% to 90%).

5.7 Implementation Prototype

In addition to our simulator-based evaluation, we wished to verify that our scheduler would function acceptably in a more "real-world" environment. We inserted the ELF scheduler into our existing prototype wireless LAN, built from Intel 80486 and Pentium laptops running NetBSD 1.2 and using 915 MHz PCMCIA card WaveLAN [127] units. The kernel device driver includes a simplified poll/response Medium Access Control (MAC) protocol, similar in spirit to the IEEE 802.11 Point Control Function [70]. In our experiments, one laptop operates as a master/base station, while the others use the slave protocol. We will report user-level throughput numbers obtained by the standard kernel TCP stack. In order for TCP to make meaningful progress in the face of these high error rates, we employ the transparent link-level error control mechanism described in Chapter 4. As we discussed in Section 5.5.3, error control is largely orthogonal to scheduling, so we could have obtained similar experimental results by disabling error control and reporting UDP throughput observed by the receiver instead of TCP throughput observed by the sender.

The scheduler consists of 400 lines of code, of which approximately 150 lines are the heart of the algorithm and the remainder is glue code, memory management, tracing support, and trace-replay error injection code.

During the course of moving from simulation to in-kernel implementation, we confronted several issues, centering on accounting for air time.

The first issue is that any shared medium link scheduler must determine which stations have data queued for transmission. Since our prototype ELF scheduler was designed as a centralized algorithm run by a single polling master station, we employed the simplest approach, namely polling. In many situations a distributed queue occupancy algorithm, typically using random backoff within per-priority time windows, would be preferable.

Timeouts were the second issue. If our master station polls a slave and either the poll or the response is lost, this is detected by a fairly long timeout. The flow in question is then billed a full timeslot worth of effort and makes no throughput progress. If we had the luxury of implementing the MAC algorithm in hardware, real-time responses would allow the cell master to detect the absence of a timely slave response partway through a time slot and try a smaller transaction, with the same slave or another, during the remainder of the time.

The third issue was piggybacking. In principle, each TCP session would require two flows, one in each direction. Though TCP provides a bidirectional byte stream, in practice much of the time there is a "forward" flow carrying data and a "reverse" flow carrying acknowledgements. In the case of a remote login session running on TCP, often packets flowing in one direction contain big blocks of program output, while the other direction consists of a mixture of acknowledgements and small keypress packets. Since our software MAC implementation had a very high per-packet cost, we wished to avoid reserving throughput for the "reverse" flows and charging them a full time slot worth of effort for every small packet. Our solution was to implement bidirectional MAC-level flows and to allow a small fixed amount of reverse traffic (up to 100 bytes) to travel for free. The result is that a file transfer is billed for large data packets but not for the small TCP acknowledgements piggybacked on MAC-level Poll or Data frames. A full-blown implementation would need to address the air time needs of the reverse flow (especially for forward flows using throughput reservations).

We will present `tcptrace` output for four TCP streams generated by a master and two slave stations. Each slave receives one reserved flow (link fraction: 5%, power factor: 300%) and one best-effort flow (power factor: 150%). One slave station experiences no errors and the other begins by experiencing no errors but then experiences 10 seconds each of 20% and 50% packet loss rates before returning to an error-free condition.

We would expect that the reserved flows always achieve their expected throughputs and that the best-effort flows would experience equal throughput whenever their error rates were below 30%. The traces in Figure 5.10 meet our expectations (the initial upward spike in throughput displayed for each flow is an artifact of the throughput averaging done by `tcptrace`). The slight sag in throughput observed by the reserved flows is an artifact of the software implementation of our MAC protocol (we are forced to use conservative time-outs to avoid packet collisions, so a lost packet takes slightly more air time than a packet that arrives). The prototype implementation confirms the more detailed simulator results presented earlier.

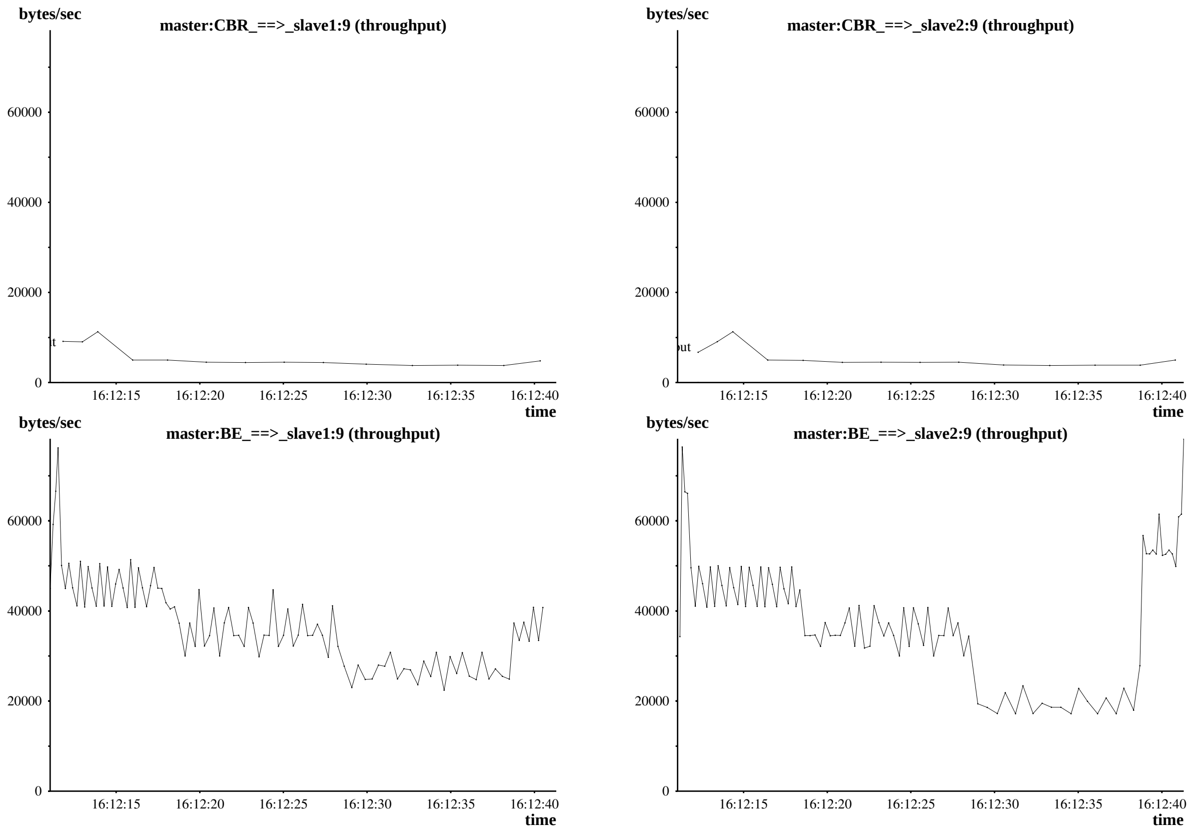

Figure 5.10: Traces of reserved and best-effort TCP streams

5.8 Related Work

5.8.1 Channel State Dependent Packet Scheduling

Channel State Dependent Packet (CSDP) scheduling [19] is based on the observation that link-level retransmission interacts poorly with FIFO scheduling in the presence of error bursts at a single mobile receiver. It is possible for all downlink packets to be trapped behind a single packet destined for a temporarily unavailable receiver, resulting in unfairness and inefficiency. CSDP scheduling replaces a single FIFO link queue with per-station queues and uses link status information to block transmission to unavailable stations, achieving throughput and fairness improvements. These results are based on simulations of TCP in a Markov-model error environment, simple scheduling policies (FIFO, round robin, and longest-queue-first), and simple channel state estimators (perfect advance knowledge and no knowledge).

5.8.2 WPS

Fair queueing in an error-prone wireless environment is examined in [76]. The WPS wireless packet scheduler extends weighted round robin in an attempt to provide fairness, swap time allocations between stations experiencing error bursts and currently error-free stations, avoid scheduling bursts (so a packet burst won't collide with an error burst), avoid polling stations experiencing an error burst, and ensure that stations not experiencing errors receive their expected throughputs. WPS takes into account imperfect information about link state and client queue occupancy. A recent extension of this work [77] adds a variety of attractive properties, such as separate management of delay and bandwidth and graceful trading off of bandwidth between leading and lagging flows. Evaluation is via a simulator using a Markov-model error environment. The main difference between this work and ELF is our belief that it is frequently appropriate for error-free flows to yield throughput to flows experiencing errors.

5.8.3 CIF-Q

A set of formal fairness properties for evaluating wireless versions of wired packet fair queueing algorithms is proposed in [89]. Among these properties are that error-free stations should not lose service to error-prone stations and that stations which receive extra throughput due to another station's unavailability are not forced to pay back this excess via a long service outage. These properties are embodied in a proposed algorithm, Channel-condition Independent packet Fair Queueing (CIF-Q), that achieves these fairness properties in the face of errors. Both analytical and simulation results are presented, the latter based on an on-off error model. ELF differs by being able to recover from the errors of some flows at the expense of other flows.

5.8.4 Server-Based Fairness Approach

The *Server-Based Fairness Approach* [111] creates one or more virtual "server" flows that are used to compensate flows for errors they have experienced in the past. The amount and timing of compensation depend on the amount of capacity reserved for each flow's compensation server, the relative weight of that flow compared to others sharing the same compensation server, and the error rates experienced by all flows compensated by that server. This approach is powerful due to its generality. For example, it is possible for SBFA embedded in HFS-C to approximate ELF by constructing a tree with two top-level nodes (reserved and best-effort, with the best-effort node allocated a tiny fraction of the link), pairing every ELF flow with its own compensation server flow, and assigning weights appropriately. One difference between the resultant scheduler and ELF is that ELF will try to meet the latency expectations of low-rate flows experiencing errors at the expense of slightly increasing the latencies of high-rate flows, even those not experiencing errors. Also, this SBFA construction might suffer from efficiency issues because the compensation server flows (50% of all flows) would frequently transition between being idle and active. The ELF scheduler is designed to react efficiently to link errors, treating them as the common case. We believe that the very generality of SBFA calls for a simple and intuitive fairness model such as our power-factor approach.

5.8.5 Utility-Fair Scheduling

Utility-fair bandwidth allocation [21] presents a framework for allowing flows to specify how much damage they incur as a result of varying amounts of throughput reduction. The scheduler then allocates throughput to each flow so that all flows perceive the same subjective quality. While this approach expresses more information about a flow's needs than ELF does, it is unclear how to apply it to location-dependent errors: it would appear that if a single flow experiences a 100% error rate then all flows will experience a quality level of zero.

5.8.6 Unified Evaluation Framework

Various wireless link schedulers are compared according to a unified framework in [88]. The paper begins by defining a "wireless fair service model" based on a list of seven properties such as short-term fairness among flows not experiencing errors. A modular architectural framework for wireless link schedulers is proposed: seven existing wireless fair schedulers are decomposed into common parts such as error-free service model, lead and lag model, compensation model, etc. Simulations and analysis show that two proposed schedulers, CIF-Q and WFS, meet the proposed principles.

Neither the proposed "wireless fair service model" nor the decomposition model encompass ELF. The core design principle of ELF is that flow fidelity should be a function of flow *importance* as defined by users rather than merely a function of error environment. This principle is at odds with those behind WPS/WFS, CIF-Q, etc.

The disagreement carries over into the architectural decomposition realm as well. The easiest way to see this is to consider a scheduler's response to a time slot during which *every* flow is "blocked" due to a predicted link outage. According to the paper's "Lead and Lag Model", this situation will cause either *every flow* or *no flows* to be compensated for missing that time slot. That is, whether a flow deserves recompense is a function of the scheduler's static design model for all flows rather than being a property settable for each flow. The ELF approach is motivated by the core belief that some flows *do* deserve more scheduler support than others.

Many of the schedulers evaluated in the paper perform timeslot swapping among flows based on channel-state prediction. The paper examines the sensitivity of whole-link throughput to prediction quality. Not surprisingly, for the algorithms evaluated, poor prediction of a flow's channel state markedly reduces throughput for that flow. But this begs the question of what to do about the plausible case of an important flow with a high error rate and poor channel-state prediction. With an ELF scheduler that flow can be assigned a power factor high enough to overcome the errors even if prediction works poorly.

5.8.7 Combining Utility-fairness and error-sensitive scheduling

The WPS wireless link scheduler (discussed above) is augmented with utility fairness, ARQ, and adaptive FEC in [51]. The scheduler monitors each flow's transmission success rate to determine the amount of error coding overhead necessary to avoid decoder failures. Utility functions and coding overhead ratios are the inputs to a procedure that allocates WPS airtime to maximize the weighted sum of all flow throughput utilities.

While the system adjusts transmission time to compensate for error coding overhead, it is not apparent that there is a parallel adjustment for packet loss (potentially caused by header synchronization failure or receiver clock drift).

The system as evaluated in the paper is outcome-fair since utility functions are based on application-level throughput and not link-level transmission air time. Like other outcome-fair systems, it suffers from the risk that a single flow experiencing a high error rate can starve other flows.

The authors discuss the potential desirability of utility functions that account for effort in addition to outcome but do not suggest specific effort-based utility functions or evaluate their behavior.

Among the conclusions of the paper is that static FEC does not perform well in some simulated error environments; this corroborates our trace-based results.

In summary, it is encouraging to see other authors considering variable-rate error coding and per-flow importance specifications. However, we believe that efficiency concerns will require further investigation into utility functions so that outcome and effort can be balanced. In addition, practical difficulties related to measuring per-flow transmission success rates (Section 5.5.2) may be encountered.

5.8.8 802.11e Draft Standard

In Section 1.5.1 we briefly discussed the ongoing work of the IEEE to add quality-of-service scheduling to the 802.11 wireless LAN standard. A simple priority-based scheduler, as proposed, will significantly improve service quality for emerging 802.11b telephone handsets in the face of bulk file transfers. However, this approach is probably not sufficient to fully address questions about allocating air time among members of a QoS class (roughly, telephones and mobile computers) in the face of location-dependent errors or avoiding starving one class if another class encounters a high error rate. Luckily the 802.11e draft standard appears to allow for substantial vendor innovation in link scheduling.

5.8.9 Summary

We believe that an explicit model of the desired outcome of a scheduling algorithm in the face of errors, as provided by WPS, CIF-Q, and SBFA, is valuable. However, we believe it is important in certain situations to give guaranteed flows “special treatment” when they encounter errors, that it is often attractive to slightly reduce overall link efficiency to obtain some amount of outcome fairness among best-effort flows, and that it must meanwhile be possible to protect the link against flows with very high error rates. ELF provides a way of coping with the fundamental challenges posed by wireless link capacity loss.

5.9 Conclusion

In this chapter we addressed the scheduling implications of severe, time-varying, and location-dependent wireless link errors in a groundbreaking way.

We began by developing a new application-centered understanding of the problem, focusing on the need to distribute link errors according to per-flow throughput-sensitivity needs in addition to more-standard criteria such as flow transmission rate, flow error rate, and link efficiency. For example, we might wish to imperceptibly reduce the throughput obtained by a bulk file transfer operation if we could then devote substantial extra transmission time to a low-rate audio flow and hence preserve its quality.

Motivated by various flow and error scenarios, we developed a new definition of fairness for wireless link schedulers. When we explicitly track flow “effort” (transmission time) against flow “outcome” (throughput), we can smoothly transition between two notions, effort-fairness and outcome-fairness, each one attractive at one end of the error rate spectrum but unattractive at the other. Effort-limited fairness is a unifying theme which balances administratively-expressed flow needs (in the form of a “power factor” scheduler parameter) and link efficiency in the face of dynamic and location-dependent errors. It is straightforward to effort-bound both constant-bitrate and link-fraction scheduling.

We presented an implementation of ELF scheduling based on packet weighted-round-robin. This scheduler overcomes the thorny challenge posed by the impracticality of measuring per-flow error rates by approximating ELF-style dynamic adjustment of weights as a function of error rate. While this implementation does not perform timeslot-swapping in response to burst errors, it would be straightforward to employ link-specific burst error prediction.

Finally, we used simulation to illustrate the operation of ELF scheduling in various error environments and briefly presented experimental results for a proof-of-concept in-kernel implementation.

Chapter 6

Conclusion

Wireless links have been a part of the ARPAnet since the 1970's, even before the adoption of TCP. Since that time, mobile hosts have shrunk in size from vans to palmtops. However, researchers are working on many of the same issues, such as type-of-service queueing and adaptive error control [120, 71, 62, 46]. The increasing ubiquity of wireless links has correspondingly increased the importance of deploying well-researched solutions to the challenges posed by wireless links while minimizing disruption to the wired Internet infrastructure.

This dissertation has demonstrated that a combination of techniques allows the existing wired Internet to extend its reach across error-prone wireless links. Adaptive per-link error control vastly increases the throughput even in harsh and dynamic error environments. Then error-sensitive link scheduling enables both best-effort and reserved-service network flows to obtain sensible throughput outcomes. These results are based on real-world trace data replayed in a real operating system kernel.

This chapter begins by summarizing the contributions made by this research investigation in Section 6.1. Areas of future work are discussed in Section 6.2 and closing remarks are in Section 6.3.

6.1 Contributions

The top-level contribution of this dissertation is a convincing demonstration of how error-prone wireless links can be "good citizens" on the Internet. At a finer grain, contributions can be broken down as follows:

- Conceptual Contributions
  - An explanation of how per-hop link-level error control is consistent with (rather than opposing) the the end-to-end argument (Section 4.2).
  - A list of principles we believe wireless link schedulers should obey (Section 5.2).
  - An understanding that wireless link errors pose *policy* and *architectural*, as opposed to merely *timing*, challenges to designers of scheduling algorithms; the sug-

gestion that outcome fairness for best-effort flows in the face of location-dependent errors is useful and sensible (Section 5.3).

  - The notion of Effort-Limited Fairness as a concept balancing per-flow throughput stability and per-link efficiency (Section 5.4).

- Design Contributions

  - An explanation of the difficulty of defining and measuring per-flow error rates (Section 5.5.2).
  - Design of an error-rate-sensitive scheduler that does not require explicit measurement of error rates (Section 5.5.3).
  - An interface accomplishing the necessary "billing relationship" between the LLC layer's adaptive error control and the MAC layer's scheduling function (Section 5.5.3).
  - A two-level, best-effort/constant-bit-rate ELF scheduler (Sections 5.4.3 and 5.5).

- Experimental Results

  - A characterization of the severity and dynamicity of various threats to WaveLAN and demonstration that packet loss, packet truncation, and bit error rate are not tightly correlated across error environments (Chapter 2; Section 3.2.3).
  - A demonstration of the necessity for error control to be adaptive and the identification of a particular simple adaptation policy as effective (Sections 3.3 and 3.5).
  - Public release of error traces (Section 3.7).
  - Three conclusions about TCP and link-level error control (Chapter 4):
    - Evidence that link-level retransmission must be very persistent to support TCP,
    - Evidence that this persistence does not give rise to substantial negative interlayer interactions, and
    - Evidence that link-level retransmission can effectively support TCP, as evidenced by measurements of TCP and UDP throughput, without layer-violating "snooping" or "spoofing" behaviors.
  - A demonstration that sensible per-flow throughput outcomes are feasible despite severe, dynamic error environments and without compromising link efficiency (Chapter 5).

6.2 Future Work

The dissertation work to date suggests several avenues for future research. The applicability of the dissertation results could be broadened by obtaining detailed traces of other wireless LANs, employing more-powerful error control techniques, and investigating power control. The quality of interactions between the link layer and the transport and application layers could be enhanced by automatic retransmission persistence, type-of-service marking, and explicit congestion notification. Research into ELF scheduling could be pushed forward by incorporating ELF into 802.11, incorporating channel-state estimation into ELF, and casting the abstract ELF scheduler into an algorithm in a more rigorous fashion. Finally, addressing inter-subnet blame monitoring may be increasingly important in the near future. The remainder of this section will address these issues in turn.

6.2.1 Broadening Applicability

Per-LAN Error Traces

Present or rapidly-arriving link technologies include IEEE 802.11b, 802.11a, 802.11g, Bluetooth, and HiperLAN. Tuning link-level error control so that each new link type supports Internet traffic well will probably require a study based on detailed packet trace data similar to that presented in Chapters 2 and 3. Capturing these traces is frequently difficult due to a tendency for device vendors to provide a fairly "black-box" interface to the underlying hardware. If this unfortunate situation persists (see Section 6.3.2), individual researchers who happen to have special access that allows them to collect good trace data are in a position to produce studies worthy of publication.

Improved Error Coding

An incremental but probably worthwhile step would be upgrading the error control techniques evaluated in Section 3.2. A simple thing to try would be interleaving, which can noticeably improve the efficiency of both block and convolutional codes. The increasing commercialization of wireless technologies has fueled research into error coding in general, suggesting that employing modern techniques such as "turbo codes" [17, 16] would be fruitful.

Power Control

While it is tempting to imagine "solving" transmission errors by increasing transmitter output power, power control is a complex issue. In general there is constant pressure to *reduce* output power. One obvious reason is that this can significantly increase battery life. However, a less-obvious consideration is each mobile host's radiated power causes interference to other hosts. In fact, a plausible argument can be made for reducing the transmit power of a mobile host while it is far from its base station, so that it will not cause excessive interference to hosts near it which are trying to communicate with other, more-distant base stations [95].

6.2.2 Improving Inter-layer Communication

Automatic Retransmission Persistence

The high levels of retransmission persistence which are important to TCP (Section 4.5.3) are unlikely to suit every application type. On the other hand, it is not clear that individual applications will be in a position to specify either a retransmission count or a maximum queue delay for each link that their packets will traverse. It would be useful to discover per-link persistence heuristics which meet the needs of many transport protocols. One possible approach is based on the increasing tendency of new transport protocols to perform congestion control in a "TCP-friendly" fashion [134]. If each protocol is updating a packet loss rate estimate every few round-trip-times, this may lead to TCP-like notions of packet timeouts. For example, even if a particular streaming media protocol never retransmits packets, it might well declare them permanently lost according to an adaptive delay bound similar to TCP's. In that case, individual links might be able to estimate useful packet lifetimes according to TCP-like metrics. One candidate approach would be for a link to maintain a per-flow running estimate of the time required to successfully transmit a single packet, and to drop a packet from a flow's queue when the head-of-line packet's transmission latency exceeded the flow's typical transmission latency by a few standard deviations.

Type-of-Service Marking

Sections 4.5.3 and 4.5.4 discussed ways for transport protocols to signal to the link layer how much link-level error control is appropriate. In addition, Section 5.4.5 briefly suggested how end-to-end signalling could be used to derive flow power factors. It appears that the field would benefit from investigation into tailoring link-level error control to the needs of real-world applications not characterized well by bulk data transfer on top of TCP. An obvious target for future research would be streaming audio or video on top of UDP or SCTP.

Explicit Congestion Notification

Wired links are beginning to deploy Explicit Congestion Notification [110], which alerts end-systems to the potential onset of network congestion by marking queued packets. While current implementations of ECN typically process (and discard) ECN information at the transport layer, it is probably advisable for interested applications or their middleware layers to obtain and react to this information.

It would be interesting to investigate ECN marking according to both per-packet retransmission count at the LLC layer and effort-bank exhaustion in the ELF scheduler at the MAC layer. In addition to providing feedback to applications, this approach could leverage existing network monitoring infrastructure such as SNMP, which could inform network administrators that certain links are frequently ECN marking.

6.2.3 ELF scheduling

ELF scheduling for 802.11

The progress of the IEEE 802.11e wireless LAN Quality of Service working group [59] indicates that per-flow scheduling for wireless links may become mainstream in the near future. However, Section 5.3 suggests that the priority-based scheduling at the core of the current draft standard may yield limited success. The main difficulty in adding ELF scheduling to 802.11e would be replacing the prototype studied in the thesis research, which relies on straightforward but inefficient polling, with a distributed approximation algorithm.

Initial work toward integrating ELF with the older 802.11b PCF is in progress [82].

Incorporating Channel-State Estimation into ELF

As discussed in Section 5.5.4, ELF could take advantage of link-specific techniques for predicting when a particular station is likely to experience either significant or unrecoverable packet corruption. A simulation study combining detailed error traces, adaptive error control, competing channel-state prediction modules, measuring the resultant whole-link efficiency of an ELF scheduler, would be straightforward and potentially valuable.

ELF Scheduler Formalization and Re-implementation

The current "proof-of-concept" ELF scheduler implementation makes several simplifications that should be remedied. For example, it defers issues related to effort-banking by idle flows and avoids long-term seizure of the link by a single flow through an ad-hoc approach much less attractive than, for example, fair service curves [124]. In addition, the flow choice function requires time linear in the number of flows, which might not be acceptable in some environments.

6.2.4 Inter-subnet Performance Monitoring

Some flows requiring a certain quality of service will traverse multiple administrative domains and subnetwork boundaries. For example, it is easy to imagine that a flow might originate at a popular web server and cross the Internet backbone, multiple links in a corporate Ethernet, and a wireless "last hop." While it is obvious that every network must do its part for the flow to obtain its desired service along the whole path, it is less clear what happens when there is a failure.

Even if each subnetwork performs its own quality monitoring, this might be statistical and coarse-grained in the sense of abstracting the experience of multiple flows over a period of time. Therefore, end-systems may frequently discover trouble early. If so, is there any way for them to narrow down the source of the problem? Will they have any way to control routing to avoid subnetworks not living up to their performance promises?

One approach, Resilient Overlay Networks [3], focuses on improving the performance of traditional best-effort traffic. However, it might be valuable for links that provide service quality to also provide failure alerts in a form more specific than ECN allows.

6.3 Concluding Remarks

6.3.1 Benefits of Measurement-driven Approach

In the course of this work it was very useful to be driven by measurement as opposed to pure intuition.

Detailed error traces were particularly valuable. For example, we observed a rich collection of error environments, with the amounts of loss, truncation, and bit corruption varying widely and somewhat independently, which was not obvious from the synthetic and stochastic error models widely employed in published work on the "wireless link problem". The dynamicity within individual error traces suggested an architectural need for link-level error control to include dynamic on-the-fly packet fragmentation and reassembly. The internal structure of error traces, including the prevalence of "medium-term" station-dependent errors, led to consideration of the wireless link scheduling problem in terms of partially de-coupling the throughput a flow "deserves" from its error rate.

Measurement was also valuable when it came to the question of competing inter-layer retransmissions. Before the results of actual experiments were in hand, it was easy to imagine it was a much more serious issue than it turned out to be. At this point both Reiner Ludwig and myself have observed, in two different network environments, that this is not a major concern (see Section 4.7.2).

6.3.2 Suggestions for Wireless LAN Equipment Designers

Designers of wireless LAN equipment could significantly increase the quality of research (and, hence, the quality of their solutions for end users) by slightly opening up their devices.

It would be very useful if end-system wireless LAN hardware supported data collection. The ability to collect detailed information about transmission and reception failures is invaluable. The level of access provided by WaveLAN is a good start: it is possible to receive even packets that fail link-level CRC. Even better would be an "extended receive" mode which would deliver soft-decision symbols from multiple antennas and delay paths, ideally before iterative decoding.

Further experimentation would be enabled by providing not only information, but control. One product of the Active Networking research community is a variety of interfaces for plugging small pieces of code into routers [125, 18]; research on software-defined radios [25] could be informative as well. The quality of wireless link scheduling research would improve dramatically if researchers could readily evaluate proposed algorithms in running systems. Ideally designers of both end-system hardware and access points would give as much consideration to high-level software *control* over MAC and FEC as to the hardware *acceleration* of these functions.

6.3.3 Closing

I recall with fondness a heady time during the 1990's when it was clear to many in the research community that the "next big thing" in networking was a revolution to overthrow the tyranny of best-effort packet switching in favor of virtual-circuit cell-switching (i.e., ATM). The jury is still out on the question of just how much per-hop per-flow state is required to support important network applications. But I hope that this dissertation helps lay to rest the belief that complex protocol-specific per-flow per-hop "hard state" is necessary for Internet traffic to safely and efficiently traverse short-haul wireless links.

Appendix A

Forward Error Correction background

Forward Error Correction is a mature field and this thesis makes no contributions in that area. However, since we rely on FEC, we include a short overview of the field for the benefit of readers unfamiliar with the area.

The field of FEC is constructed on a firm mathematical foundation, and there are closed-form solutions for the behaviors of many codes under a variety of error models. Error coding books exist at several levels of abstraction: handbooks [94], introductory approaches [4], and in-depth studies [131, 74]. Error codes exist for many different applications, including erasure channels (which can blot out a symbol rather than corrupting it), asymmetric channels (which may tend to force symbols to certain values more than others), and burst error channels (which tend to corrupt symbols in clumped or bursty patterns).

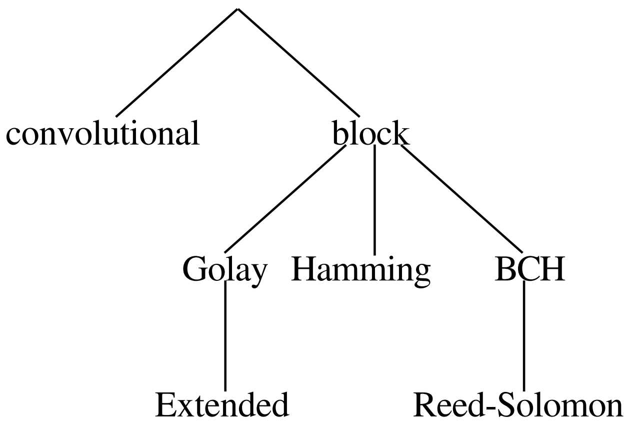

Figure A.1: Partial taxonomy of common error correcting codes

One important distinction is between block codes and convolutional codes (Figure A.1). A block code operates by accepting a block of user data, typically of fixed size, and adding a certain number of extra symbols to the block. As long as the number of transmission errors per block is under a certain threshold, the receiver will be able to correctly recover the original user data. If the number of errors exceeds the correction threshold, but is below a detection threshold, the receiver will be able to report that a block is corrupted. Otherwise, the receiver may incorrectly believe that an entirely different block was sent. The fate of each block is independent of every other block. In some block codes, the block size is directly related to the symbol size. That is, if a transmission channel accepts (and therefore corrupts) two-bit symbols, the block size will be a multiple of two bits. Encoding and decoding block codes typically involves matrix algebra and/or polynomial arithmetic. The ratio of extra symbols to user symbols within a transmitted block may be adjusted smoothly to meet the needs of a particular application.

Convolutional codes, on the other hand, operate on data streams or on variable-sized blocks (above a typically small minimum size). The encoder is a simple linear-feedback shift register, similar to a CRC generator. The effect of the encoder is that each user symbol is "smeared" across some number of transmission symbols (based on the length of the shift register). Convolutional codes have correction and detection thresholds analogous to those of block codes, but these thresholds are related to the number of corrupted symbols within the "smearing window," which is typically on the order of tens of bits. Convolutional code redundancy overheads are typically a small integer multiple of the user data size (block code overheads are often fractional), though there are "punctured" convolutional codes which allow sub-multiple overhead amounts. While convolutional codes are not as straightforwardly robust in the face of burst errors, they have several attractive features:

- They do not impose any data blocking on applications.
- They can be implemented cheaply and efficiently in hardware. Multi-rate codecs which operate at transmission rates of tens of megabits are commercially available as VLSI components.
- They can easily take advantage of "soft decision" demodulation, in which the demodulator provides the decoder not only with the most likely symbol given the arriving signal but also the likelihood that this symbol is correct. Soft-decision decoding typically yields an increase in performance roughly equivalent to doubling the transmission power.

For the purposes of this thesis, we will employ a Reed-Solomon block code. This decision is based on several reasons. First, the WaveLAN hardware presents us with a "hard decision" environment; that is, it provides us with bits, but not with bit certainty estimates, so we cannot take advantage of soft decision decoding. Second, Reed-Solomon codes can operate with a symbol size of 8 bits. This is attractive for software implementations, as byte operations are typically more efficient than bit operations, and network packets are typically an even number of bytes long. Third, the performance of Reed-Solomon codes, as with other linear block codes, is insensitive to the particular user data bits (and we have demonstrated that the underlying channel probably is, too). While production systems may choose convolutional codes to obtain the benefits of soft-decision decoding, the adaptation issues presented by a multi-rate codec are similar whether the code is block or convolutional. As it happens, the recent IEEE 802.11e draft standard [59] employs Reed-Solomon block codes above a direct-sequence spread spectrum physical layer, an approach very similar to the one followed in the course of this research.

Bibliography

[1] Anthony S. Acampora and Mahmoud Naghshineh. An architecture and methodology for mobile-executed handoff in cellular ATM networks. *IEEE Journal On Selected Areas In Communications*, 12(8):1365–1375, October 1992.

[2] Mark Allman and Van Jacobson. On estimating end-to-end network path properties. In *Proceedings of ACM SIGCOMM '99*, Cambridge, MA, September 1999. ACM SIGCOMM.

[3] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*. Association for Computing Machinery, October 2001.

[4] Benjamin Arazi. *A Commonsense Approach to the Theory of Error Correcting Codes.* MIT Press, 1988.

[5] ATM Forum. *ATM User-Network Interface Specification: Version 3.0*, 1993.

[6] AT&T Wireless Communications and Networking Division. Data manual - WaveLAN air interface. Technical Report 407-0024785 Rev. 2, AT&T Corporation, July 1995.

[7] Ender Ayanoglu, Sanjoy Paul, Thomas F. LaPorta, Krishan K. Sabnani, and Richard D. Gitlin. AIRMAIL: a link-layer protocol for wireless networks. *ACM/Baltzer Wireless Networks Journal*, pages 47–60, February 1995.

[8] B. R. Badrinath, A. Bakre, T. Imielinski, and R. Marantz. Handling mobile clients: A case for indirect interaction. In *Proceedings of IEEE WWOS-IV*, pages 91–97, Napa, CA, October 1993. IEEE Press.

[9] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 136–143, May 1995.

[10] Ajay Bakre and B.R. Badrinath. Implementation and performance evaluation of indirect TCP. *IEEE Transactions on Computers*, 46(3), March 1997.

[11] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, December 1997.

[12] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, Mark Stemm, Elan Amir, and Randy H. Katz. TCP improvements for heterogeneous networks: The Daedalus approach. In *Proceedings of the 35th Annual Allerton Conference on Communication, Control, and Computing*, October 1997.

[13] M.H. Barton, J.P. McGeehan, A.R. Nix, and M.C. Lawton. Error rate prediction for high data rate short range systems. In *Wireless Personal Communications*, pages 251–279. Kluwer, 1993.

[14] Frédéric. J. Bauchot and Fabien Lanne. IBM wireless RF LAN design and architecture. *IBM Systems Journal*, 34(3):390–408, 1995.

[15] Jon C.R. Bennett and Hui Zhang. Hierarchical packet fair queueing algorithms. *IEEE/ACM Transactions on Networking*, 5(5):675–689, October 1997.

[16] C. Berrou and A. Glavieux. Turbo codes: General principles and applications. In *6th Tierrenia Int. Workshop on Digital Communications*, Tierrenia, Italy, September 1993.

[17] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo codes. In *IEEE Int. Conf. on Communications*, pages 1064–1070, 1993.

[18] Steve Berson, Robert Braden, Ted Faber, and Bob Lindell. The ASP EE: An Active Network Execution Environment. In *DARPA Active Networks Conference and Exposition*. IEEE CS Press, 2002.

[19] P. Bhagwat, P. Bhattacharya, A. Krishna, and S. Tripathi. Enhancing throughput over wireless LANs using channel state dependent packet scheduling. In *Proceedings of INFOCOMM '96*. IEEE Communication Society, 1996.

[20] Vaduvur Bharghavan, Alan Demers, Scott Shenker, and Lixia Zhang. MACAW: A media access protocol for wireless LANs. In *ACM SIGCOMM '94*, pages 212–225, August 1994.

[21] Guiseppe Bianchi, Andrew T. Campbell, and Raymond R.-F. Liao. On Utility-Fair Adaptive Services in Wireless Networks. In *Proceedings of the Sixth International Workshop on Quality of Services (IWQOS '98)*, Napa Valley, CA, May 1998. IEEE Communications Society.

[22] S. K. Biswas, J. D. Porter, and A. Hopper. Performance of a multiple access protocol for an ATM based pico-cellular radio LAN. In *Proceedings of The Third IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 139–144. IEEE, 1992.

[23] Ethan Blanton and Mark Allman. On making TCP more robust to packet reordering. *ACM Computer Communication Review*, 32(1), January 2002.

[24] Marjory S. Blumenthal and David D. Clark. Rethinking the design of the internet: The end to end arguements vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1):70–109, August 2001.

[25] Vanu Bose, Mike Ismert, Matthew Welborn, and John Guttag. Virtual radios. *IEEE Journal on Selected Areas in Communications*, April 1999.

[26] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource Reservation Protocol (RSVP) – Version 1 Functional Specification, September 1997. IETF Request for Comments 2205.

[27] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*. ACM, October 1998.

[28] Kevin Brown and Suresh Singh. M-TCP: TCP for mobile cellular networks. *ACM Computer Communications Review*, 27(5), 1997.

[29] Ramón Cáceres and Liviu Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE Journal on Selected Areas in Communications*, 13(5):850–857, June 1995.

[30] Jian Cai and David J. Goodman. General Packet Radio Service in GSM. *IEEE Communications*, pages 122–131, October 1997.

[31] Anne-Florence Canton and Tijani Chahed. End-to-end reliability in UMTS: TCP over ARQ. In *Proceedings of IEEE GLOBECOM 2001*, 2001.

[32] Vinton G. Cerf and Robert E. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5):637–648, 1974.

[33] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of ACM SIGCOMM '92*, pages 14–26. ACM SIGCOMM, August 1992.

[34] Spencer Dawkins, Gabriel E. Montenegro, Markku Kojo, Vincent Magret, and Nitin H. Vaidya. *End-to-end Performance Implications of Links with Errors*. The Internet Society, August 2001. Internet RFC 3155 / BCP 50.

[35] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair-queueing algorithm. In *Proceedings of ACM SIGCOMM '89*, 1990.

[36] Antonio DeSimone, Mooi Choo Chuah, and On-Ching Yue. Throughput performance of transport-layer protocols over wireless LANs. In *Proceedings of IEEE GLOBECOM 1993*, pages 542–549, December 1993.

[37] Dan Duchamp and Neil F. Reynolds. Measured performance of a wireless LAN. In *Proceedings of the 17th Conference on Local Computer Networks*, pages 494–499. IEEE, Sep 1992.

[38] David Eckhardt and Peter Steenkiste. Measurement and analysis of the error characteristics of an in building wireless network. In *Proceedings of the SIGCOMM '96 Symposium on Communications Architectures and Protocols*, pages 243–254, Stanford, August 1996. ACM.

[39] David A. Eckhardt and Peter Steenkiste. Improving Wireless LAN Performance via Adaptive Local Error Control. In *Sixth International Conference on Network Protocols*, Austin, TX, October 1998. IEEE Computer Society.

[40] David A. Eckhardt and Peter Steenkiste. A Trace-based Evaluation of Adaptive Error Correction for a Wireless Local Area Network. *Mobile Networks and Applications (MONET)*, 4(4), 1999. Special Issue on Adaptive Mobile Networking and Computing.

[41] David A. Eckhardt and Peter Steenkiste. Effort-limited fair (ELF) scheduling for wireless networks. In *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[42] David A. Eckhardt and Peter Steenkiste. An Internet-style approach to wireless link errors. *Wireless Communications and Mobile Computing (JWCMC)*, 2(1), February 2002. Special Issue on Reliable Transport Protocols for Mobile Computing.

[43] J. Escobar. Run-away dynamics of CDMA channels in an adaptive packet radio network. *Wireless Networks*, 1(1):37–46, 1995.

[44] European Telecommunications Standards Institute. *Radio Link Protocol for data and telematic services on the Mobile Station - Base Station System (MSS-BSS) interface and the Base Station System - Mobile Switching Center (BSS-MSC) interface, GSM Specification 04.22*, version 5.0.0 edition, December 1995.

[45] Gorry Fairhurst and Lloyd Wood. Advice to link designers on link Automatic Repeat reQuest (ARQ), August 2002. Internet RFC 3366.

[46] William C. Fifer and Frederick J. Bruno. The low-cost packet radio. *Proceedings of the IEEE*, pages 33–42, Jan 1987.

[47] Sally Floyd. TCP and successive fast retransmits, February 1995. Obtain via ftp://ftp.ee.lbl.gov/papers/fastretrans.ps.

[48] Sally Floyd, Jamshid Mahdavi, Matt Mathis, and Matt Podolsky. An extension to the selective acknowledgement (SACK) option for TCP, July 2000. Internet RFC 2883.

[49] G. David Forney, Jr. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, March 1973.

[50] Armando Fox and Eric A. Brewer. Reducing WWW latency and bandwidth requirements via real-time distillation. In *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, May 1996. World Wide Web Consortium.

[51] Xia Gao, Thyagarajan Nandagopal, and Vaduvur Bharghavan. Achieving application level fairness through utility-based wireless fair scheduling. In *Proceedings of IEEE GLOBECOM 2001*, San Antonio, Texas, November 2001.

[52] Samil Goel and Dheeraj Sanghi. Improving TCP performance over wireless links. In *Proceedings of TENCON '98*. IEEE Communication Society, December 1998.

[53] Tom Goff, James Moronski, Dhananjay S. Phatak, and Vipul Gupta. Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments. In *Proceedings of INFOCOM 2000*, pages 1537–1545, 2000.

[54] Andrei Gurtov and Reiner Ludwig. Evaluating the Eifel algorithm for TCP in a GPRS network. In *Proceedings of European Wireless*, Florence, Italy, February 2002.

[55] J. Hagenauer. Rate-compatible punctured convolutional codes (RCPC codes) and their applications. *IEEE Transactions On Communications*, 36(4):389–400, April 1988.

[56] Khalid Hamied and Gordon L. Stuber. A non-iterative algorithm for estimating the impulse response of ISI channels. In *Wireless Personal Communications*, pages 175–186. Kluwer, 1993.

[57] Gunnel Hasselgren and Balazs Kiacz. Personal communication, March 2002.

[58] Dr. Joseph H. Hertz. *The Authorised Daily Prayer Book*. Bloch Publishing Company, New York, 1982.

[59] IEEE 802 Committee of the IEEE Computer Society. *Medium Access Control (MAC) Enhancements for Quality of Service (QoS), IEEE Std 802.11e/D1.4*. The Institute of Electrical and Electronics Engineers, New York, New York, October 2001.

[60] Van Jacobson. Congestion avoidance and control. In *Proceedings of SIGCOMM '88: Communications, Architectures, and Protocols*, pages 314–329. ACM SIGCOMM, August 1988.

[61] Martin Johnsson. HiperLAN/2 - the broadband radio transmission technology in the 5 ghz frequency band, 1999. available from www.hiperlan2.com.

[62] John Jubin and Janet D. Tornow. The DARPA packet radio network protocols. *Proceedings of the IEEE*, pages 21–32, Jan 1987.

[63] Phil Karn. MACA–A new channel access method for packet radio. In *Proceedings of the 9th ARRL/CRRL Amateur Radio Computer Networking Conference*, September 1992.

[64] Phil Karn. The Qualcomm CDMA digital cellular system. In *Proceedings of the USENIX Mobile and Location-Independent Computing Symposium*, pages 35–39. USENIX Association, August 1993.

[65] Phil Karn. Toward new link-layer protocols. *QEX*, pages 3–10, June 1994. reprinted from Proceedings of the TAPR 1994 Annual Meeting.

[66] Phil Karn. Error control coding. Technical report, September 1996. http://people.qualcomm.com/karn/dsp.html.

[67] H. Kim, S.K. Biswas, P. Narasimhan, R. Siracusa, and C. Johnston. Design and implementation of a QoS oriented data-link control protocol for CBR traffic in wireless ATM networks. *Wireless Networks*, 7:531–540, 2001.

[68] Rajesh Krishnan, Mark Allman, Craig Partridge, and James P.G. Sterbenz. Explicit transport error notification for error-prone wireless and satellite networks. Technical Report 8333, BBN Technologies, February 2002.

[69] LAN/MAN Standards Committee of the IEEE Computer Society. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band*, IEEE Std 802.11b-1999. The Institute of Electrical and Electronics Engineers, New York, New York, 1999.

[70] LAN/MAN Standards Committee of the IEEE Computer Society. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1999, ISO/IEC 8802-11:1999(E). The Institute of Electrical and Electronics Engineers, New York, New York, 1999.

[71] Barry M. Leiner, Donald L. Nielson, and Fouad A. Tobagi. Issues in packet radio network design. *Proceedings of the IEEE*, pages 6–20, Jan 1987.

[72] Paul Lettieri and Mani B. Srivastava. Adaptive frame length control for improving wireless link throughput, range, and energy efficiency. In *Proceedings of IEEE INFOCOM '98*, pages 564–571, San Francisco, CA, March 1998.

[73] A. Lewis and C. Guy. Assessing the radio performance of wireless LANs for mobile multimedia applications. In *Proceedings of the 2nd International Workshop on Mobile Multimedia Communications*, pages A5/6/1–6. Hewlett-Packard Laboratories, April 1995.

[74] Shu Lin and Daniel J. Costello, Jr. *Error control coding: fundamentals and applications.* Prentice-Hall, 1983.

[75] Songwu Lu and Vaduvur Bharghavan. Adaptive resource management algorithms in mobile computing environments. In *Proceedings of the SIGCOMM '96 Symposium on Communications Architectures and Protocols*, Stanford, August 1996. ACM.

[76] Songwu Lu, Vaduvur Bharghavan, and Rayadurgam Srikant. Fair scheduling in wireless packet networks. In *Proceedings of ACM SIGCOMM '97.* IEEE Computer Society, September 1997.

[77] Songwu Lu, Thyagarajan Nandagopal, and Vaduvur Bharghavan. A wireless fair service algorithm for packet cellular networks. In *Proceedings of The Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM '98)*, Dallas, TX, October 1998. ACM SIGMOBILE.

[78] Reiner Ludwig. *Eliminating Inefficient Cross-Layer Interactions in Wireless Networking.* PhD thesis, Aachen University of Technology, April 2000.

[79] Reiner Ludwig, Almudena Konrad, Anthony D. Joseph, and Randy H. Katz. Optimizing the end-to-end performance of reliable flows over wireless links. *ACM/Baltzer Wireless Networks Journal (Special issue: Selected papers from MOBICOM 99)*, 1999.

[80] Matthew Mathis and Jamshid Mahdavi. Forward Acknowledgment: Refining TCP congestion control. In *Proceedings of SIGCOMM '96*, August 1996.

[81] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the TCP Congestion Avoidance algorithm. *Computer Communications Review*, 27(3), July 1997.

[82] David J. Matsumoto. An evaluation of ELF over 802.11. Master's thesis, Carnegie Mellon University, 2002. (in preparation).

[83] Michael Meyer. TCP performance over GPRS. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC) '99*, pages 1248–1252, 1999.

[84] Paul V. Mockapetris. Domain names - concepts and facilities, November 1987. Internet RFC 1034.

[85] Paul V. Mockapetris. Domain names - implementation and specification, November 1987. Internet RFC 1034.

[86] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[87] Sanjiv Nanda, Richard Ejzak, and Bharat T. Doshi. A retransmission scheme for circuit-mode data on wireless links. *IEEE Journal on Selected Areas in Communications*, pages 1338–1352, Oct 1994.

[88] Thyagarajan Nandagopal, Songwu Lu, and Vaduvur Bharghavan. A unified architecture for the design and evaluation of wireless fair queueing algorithms. In *Proceedings of The Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM '99)*, 1999.

[89] T. S. Eugene Ng, Ion Stoica, and Hui Zhang. Packet fair queueing algorithms for wireless networks with location-dependent errors. In *Proceedings of INFOCOMM '98*. IEEE Communication Society, 1998.

[90] G. T. Nguyen, R. H. Katz, B. Noble, and M. Satyanarayanan. A trace-based approach for modelling wireless channel behavior. In *Proceedings of the Winter Simulation Conference*, 1996.

[91] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS Field) in the IPv4 and IPv6 headers, December 1998. Internet RFC 2474.

[92] B. Noble, G. Nguyen, M. Satyanarayanan, and R. Katz. Mobile network tracing. Internet RFC 2041, September 1996.

[93] Brian D. Noble, M. Satyanarayanan, Giao T. Nguyen, and Randy H. Katz. Trace-based mobile network emulation. In *Proceedings of the ACM SIGCOMM Conference*, Cannes, France, September 1997.

[94] Joseph P. Odenwalder. Error control coding handbook (final report). Technical report, LINKABIT Corporation, San Diego, CA, July 1976.

[95] Seong-Jun Oh, Tava Olsen, and Kimberly Wasserman. Distributed power control and spreading gain allocation in CDMA data networks. In *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[96] Christina Parsa and J. J. Garcia-Luna-Aceves. Improving TCP performance over wireless networks at the link layer. *ACM Mobile Networks and Applications*, 5(1):57–71, March 2000.

[97] Vern Paxson and Mark Allman. Computing TCP's retransmission timer, November 2000. Internet RFC 2988.

[98] Kostas Pentikousis. Tcp in wired-cum-wireless environments. *IEEE Communications Surveys & Tutorials*, 3(4), 2000.

[99] R. L. Pickholtz, D. L. Schilling, and L. B. Milstein. Theory of spread spectrum communications–a tutorial. *IEEE Transactions On Communications*, 30(5):855–884, May 1982. Available in Abramson, ed.

[100] J. Porter and A. Hopper. An overview of the ORL wireless ATM system. In *Proceedings of the IEEE ATM Workshop*. IEEE, September 1995.

[101] Jon Postel. Internet Protocol, September 1981. Internet RFC 791.

[102] Jon Postel. Transmission Control Protocol, September 1981. Internet RFC 793.

[103] Broadband Radio Access Networks (BRAN) Project. High Performance Radio Local Area Network (HIPERLAN) Type 1 Functional Specification. Technical Report EN 300 652 Ver. 1.2.1, European Telecommunications Standards Institute, July 1998.

[104] Broadband Radio Access Networks (BRAN) Project. HIPERLAN Type 2 System Overview. Technical Report TR 101 683 Ver. 1.1.1, European Telecommunications Standards Institute, February 2000.

[105] Proxim, Inc. RangeLAN2 tech guide. http://www.proxim.com/support/techtips/techgd.shtml.

[106] Qualcomm, Inc. Proposed EIA/TIA interim standard: Wideband spread spectrum digital cellular system—dual-mode mobile station–base station compatibility standard. Technical Report 80-7814 Rev DCR 03567, Qualcomm Inc., San Diego, California, April 1992. Accepted as TIA IS-95.

[107] Qualcomm, Inc. Q1650 k=7 multi-code rate viterbi decoder technical data sheet. 6455 Lusk Boulevard, San Diego, CA 92121-2779, 1992.

[108] Qualcomm, Inc. Master selection guide. 6455 Lusk Boulevard, San Diego, CA 92121-2779, 1994.

[109] K. Raith, E. Lissakers, J. Uddenfeldt, and J. Swerup. Cellular for personal communications. In *Wireless Personal Communications*, pages 1–20. Kluwer, 1993.

[110] K.K. Ramakrishnan, Sally Floyd, and David L. Black. The addition of explicit congestion notification (ECN) to IP, September 2001. Internet RFC 3168.

[111] Parameswaran Ramanathan and Prathima Agrawal. Adapting Packet Fair Queueing Algorithms to Wireless Networks. In *Proceedings of The Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM '98)*, Dallas, TX, October 1998. ACM SIGMOBILE.

[112] T. Rappaport. Characterization of UHF multipath radio channels in factory buildings. *IEEE Transactions On Antennas And Propagation*, pages 1058–1069, August 1989.

[113] Theodore S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, 1996.

[114] B. Rathke, M. Schlaeger, and A. Wolisz. Systematic measurement of tcp performance over wireless LANs. Technical Report TKN-98-001, Telecommunication Networks Group, Technische Universität Berlin, December 1998.

[115] D. Raychaudhuri, L.J. French, R.J. Siracusa, S.K. Biswas, R. Yuan, P. Narasimhan, and C.A. Johnston. WATMnet: A prototype wireless ATM system for multimedia personal communication. *IEEE Journal on Selected Areas in Communication*, 15(1):83–95, January 1997.

[116] D. Raychaudhuri and N. D. Wilson. ATM-based transport architecture for multiservices wireless personal communication networks. *IEEE Journal On Selected Areas In Communications*, 12(8):1401–1414, October 1994.

[117] M.S. Ryan and G.R. Nudd. The Viterbi algorithm. Technical Report RR–238, Department of Computer Science, University of Warwick, Coventry, CV4 7AL, United Kingdom, February 1993.

[118] A. Salmasi and K.S. Gilhousen. On the system design aspects of code division multiple access (CDMA) applied to digital cellular and personal communication networks. In *Proceedings of the 41st IEEE Vehicular Technology Conference*, pages 57–62. IEEE, May 1991.

[119] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–289, November 1984.

[120] Nachum Shachman and Jil Westcott. Future directions in packet radio architecture and protocols. *Proceedings of the IEEE*, pages 83–99, Jan 1987.

[121] S. Sheng, A. Chandrakasan, and R. Brodersen. A portable multimedia terminal. *IEEE Communications Magazine*, pages 64–75, December 1992.

[122] Prasun Sinha, Narayanan Venkitaraman, Raghupathy Sivakumar, and Vaduvur Bharghavan. WTCP: a reliable transport protocol for wireless wide-area networks. In *Proceedings of ACM MOBICOM '99*, Seattle, WA, August 1999.

[123] Michael Stangel and Vaduvur Bharghavan. Improving TCP performance in mobile computing environments. In *International Conference on Communications '98, Atlanta, GA*, 1998.

[124] Ion Stoica and Hui Zhang. A hierarchical fair service curve algorithm for link-sharing, real-time and priority service. In *Proceedings of SIGCOMM '97*, 1997.

[125] Eduardo Takahashi, Peter Steenkiste, Jun Gao, and Allan Fisher. A Programming Interface for Network Resource Management. In *1999 IEEE Open Architectures and Network Programming (OPENARCH'99)*, pages 34–44, New York, March 1999. IEEE.

[126] Fouad A. Tobagi, Richard Binder, and Barry Leiner. Packet radio and satellite networks. *IEEE Communications*, pages 24–40, November 1984. Available in Abramson, ed.

[127] Bruce Tuch. Development of WaveLAN, an ISM band wireless LAN. *AT&T Technical Journal*, pages 27–37, July/August 1993.

[128] M. Umehira, M. Nakura, H. Sato, and A. Hashimoto. ATM wireless access for mobile multimedia: concept and architecture. *IEEE Personal Communications*, pages 39–48, October 1996.

[129] Nitin H. Vaidya, Miten Mehta, Charles Perkins, and Gabriel Montenegro. Delayed duplicate acknowledgements: A TCP-unaware approach to improve performance of TCP over wireless. Technical Report 99-003, Computer Science Dept., Texas A&M University, February 1999.

[130] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2), April 1967.

[131] Andrew J. Viterbi and James K. Omura. *Principles of digital communication and coding*. McGraw-Hill, 1979.

[132] S. Y. Wang and H. T. Kung. Use of TCP decoupling in improving TCP performance over wireless networks. *Wireless Networks*, 7:221 – 236, 2001.

[133] M. Weiser. Some computer science issues in ubiquitous computing. *Communictions of the ACM*, 36(7):75–84, July 1993.

[134] Yang Richard Yang, Min Sik Kim, and Simon S. Lam. Transient behaviors of TCP-friendly congestion control protocols. In *Proceedings of INFOCOMM 2001*. IEEE Communication Society, 2001.

[135] Raj Yavatkar and Namrata Bhagawat. Improving end-to-end performance of TCP over mobile internetworks. In *Mobile '94 Workshop on Mobile Computing Systems and Applications*, December 1994.

[136] Bruce Zenel and Dan Duchamp. General purpose proxies: Solved and unsolved problems. In *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*. IEEE, May 1997.