Improving Wireless LAN Performance via Adaptive Local Error Control

David A. Eckhardt and Peter Steenkiste
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
davide+@cs.cmu.edu, prs+@cs.cmu.edu

Abstract

Wireless links can exhibit high error rates due to attenuation, fading, or interfering active radiation sources. To make matters worse, error rates can be highly variable due to changes in the wireless environment. Researchers and developers have explored a wide range of solutions to optimize communication in this difficult error environment, including traditional end-to-end solutions, link-layer solutions, and solutions involving layer four processing inside the network. A significant challenge is ensuring that systems with multiple layers of error control avoid compromising performance by duplication of effort.

We argue and demonstrate that protocol-independent link-level local error control can achieve high communication efficiency even in a highly variable error environment, that adaptation is important to achieve this efficiency, and that inter-layer coexistence is achievable.

The Logical Link Control layer of our WaveLAN-based experimental LAN includes three error control mechanisms: local retransmission, adaptive packet shrinking, and adaptive error coding. Measurements generated on a variety of network topologies and trace-based error environments demonstrate excellent TCP performance improvements and good coexistence with TCP's end-to-end retransmission strategy.

1. Introduction

Wireless transmissions are subject to interference from outside sources, absorption, scattering, fading, and intersymbol interference. This can result in very high error rates. Moreover, since conditions change over time (due to mobility or intermittent interference sources), the error environment will also change. Such a variable, high error environment can create problems for transport protocols and applications. The networking community has explored a broad spectrum of solutions to deal with wireless error environments. They range from "local" solutions that decrease the link error rate observed by upper layer protocols or applications to transport protocol modifications and proxies inside the network that modify the behavior of the higher level protocols [3, 2, 29, 30, 9].

In this paper we focus on the problem of improving communication in a wireless LAN. We show that purely local error control, which does not require wired hosts to modify their behavior or IP routers to understand transport or application protocols, can be an effective approach to dealing with the dynamic error environment of a wireless LAN. It allows users to obtain a high percentage of the available link bandwidth, even when using standard protocols such as TCP in harsh error conditions. The key is to have local error control be aggressive so it maximizes throughput, adaptive so error control overhead is paid only when needed, and designed in such a way that it does not interfere with higher layer protocols such as TCP. Our conclusions are based on measurements of a wireless LAN that incorporates adaptive local error control mechanisms. We also identify under which conditions our results apply and discuss when more complex solutions may be needed.

The remainder of this paper is organized as follows. In Section 2 we discuss the tradeoffs between local and end-to-end error control and consider local error control design issues. A description of our Medium Access Control (MAC) and Logical Link Control (LLC) protocols is found in Section 3. We describe our approach, based on synthetic and trace-based traffic loads, in Section 4 and present our synthetic evaluation in Section 5. Section 6 describes packet-shrinking and error coding extensions to the LLC layer, which are subjected to trace-based evaluation in Section 7. We discuss related work in Section 8 and summarize in Section 9.

2. Local error control

In this section we first examine the tradeoffs between local and end-to-end error control for wireless LANs and then discuss design options for local error control mechanisms.

2.1. Local versus end-to-end error control

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

First, entities directly connected by a link are the most likely to understand 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.

Second, local error control may be significantly more efficient than end-to-end error control. 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.

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, trying to do too much locally can lead to its own problems [27]. 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.

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.

There are several flavors of purely local solutions. The lowest level solution is hardware error control techniques such as adaptive codecs and multi-rate modems [20, 26]. While these are attractive in terms of simplicity, they may leave a noticeable residual error rate. In addition, while they reduce the average error rate, they can not typically differentiate between packets belonging to different flows. For example, in a shared-channel LAN different mobile stations may experience widely varying error conditions (due to, e.g., different locations [10]); tracking the appropriate coding level for different stations would be challenging with a transparent, low-level hardware approach.

An alternative is "pure" link-layer approaches such as IEEE 802.11 [15], MACA [17], and MACAW [5] that 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" (rather than protocol-aware) by tailoring the level of error control to the nature of the flow (e.g., bounding retransmission for packets with a limited lifetime). Flow information could be obtained through protocols such as RSVP [6] or the IP type-of-service header bits [12].

"Protocol-aware" link-layer protocols [3] 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 [2, 7, 3, 29] which perform significant and stateful protocol translation, or even data transcoding [30, 14], 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 paper 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 [15], MACA [17]), 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 3) and apply synthetic burst loss patterns designed to provide particular stresses (Section 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 6) and WaveLAN-specific error traces (Section 7).

3. MAC and LLC

We describe the design and implementation of our media access control (MAC) protocol, extensions for retransmission-only error control, and briefly characterize performance of our experimental system.

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. Two types of transactions are relevant to this paper. 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. The POLL-DATA message is sent by the master; it is used to transfer data to the slave and to invite the slave to send data. The slave responds with an DATA-ACK message that acknowledges the master's packet and can return data, if requested.

The operation of the MAC is similar to the IEEE 802.11 Point Control Function [15]. Clearly, many features could be added. For example, one could add contention periods, similar to the 802.11 Distributed Coordination Function (DCF).

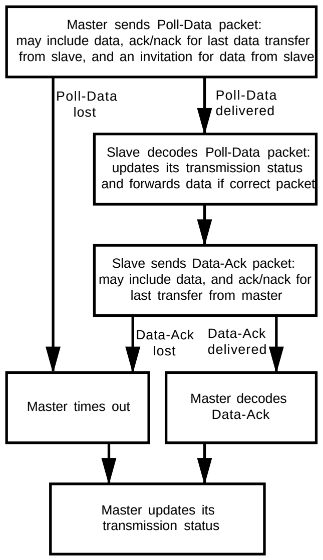
Master sends Poll-Data packet: may include data, ack/nack for last data transfer from slave, and an invitation for data from slave
Poll-Data lost
Poll-Data delivered
Slave decodes Poll-Data packet: updates its transmission status and forwards data if correct packet
Slave sends Data-Ack packet: may include data, and ack/nack for last transfer from master
Data-Ack lost
Data-Ack delivered
Master times out
Master decodes Data-Ack
Master updates its transmission status

Figure 1. MAC and LLC actions

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. First, both parties (master and slave) keep copies of each data packet until they receive an acknowledgement from the other party. Second, fields are added to the POLL-DATA and DATA-ACK headers to store acknowledgement information for data packets flowing in the opposite direction. The resulting actions executed by the MAC and LLC layers on both master and slave are shown in Figure 1.

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 Mb/s, which drops to about 1.8 Mb/s after considering the effects of headers and standard MAC overhead. A standard Ethernet-style WaveLAN driver achieves 1.4 Mb/s for a single TCP stream between two hosts, compared to .8 Mb/s when using our MAC protocol. This 43% loss in throughput

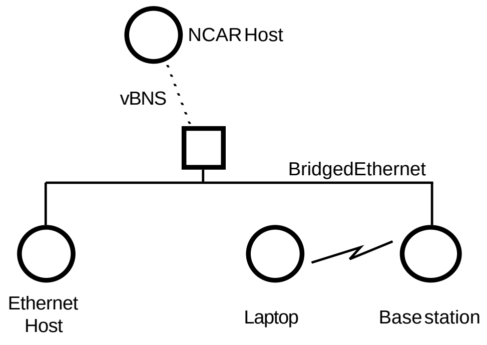

Figure 2. Hosts used in experiments

is a result of 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. Experimental approach

Our evaluations 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.

Experiments were run in both a LAN and WAN environment (see Figure 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.

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. The killer operates on a variable-length repeating list of packet error events, and applies appropriate errors (e.g., packet loss) to each packet as it is transmitted, received, or both, depending on the error trace. 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. The packet killer emulates packet loss on output by setting the Ethernet destination MAC address to a constant unused by any WaveLAN

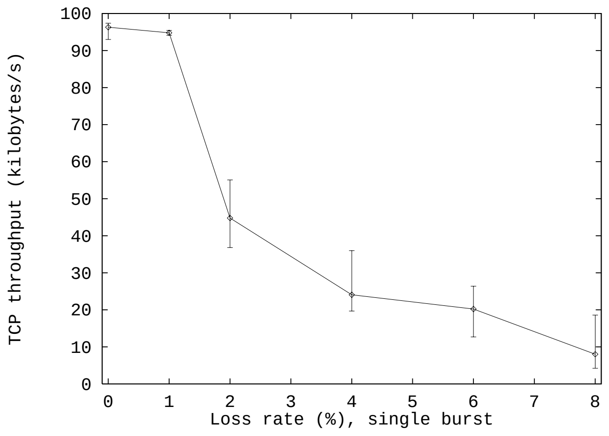

Figure 3. TCP versus data-packet loss in bursts of one to eight packets per hundred.

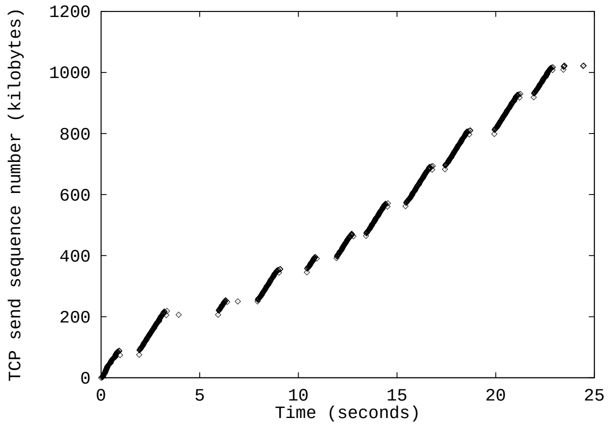

Figure 4. TCP encountering 2% packet loss in 2-packet bursts.

unit, so that appropriate air time is consumed. On input, a loss event is handled by the device driver's interrupt routine immediately discarding the packet.

5. Pattern-based evaluation

In this section, we will compare the performance of TCP with and without local error control. To understand interactions between local and end-to-end error control, we disabled packet shrinking and error coding, leaving only local retransmission, and employed simple synthetic packet loss patterns.

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 when local retransmission appears to be attractive.

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 3 shows the throughput of TCP without local

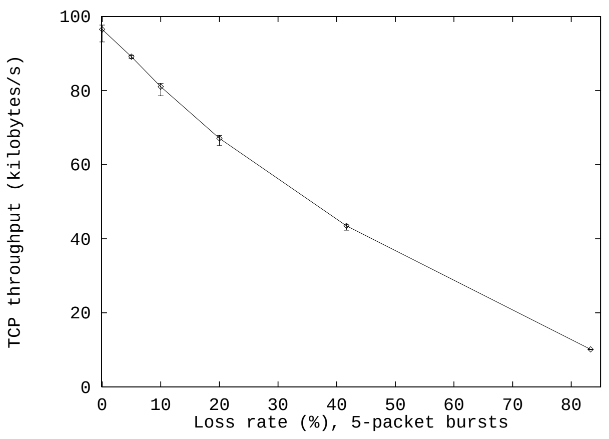

Figure 5. TCP with link-level retransmission, 0% to 83% loss.

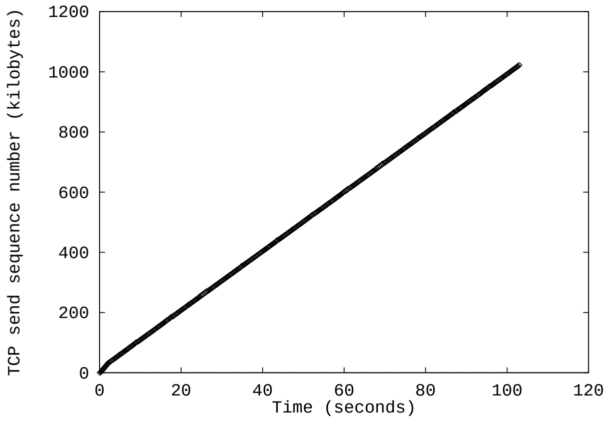

Figure 6. Trace of TCP with link-level retransmission, 83% loss.

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 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 [16, 13, 21] and performance drops by a factor of two. It continues to drop quickly as the burst size increases. Conventional wisdom, supported by recent work [22], 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 shows a TCPDUMP trace of a representative observation from the 2% loss (2-packet burst) case of Figure 3. We see that timeouts substantially degrade performance.

Figure 5 shows the throughput of TCP when we enable local retransmission. Five-packet bursts were dropped from

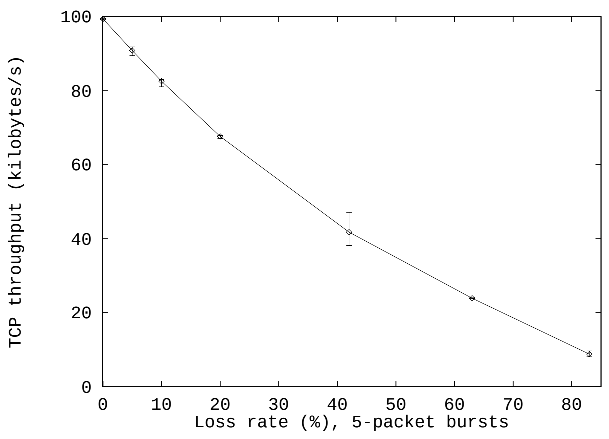

Figure 7. Effect of packet loss on TCP, Ethernet transmitter.

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 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 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 bandwidth of the wireless link.

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 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 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, 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

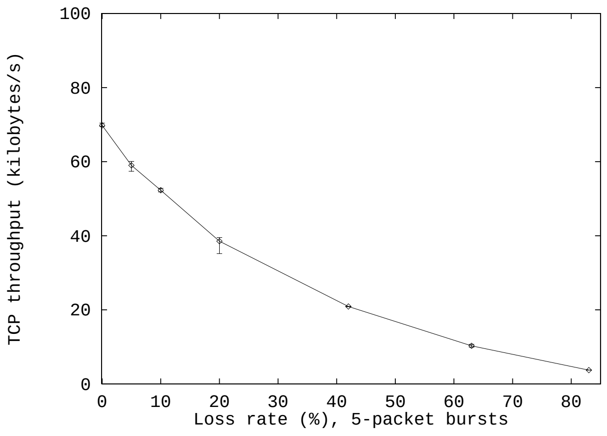

Figure 8. Effect of packet loss on TCP, distant transmitter.

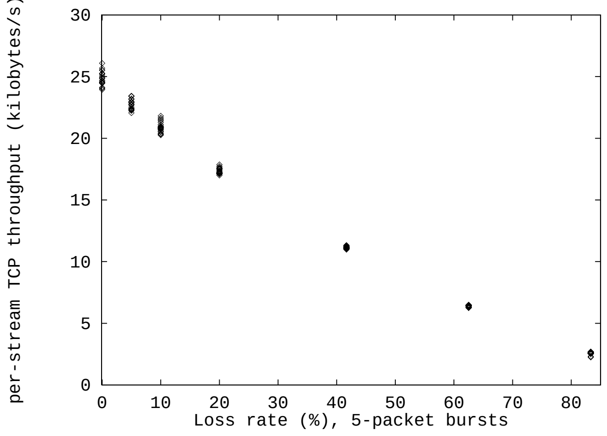

Figure 9. User throughput of four competing TCP streams.

high per-packet cost of our software MAC implementation (Section 3).

Figure 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.

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 with TCP.

5.3.1. 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 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 [3], 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.

5.3.2. Dynamic error environment

Beside 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.

5.3.3. 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.

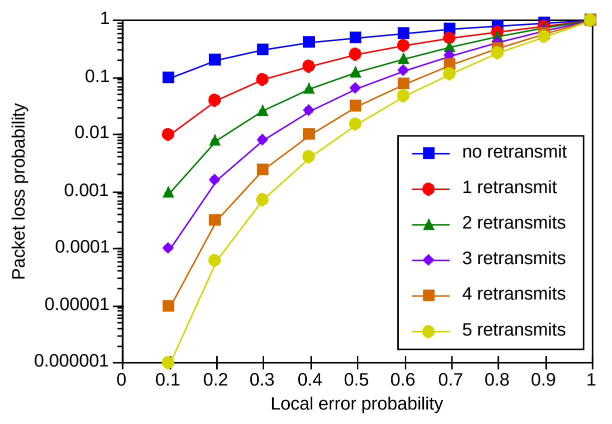

Figure 10. End-to-end packet loss rate as a function of local loss rate and maximum retransmit count

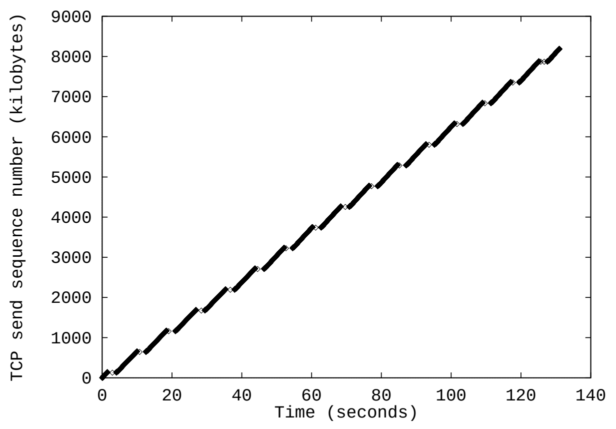

Figure 11. Trace of competing-retransmission scenario: 100-packet burst every 500 packet times.

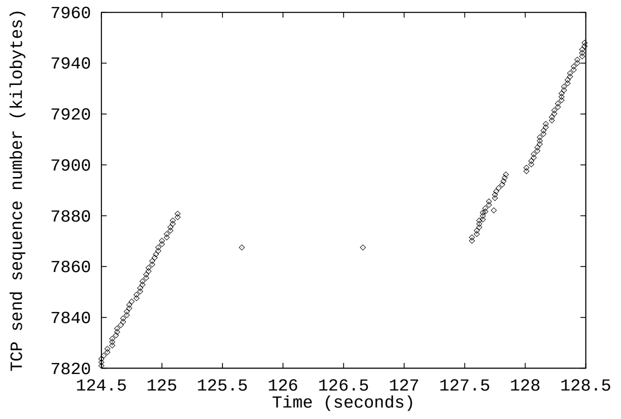

Figure 12. Zoomed-in trace of competing-retransmission scenario.

Figure 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 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 [9, 3]. 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 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 11 shows the packet trace for a single-hop TCP flow encountering this error pattern. The test program reported a throughput of 64 KB/s, about 80% of the link capacity available after packet loss (80 KB/s). The trace shows that between the error bursts TCP performs well. Figure 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%.

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, which implements retransmission using a periodic timer with a "slow timeout" interval, typically 500 milliseconds. 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 "ticks"), this relatively coarse timeout is supported by RFC 793 [25], which suggests a minimum value of 1 second for the retransmission timeout. Furthermore, preliminary work [24] suggests that long-haul Internet connections may require a minimum retransmission timeout of at least one second to avoid many false timeouts. As a result, long timeouts should be considered more a protocol feature than an implementation artifact.

It is also interesting to note that, regardless of the resolution of the timers, the values they are set to allow for some variation. For example, many systems set the retransmission timer to an estimate of the round-trip time plus four times the mean deviation from that estimate [16]. 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.

We conclude that retransmitting overly persistently, such as the perpetual retransmission implemented in our simple LLC, can hurt performance via inter-layer duplicate retransmissions. However, giving up after a few transmissions has the potential to seriously compromise TCP performance, while 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. While these initial results are promising, clearly further investigation would lead to better persistence heuristics for coexistence with TCP.

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 5.3.1 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 type-of-service marking [12] or RSVP-like protocols [6]. Different low-level error control policies could be implemented for packets belonging to flows of different types. For example, 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.

6. Adaptive local error control

We discuss the error environment for a particular network (WaveLAN) and present the design and implementation of more sophisticated adaptive local error control for such networks. We discuss its performance for both TCP and UDP streams in the next section, extending the pattern-based evaluation of the previous section to "real-world" situations.

6.1. LLC Design

An extensive study of the error characteristics of a wireless, in-building LAN is presented in [10]. The study showed that three classes of errors are common: packet loss, packet truncation, and bit corruption errors. While a simple local error control strategy would retransmit the packet in all of these cases, analysis shows that corrupted packets often have only a few bit errors and that truncation is rare for short packets. This suggests that packet shrinking and forward error correction (FEC) could be used to reduce the impact of packet truncation and bit corruption errors. However, given the high variability of the error environment, these techniques would need to be applied adaptively to avoid unnecessary overhead when conditions are good. A simulation-based evaluation [11] of adaptive packet shrinking and FEC shows that these techniques can significantly increase the useful throughput of a wireless LAN, using relatively simple adaptation policies. Given these results, our LLC combines adaptive FEC and packet shrinking with local retransmission.

6.2. LLC Implementation

Adaptive packet shrinking and FEC can be added fairly easily to the MAC and LLC implementations described in Section 3. Two changes are needed.

First, packet shrinking and FEC functionality must be added to the the packet send and receive functions. While it is possible to do end-to-end management of packet shrinking by adjusting the 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 (the IP and TCP headers are each typically 20 bytes). This would also limit adaptation by breaking a packet once into fixed-sized sub-packets, thus committing the link to fragmentation policy on a coarse grain. For these reasons, we implement packet shrinking through packet segmentation and reassembly over the wireless link. This requires segmentation and reassembly buffers on each host and some changes to the packet headers: every DATA transmission adds to the packet sequence number a starting byte offset, a byte count, and a "packet complete" bit, while acknowledgements consist of a packet sequence number and a cumulative length indicating correctly received bytes. With these changes, data transactions can be executed for variable-sized segments instead of complete IP packets. Since we lack hardware FEC support, our trace-based evaluation emulates the effect of FEC. The effectiveness of Reed-Solomon coding is evaluated off-line for each trace, and is applied at run time.

Second, the master tracks the quality of the wireless link, feeding its own observations and error reports that slaves include in DATA-ACK packets to an adaptation policy module, which determines an appropriate segment size and level of FEC. It shrinks and encodes its POLL-DATA packets accordingly, and includes the maximum segment size and minimal FEC level in its POLL-DATA packets, so slaves can format their packets appropriately.

6.3. Adaptation Policies

Given the mechanisms described above, a wide variety of adaptive error control policies can be implemented. We implemented and evaluated three fixed policies and four adaptive policies, similar to the ones presented in [11].

BOLD represents pure link-level local retransmission without coding or shrinking: maximally-sized packets (5 255-byte Reed-Solomon blocks) are sent with no error coding.

LIGHT transmits maximally-sized packets with 5% coding overhead. This is potentially a good policy since many packets are not badly damaged (as we will see in Table 1).

ROBUST attempts to excel in difficult conditions. It sends minimally-sized packets (1 255-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 poll-response transactions 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. Whenever two or more transactions 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 transactions 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.

A simulation-based evaluation [11] shows that the simple adaptive packet shrinking and FEC policies significantly increase the useful throughput of a wireless LAN, while the fixed policies are effective only in some environments. The key observation that allows adaptation is that, in a wireless LAN, significant changes in the error environment are often caused by human actions, e.g., movement of a laptop or turning on a cordless phone. However, the simulation necessarily makes several simplifying assumptions and does does not consider interactions with end-to-end protocols.

7. Trace-based evaluation of local error control

We present the results of operating the adaptive error control implementation described in the previous section in a variety of trace-based emulated error environments. By doing this, we hope to provide a more realistic evaluation of local error control, investigate the importance of *adaptive* coding and shrinking, and, finally, verify that retransmission, coding, and shrinking can work together to provide an environment suitable for TCP.

7.1. Error Traces

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 used PCMCIA WaveLAN[28] interfaces operating in the 902-928 MHz frequency band.

We instrumented our kernel device driver to collect all packets (even runt packets and those that failed Ethernet CRC) into an 8-megabyte kernel trace buffer. This allowed collection of approximately 30 seconds of continuous network activity. The traces were post-processed to yield, for each transmission, a timestamp, the transmitted and received packet lengths, signal information from the RF modem, a packet sequence number, and a list of corrupted bits.

In this paper we will use traces that capture five different error environments; their characteristics are summarized in Table 1. The “office” scenario represents a best-case, interference-free situation. In the “walking” scenario, interference is provided by a nearby cordless phone base station and a cordless phone handset moving around the room at walking speed. The “adjacent” case is similar but the handset is nearby and power-cycled roughly twice per second to produce a trace with challenging dynamicity. The “table” case is a less-challenging desktop situation, again with cordless phone interference. Finally, the “walls” case investigates attenuation due to distance, concrete block walls, and classroom furniture. As compared to the interference environments, attenuation has an almost insignificant truncation rate, but has significant packet corruption. More details on the traces and our trace methodology may be found in [11].

Finally, since the traces contain a richer set of events than that used in our pattern-based evaluation, we must expand the packet killer. On output or input, a truncation event causes the killer to prefix the packet with a special flag that includes the truncation length. The receiving machine’s LLC layer will notice the flag, extract the truncation length, and ignore the packet contents. This is conservative: we could have examined erasure correction or extracting partial packet contents. On output or input, a bit corruption event causes the killer to prefix the packet with a special flag indicating the minimum-strength Reed-Solomon code required to decode the packet. The receiving machine’s LLC layer will compare

Trial Name	Packets Sent	Bits Received	Packet Loss Rate	Non-lost Packet Truncation Rate	Non-truncated Packet Corruption Rate	Received-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 1. Summary of various error scenarios.

the killer event against the code the sender used to determine whether to signal a decoder failure.

7.2. Evaluation of adaptive strategies

To evaluate the performance of a policy, we measured the single-session TCP throughput across the link. In addition, 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. The policy performance results presented in Table 2 represent the averages of five 1-megabyte transfers, reported in kilobytes per second.

The performance of the static policies is fairly predictable. BOLD 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%: TCP is actually unable to complete its 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 "table" and "walls" cases.

7.3. Summary

The above results suggest that FEC coding can build on top of local retransmission to further increase usable bandwidth 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. Most importantly, Figure 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.

8. Related work

Both the IS-54 TDMA system [23] and the Qualcomm IS-95 CDMA [18] system provide link-level retransmissions for packet data over digital cellular phone networks. The AIRMAIL design [1] 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 [19] also suggests an architecture including link-level fragmentation and adaptive error coding. We believe that our pattern- and trace-based evaluations provide support for this style of link-level error control mechanisms, which are independent of specific transport protocols.

Several high-speed wireless LANs include both link-level retransmission and packet fragmentation. The IBM Wireless RF LAN [4] performs fixed-size fragmentation primarily to facilitate time scheduling and comply with frequency-hopping dwell time limits. The IEEE 802.11 wireless LAN standard [15] 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.

Though the MACA [17] and MACAW [5] 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

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

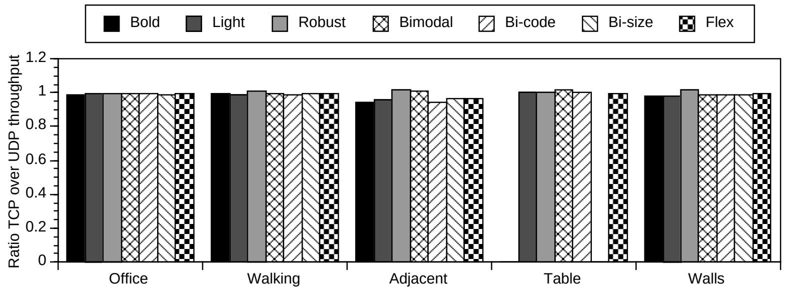

Figure 13. Comparison of TCP and UDP throughputs for the experimental scenarios. The similarity of these throughput figures indicates that TCP is not falsely assuming link congestion.

increased TCP throughput.

A comparison [3] among the Berkeley LL family of link-layer retransmission protocols demonstrated a noticeable performance enhancement obtained by filtering duplicate 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.

The behavior of TCP NewReno operating over wireless fading links without link-level error control, including the relationships between error burst patterns, fast recovery, and timeouts, is studied in [8]. Interactions between link-level and network-level error control are investigated in [31].

9. Conclusion

The primary contribution of this paper is a demonstration 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 type-of-service marking 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. We then employed a more sophisticated adaptive local error control implementation designed for WaveLAN hardware that includes local retransmission, adaptive packet shrinking, and FEC coding to address packet loss, truncation, and corruption, respectively. Simple adaptation policies outperformed static policies across a range of error environments while allowing effective end-to-end communication using both TCP and UDP.

Acknowledgements

This research was supported in part by the Defense Advanced Research Projects Agency/ITO monitored by NRaD under contract N66001-96-C-8528.

Jamshid Mahdavi and Matt Mathis of the Pittsburgh Supercomputer Center and Vern Paxson of the Lawrence Berkeley National Laboratory deepened our understanding of TCP retransmission timers. David Johnson and David Maltz gave us many useful comments. The WaveLAN driver we modified was written by Bob Baron of the Coda research project. The responsibility for any inaccuracies or deficiencies in this work is, of course, our own.

References

[1] E. Ayanoglu, S. Paul, T. F. LaPorta, K. K. Sabnani, and R. D. Gitlin. AIRMAIL: a link-layer protocol for wireless networks. *ACM/Baltzer Wireless Networks Journal*, pages 47–60, February 1995.

[2] 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.

[3] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, December 1997.

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

[5] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: A media access protocol for wireless LANs. In *ACM SIGCOMM '94*, pages 212–225, August 1994.

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

[7] R. Cáceres and L. 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.

[8] A. Chockalingam, M. Zorzi, and R. R. Rao. Performance of TCP on wireless fading links with memory. In *Proceedings of IEEE ICC '98*, June 1998.

[9] A. DeSimone, M. C. Chuah, and O.-C. Yue. Throughput performance of transport-layer protocols over wireless LANs. In *Proceedings of IEEE GLOBECOM 1993*, pages 542–549, December 1993.

[10] D. Eckhardt and P. 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.

[11] D. Eckhardt and P. Steenkiste. A trace-based evaluation of adaptive error correction for a wireless local area network. *Journal on Special Topics in Mobile Networking and Applications (MONET)*, 1998. To appear in forthcoming issue on Adaptive Mobile Networking and Computing.

[12] P. Ferguson. Simple Differential Services: IP TOS and Precedence, Delay Indication, and Drop Preference, March 1998. Internet draft draft-ferguson-delay-drop-02.tex.

[13] S. Floyd. TCP and successive fast retransmits, Feb. 1995. Obtain via ftp://ftp.ee.lbl.gov/papers/fastretrans.ps.

[14] A. Fox and E. 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.

[15] IEEE Local and Metropolitan Area Network Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, New York, 1997.

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

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

[18] P. 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.

[19] P. Lettieri and M. 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.

[20] S. Lin and D. J. Costello, Jr. *Error control coding: fundamentals and applications*. Prentice-Hall, 1983.

[21] M. Mathis and J. Mahdavi. Forward Acknowledgment: Refining TCP congestion control. In *Proceedings of SIGCOMM '96*, Aug. 1996.

[22] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP Congestion Avoidance algorithm. *Computer Communications Review*, 27(3), July 1997.

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

[24] V. Paxson. Personal communication.

[25] J. Postel. Transmission Control Protocol, September 1981. Internet RFC 793.

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

[27] J. Saltzer, D. Reed, and D. Clark. End-to-end arguments in system design. *ACM TOCS*, 2(4):277–289, November 1984.

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

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

[30] B. Zenel and D. Duchamp. General purpose proxies: Solved and unsolved problems. In *Proceedings of Hot-OS VI*, May 1997. http://www.mcl.cs.columbia.edu/∼baz/ps/hot-os-vi.ps.

[31] M. Zorzi and R. R. Rao. Error control in multi-layered stacks. In *Proceedings of GLOBECOM '97*, November 1997.