An Internet-style Approach to Wireless Link Errors

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 differ from traditional "wired" links in two ways that challenge the existing Internet. On wireless links packet loss or corruption due to transmission errors is not rare, which calls into question the standard Internet assumptions that transmission errors should be corrected by transport-level protocols at end systems and that end-to-end packet loss typically indicates network congestion. Also, the severity and location-dependent nature of these errors calls into question the meaning of "fair" scheduling, per-flow quality of service, and even looser notions such as service level agreements, when applied to wireless links. An important question is whether the two unique problems posed by wireless links can be successfully addressed within the standard Internet architecture, as opposed to requiring new transport protocols designed specifically for wireless links or requiring wireless links to "fix up" the operation of specific end-to-end protocols. We provide experimental evidence that a combination of protocol-blind link-level local error control, which lessens the damage, and error-sensitive link scheduling, which ensures sensible outcomes in response to link capacity loss, provides a good operating environment while adhering to traditional Internet design practices.

1 Introduction

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 and that end-to-end packet loss typically indicates network congestion. 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 WFQ) assume that packet transmission

This is an *internal* version of a paper that appeared in Volume 2, Issue 1 of *Wireless Communications and Mobile Computing* (JWCMC) in February of 2002. . .

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 Internet approach has been to standardize baseline, widely-usable mechanisms as opposed to policy. In particular, it 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. Nearly every link includes some form of error detection, such as per-character parity or a per-frame checksum, tuned to match the link's expected corruption patterns, and some links include link-specific error control mechanisms such as error coding or retransmission. In the other direction, congestion control, in-order delivery, application-visible framing, and retransmission policies (including timing and persistence) 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.

An important question is whether the two unique problems posed by wireless links can be successfully addressed within the standard Internet architecture, as opposed to requiring new transport protocols designed specifically for wireless links [31, 2, 8] or requiring wireless links to "fix up" the operation of specific end-to-end protocols [4]. Ideally, wireless link errors could be addressed locally by a mechanism that is useful to most applications and detrimental to few or none.

1.2 Solution Summary

We believe that it is possible to effectively address wireless link errors locally, through a combination of link-level error control, which lessens the damage, and error-sensitive link scheduling, which ensures sensible outcomes in response to the 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 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, existing end-to-end protocols can effectively use wireless links despite dynamic and severe error environments.

1.3 Evaluation overview

Throughout this paper we will support our claims with measurements. Our evaluation platform consists of Intel 80486 and Pentium laptops using PCMCIA WaveLAN I [29] interfaces operating in the 902-928 MHz frequency band. The machines run the NetBSD 1.2 kernel and all the measurements use the stock NetBSD TCP protocol stack implementation.

Since the WaveLAN hardware does not support link-level error control, we modified the kernel device driver to support a simple poll/response Medium Access Control (MAC) protocol, similar in spirit to the IEEE 802.11 Point Control Function [16]. Like other LAN MAC protocols, ours includes an immediate link-level acknowledgement of each packet. Our hardware has a nominal throughput of 2 Mb/s, and we obtain an effective throughput of approximately 0.8 Mb/s after factoring in link-level headers, hardware medium access overhead, and the fact that we implement both MAC and LLC in software. To obtain repeatable results, we obtained traces of packet losses, packet truncations, and bit inversions and modified our device driver to replay these traces in a running system. Since we are using an off-the-shelf NetBSD TCP implementation executing on real hardware, we can accurately measure how well TCP performs in a wide range of error environments.

1.4 Road Map

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. In Section 3 we use synthetic burst loss patterns to show how persistent link-level retransmission can preserve TCP throughput under harsh error conditions. Then in Section 4 we add more sophisticated error control techniques (forward error coding, packet size adjustment, and error control adaptation) and base our evaluation on error traces captured from a local area network.

Once we have demonstrated that adaptive local link-level error control allows end-to-end transport protocols to function effectively, we focus on meeting the throughput and stability needs of particular flows in the face of severe and location-dependent errors. In Section 5 we motivate the need for error-sensitive link scheduling and analyze detailed examples to form our proposed solu-

tion, *effort-limited fair* (ELF) scheduling. In Section 6 we describe the *power factor* administrative control which defines how a scheduler should respond to capacity loss. In Section 7 we evaluate the behavior of an ELF scheduler via trace-driven simulation.

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.

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 most able to understand and manage its particular characteristics. It is impractical for end-systems to decide which packet losses represent congestion versus intermittent link errors. Link stations can respond more quickly to changes in the error environment, and can employ error control techniques, such as soft-decision decoding and energy combining, which are difficult to implement end-to-end. Second, end-to-end error control necessarily involves multiple error-free links in solving a purely local problem: end-to-end retransmission demands time on every link, while local retransmission requires extra link time 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.

Despite these attractions, trying to do too much locally can lead to its own problems [11, 28]. First, local error recovery mechanisms may alter the characteristics of the network. 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. If a local recovery protocol and TCP retransmit the same packets, this can result in excessive link time consumption or even queue overflow. Finally, a given data packet may bear information with a limited useful lifetime. For example, it may be 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 approaches. Because different flows have different latency and reliability requirements, and need to communicate with different link stations, *hardware-only solutions* such as adaptive codecs and multi-rate modems are insufficient. *"Pure" link-layer approaches* such as IEEE 802.11 [16], MACA [18], and MACAW [5] 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 [7] or IP Differentiated Services marking [26]. *"Protocol-aware" link-layer protocols* [4] may inspect the packets they pass in order to give special treatment where it is most needed. While this can significantly improve performance, each hop must understand a wide variety of transport or even application protocols. The Internet currently carries a variety of incompatible streaming audio and video protocols, and this situation is likely to persist. The development of new protocols and the deployment of link-level protocol parsers could evolve into a counterproductive "arms race." *"Gateway-style" or "indirect" error control* performs significant and stateful protocol translation [31, 9, 3, 8, 4], or even data transcoding, between subnets with greatly differing characteristics. In addition to 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 the rapid deployment of protocols incorporating link-layer error control (e.g., 802.11 [16]): 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, although, as we discuss below, our results suggest that link-layer error control may benefit from awareness of flow-specific performance needs.

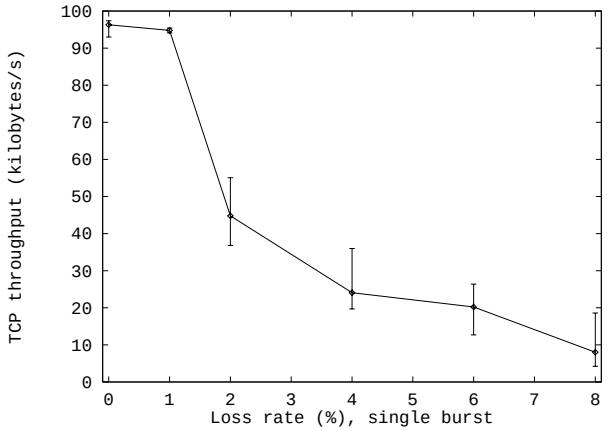

Figure 1: TCP versus data-packet loss in bursts of one to eight packets per hundred.

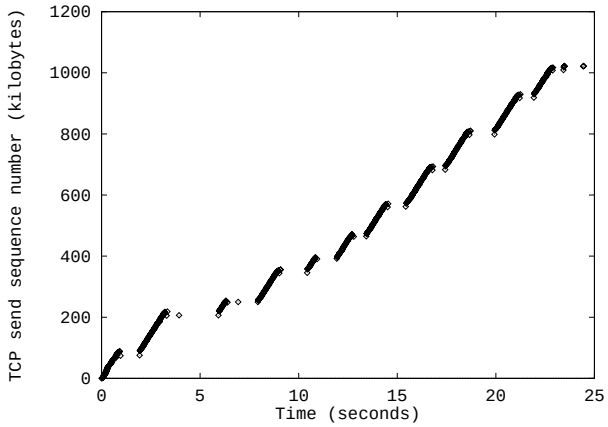

Figure 2: TCP encountering 2% packet loss in 2-packet bursts.

3 Pattern-based evaluation

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.

3.1 Basic robustness evaluation

We first focus on the simplest possible scenario: a single TCP connection between two wireless hosts. Figure 1 shows the throughput of TCP without local retransmission. As for all the results in this section, each data point summarizes the results of five 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 1 consists of a single burst of one to eight packets being dropped out of every hundred

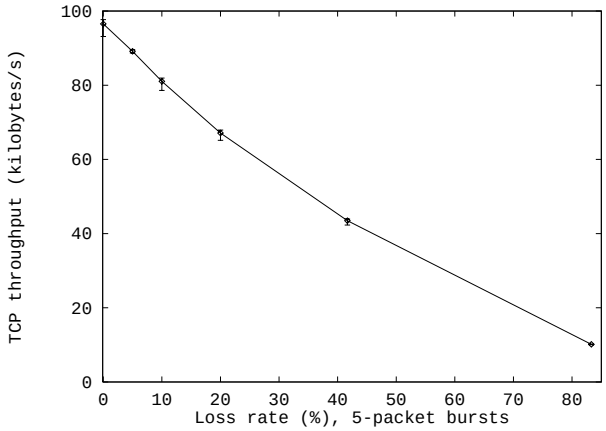

Figure 3: TCP with link-level retransmission, 0% to 83% loss.

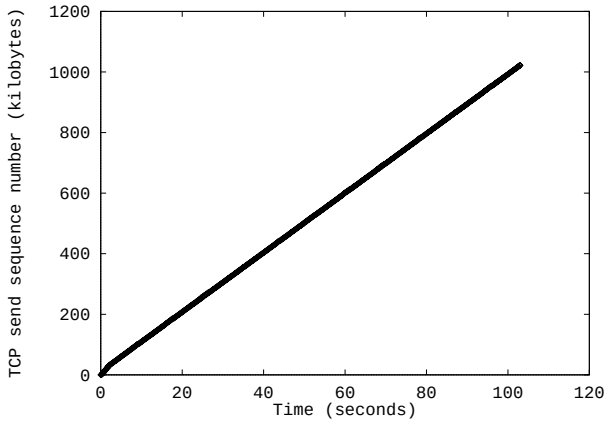

Figure 4: Trace of TCP with link-level retransmission, 83% loss.

packets, resulting in error rates between 1% and 8%. We see that performance degrades quite quickly. TCP handles single packet drops well, but when multiple packets per window are dropped TCP congestion avoidance and timeouts are triggered [17, 15] and performance drops by a factor of two. It continues to drop quickly as the burst size increases. Since burst losses are particularly challenging, this matches the conventional wisdom, supported by recent work [23], that TCP Reno can handle packet loss rates of up to 1-2%. Figure 2 shows a TCPDUMP trace of a representative observation from the 2% loss (2-packet burst) case of Figure 1. We see that timeouts substantially degrade performance.

Figure 3 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 1: 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 shows a TCP trace from the 83% loss case. The reason that this works well is straightforward. 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.

3.2 Analysis

In this section we analyze why local retransmission is so effective. An examination of why it works during a period of fixed error rate will suggest necessary features of successful local error control, and we will explain how local retransmission makes variable error rates tolerable by TCP's end-to-end retransmission strategy. Finally, we will examine the degree to which retransmission must be persistent in order to be effective.

3.2.1 Steady state conditions

When we consider the case of a link with a fixed error rate, local error control must meet several requirements for TCP to exhibit stable behavior. Since TCP interprets packet loss as a sign of congestion (Figure 1), local error control should be persistent enough that lost packets almost always indicate congestion. Packet reordering causes the receiver to generate 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 (an alternative approach, suppressing duplicate acknowledgements [4], requires special-case router code for each supported transport protocol). Because TCP estimates the round-trip time and uses this estimate to set its retransmission timers, local error control needs to be fast enough to make multiple retransmission attempts before TCP times out. Otherwise TCP will observe wildly fluctuating round-trip times which could cause excessive retransmissions or excessively long backoffs.

3.2.2 Dynamic error environment

We also must consider what happens when 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 queue will drain more slowly and will eventually overflow, causing packets to be lost. TCP will back off and retransmit the lost packets. Since we have a congestion condition, this is exactly the right response.

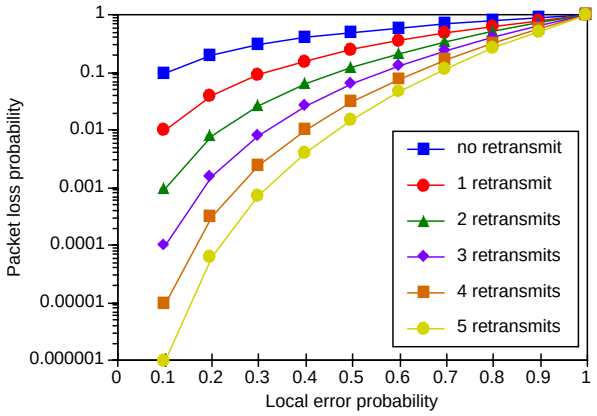

Figure 5: End-to-end packet loss rate as a function of local loss rate and maximum retransmit count

3.2.3 Persistence of local error control

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 5 shows how the end-to-end packet loss rate changes as a function of the local packet error rate and the maximum retransmit count, assuming steady state conditions and random errors (bursty errors increase the end-to-end loss rate). Not surprisingly, the results show that, for high error environments, retransmission may need to be very persistent to keep the packet loss rate below 1%. As we discussed in Section 3.1, 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 interfere with TCP retransmission [11, 4]. To better understand this interaction, 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 6 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 error bursts TCP performs well. Figure 7 shows in more detail what happens during an error burst. We observe two types of redundant packet: "probe" packets used by the sender to re-establish contact, and a sequence of packets

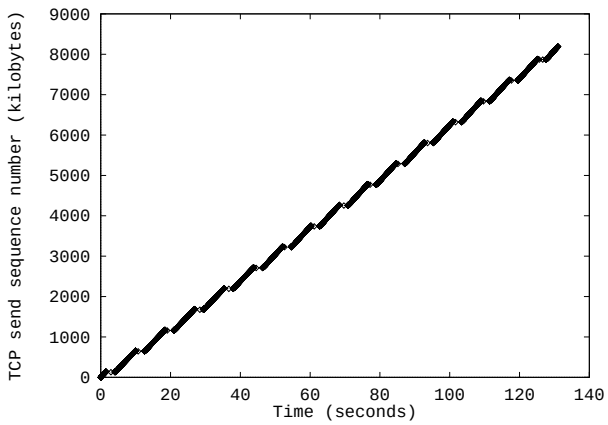

Figure 6: Trace of competing-retransmission scenario: 100-packet burst every 500 packet times.

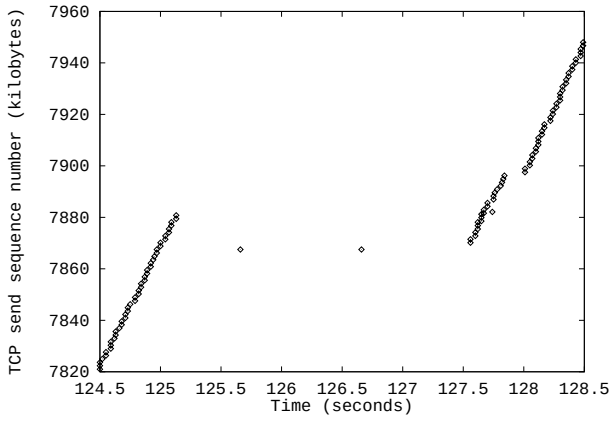

Figure 7: Zoomed-in trace of competing-retransmission scenario.

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 positive result depends on end-to-end retransmission timeouts being substantially longer than the per-hop round-trip time. TCP retransmission timers are typically more conservative in the presence of round-trip-time variability [17] and TCP traces suggest that long-haul Internet connections avoid many false timeouts by maintaining a minimum retransmission timeout of at least one second [1], so a LAN link will typically have time for multiple local retransmissions before the end systems time out.

3.3 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.

While we have encouraging results for TCP, it seems clear that other applications, such as streaming video, could require different low-level retransmission policies. Luckily, the Internet seems to be evolving toward making flow-type information available to link-layer elements via Differentiated Services marking [26] or RSVP-like protocols [7]. Different low-level error control policies could be implemented for different flow 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 Trace-based evaluation

In order to examine how link-level error control enables TCP to perform in a more real-world situation, we will employ bit-level traces of errors due to interference and attenuation while adding error coding and adaptive packet sizing to the simple retransmission scheme we evaluated in the previous section. Since the traces include not only packet losses but also bit corruptions and packet truncations, some intelligence is required in order to predict which combination of packet size and error coding will allow each transmission to succeed most efficiently. Therefore, in this section we will employ various error control policies and error environments. We will observe that adaptive link-level error control extracts significant throughput even in the face of harsh errors and that the resulting throughput is usable by TCP.

Trial Name	Error Source	Packet Loss	Packet Truncation	Packet Corruption	Bit Error Rate
Office	none	none	none	trivial	0
Walking	moving phone	trivial	trivial	5%	1.6×10^{-4}
Adjacent	on/off phone	31%	23%	trivial	1×10^{-3}
Table	nearby phone	5%	2%	94%	4.9×10^{-3}
Walls	walls, distance	1%	2%	27%	3.8×10^{-4}

Table 1: Summary of various error scenarios.

The error traces are summarized in Table 1. Column headings are cumulative, e.g., "packet corruption" is the percentage of non-lost, non-truncated packets which were corrupted. While the "office" trace is essentially error-free, standard TCP without link-level error control would be seriously challenged by the "walking" trace and essentially unable to operate in environments represented by the other three traces. We will observe the unmodified NetBSD kernel TCP im-

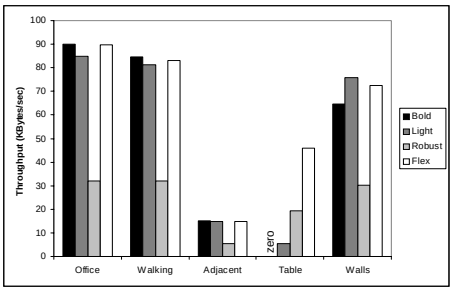

Figure 8: TCP throughput as a function of error trace and error control policy.

plementation running over four different error-control policies in each of the five emulated error environments. 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. ROBUST attempts to excel in difficult conditions. It sends minimally-sized packets (1 255-byte block) with 29% devoted to coding overhead. FLEX adapts the packet size and degree of FEC redundancy independently, varying packet size from 1 to 5 blocks and error coding overhead from 0% to 78% according to simple heuristics. More details of the error patterns exhibited by the traces and the LLC mechanisms we use in response are available in [12] and [13].

To evaluate the performance of a policy, we measured single-flow TCP throughput across the link. Then we measured the throughput obtained by a program that sprays UDP packets as fast as possible, ignoring queue overflows and packet losses, and reports only the number of packets received. The policy performance results presented in Figure 8 represent the averages of five 1-megabyte transfers, reported in kilobytes per second.

The results indicate that link-level error control recovers significant throughput in even harsh error conditions and that adaptation helps significantly. The noticeable and widely varying bit error rate would be difficult to address on an end-to-end basis but is easily overcome at the link level. Each static error control policy performs well in some scenarios but poorly in others. For example, BOLD does very well in low-error situations because it doesn't waste any overhead on short packets or error coding overhead, but in the "table" case, which has a packet corruption rate of 94%, it is unable to complete TCP's three-way handshake, let alone transfer any data. The FLEX policy, which can independently adjust packet size and error coding, is clearly the best policy overall. A more detailed discussion can be found in [12].

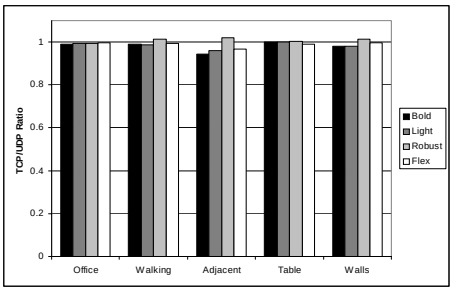

Figure 9: Comparison of TCP and UDP throughputs for the experimental scenarios. The similarity of these throughput figures indicates that TCP is not falsely assuming link congestion.

Figure 9, which compares TCP and UDP throughput in each situation, demonstrates that the throughput recovered by link-level error control, measured by UDP, is fully usable by off-the-shelf TCP under every combination of error environment and adaptation policy. That is, none of the error control strategies is causing TCP to lose performance due to false triggering of its congestion avoidance mechanism. While the link-level error control strategies are diverse, all follow the rules outlined in Section 3.2.1.

5 Error-sensitive link scheduling

Though researchers have devised many ways of reducing the impact of errors in wireless networks, none of these solutions can completely eliminate their effect. Techniques such as adaptive link-level error correction (Sections 2-4, [12, 13, 19, 22]) and swapping transmission slots based on error prediction [20, 25] can reduce the capacity loss due to errors but cannot eliminate it. For example, retransmission can deliver lost packets, but each failure irrevocably consumes air time. The air time costs of error coding and packet shrinking are fractional packets rather than multiple packets, but these defensive measures still reduce link throughput. In short, *capacity loss is a fundamental property of wireless networks.*

Below we will use example scenarios to argue that severe and location-dependent link errors call for wireless schedulers that explicitly allocate link capacity loss to flows according to their tolerance for throughput variation. To illustrate the impact of variable link capacity we will use an example wireless cell with a capacity of 800 kilobits per second allocated by a centralized packet scheduler of the Weighted Fair Queueing (WFQ) family [10]: link time is distributed to flows according to a set of weights; throughput reservations can be supported by adjusting weights as flows arrive and leave. Our scenarios contain two reserved flows and two best-effort flows, with flow properties and weights summarized in Table 2. We will use the term *fidelity* to denote the degree to which a scheduler

is "faithful" to a particular flow, i.e., the ratio of the flow's observed throughput to throughput expected if the link were error-free. In the table, results marked with "$\surd$" represent flows meeting their needs; "$\otimes$" represents un-met needs; "$\approx$" represents flows without specific requirements.

Table 2: Possible results of a 50% packet errors (rates in kbit/sec)

Client	Target rate	WFQ Weight	Expected rate	Effort-fair rate	Preferable rate
Audio	reserved 8	1.0	8	4 $\otimes$	8 $\surd$
Video	reserved 350	44.0	350	175 $\otimes$	350 $\surd$
FTP1	available	27.5	221	110 $\approx$	21 $\approx$
FTP2	available	27.5	221	110 $\approx$	21 $\approx$

5.1 Pervasive errors

Let us first consider what happens if 50% of all packets are lost. A scheduler oblivious to capacity loss will assign transmission slots to flows according to their weights. With this *effort-fair* approach, each flow receives 100% of its expected *effort* (air time), but errors degrade its performance to only 50% of its expected *outcome* (Table 2, column 5). The result is that both rate-sensitive flows fail to achieve their desired rates, even though there is enough remaining link capacity. An alternative is to use a priority-based scheduler that understands the effects of capacity loss and gives extra effort to the audio and video flows so they achieve their desired outcomes (column 6). While this results in high fidelity for this scenario, it is too simplistic to be practical. For example, with a 50% error rate, the priority-based scheduler would allocate 89% of the useful bandwidth to the reserved flows; if the error rate exceeded 55%, the FTP flows would get no air time even though this futile sacrifice would not enable the reserved flows to meet their reservations. A more reasonable solution would be to have the audio flow meet its reservation and the other flows share the remaining bandwidth. In this high-error situation, even the error-blind effort-fair scheduler would be preferable, since it wouldn't artificially starve flows.

We conclude that, while both the effort-fair WFQ scheduler and the priority scheduler may yield the desired effect in some cases, they fail in others. The key issue is how much help, in the form of extra effort, we should give flows in pursuit of fidelity. While some help can be very beneficial, too much help creates excessive pain for other flows.

5.2 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/second) and one FTP flow (best-effort).

Table 3: Location-dependent errors (rates in kbit/sec)

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

The results shown in Table 3 confirm our earlier observations. With an effort-based WFQ scheduler, the reserved flow that encounters errors does not meet its reservation because the capacity loss is distributed according to the unequal environmental effects. A priority-based scheduler solves this problem by giving the second video flow more air time to overcome the errors, but priority does not address the issue of fairness between the two FTP flows. The last column shows a more desirable outcome: both video flows meet their needs and the best-effort FTP flows experience *outcome fairness*, i.e., equal throughput despite unequal error rates.

It is easy to see that outcome fairness, like priority, can lead to starvation when error rates are high. But even in non-pathological cases there is cause for concern. As is shown in the last line of Table 3, giving extra air time to high-error stations at the expense of error-free or low-error stations reduces the overall efficiency of the link. This effect becomes more significant as the difference in error rates between stations increases. Clearly network managers will want to limit the effect of error recovery on network efficiency.

5.3 Summary

We showed that a traditional WFQ scheduler that is oblivious to errors does not ensure either throughput reservations or best-effort fairness. An error-sensitive scheduler can use inter-class priority to maintain throughput for reserved flows and equal-fidelity scheduling to maintain outcome fairness among best-effort flows. These mechanisms compensate flows for link errors by giving them extra air time for error recovery (e.g., retransmission or error coding). However, we also showed

how both priority and outcome fairness lead to futile starvation of other flows when error rates are very high. We must find a way to balance desired outcomes against excessive link inefficiency. This can be done by adding effort limits to an outcome-fair scheduler, which we will describe in the next section.

6 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.

6.1 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 10).

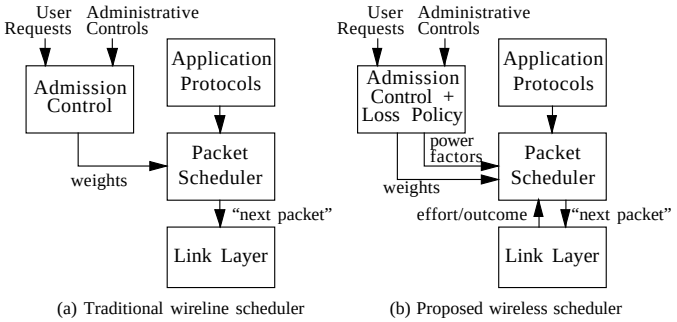

Figure 10: Scheduler models

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*.

6.2 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

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

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.

6.3 Fixed-rate reservations

Providing absolute bandwidth reservations (as opposed to link shares), which WFQ can do, requires additional support in ELF. The reason is that a error-adjusted fraction of a deflated link will be smaller than the expected fraction of the error-free link. We will obtain absolute bandwidth reservations by both adjusting the weights of guaranteed flows upward as described above and 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

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

which ensures that the best-effort aggregate flow will consume only whatever link time is left over after every best-effort 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 6.2.

6.4 Example

This scheduler can support a variety of policies. For example, the hybrid outcome/effort scheduler described in Table 3 can be implemented by setting the power factors of each flow to at least 200%. 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.

6.5 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. Since the vast majority of existing Internet traffic is best-effort, it is important to handle this case well. Positive 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 cause only a mild degradation in throughput, which will be shared among all the best-effort flows. This is much more attractive than allowing each link-level error burst to stall one or more end-to-end flows.

For flows that signal a need for a particular throughput (via, e.g., RSVP), existing wireline admission control modules can be adapted 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 8Kb/s or 64Kb/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.

6.6 Discussion

The proposed "power factor" scheduler model meets the requirements outlined in Section 5. 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 keep link efficiency over 50% as long as each flow's error rate is under 50%.

7 Scheduler simulation

In this section we will present simulation results for a particular ELF scheduler based on weighted round-robin (WRR). This scheduler supports reserved flows with absolute-rate reservations and best-effort flows which distribute the remaining capacity via weights. 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, our prototype network has a relatively high per-packet cost, so charging per-packet instead of per-byte is reasonable.

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 (this ignores 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 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 a moving-window average of the link-level throughput each flow achieves, as a percentage of what it would expect in the absence of errors. At the top of each plot a "Link" pseudo-flow represents the total link throughput as a fraction of the error-free link throughput.

Figure 11 displays the throughput obtained by four flows operating in the "walls" trace environment (Table 1). Two protected flows, audio and video, expect 1% and 44% of the link, while two unprotected flows will share the remainder. The audio flow has a power factor of 300% for reliability in most plausible error environments, and the video flow has a power factor of 223%. These values mean that the two protected flows will consume the entire link when the error rate

is 50% 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. In the trace, both packet losses and bit corruptions vary in severity due to a person moving through the main signal path. The scheduler ensures that error bursts, visible as sharp throughput drops on the "Link" plot, leave the audio flow essentially unharmed. While the video flow is not affected by mild error bursts, harsher ones result in temporary throughput reductions which are quickly compensated for. Finally, the scheduler divides the error burden equally between the two best-effort flows.

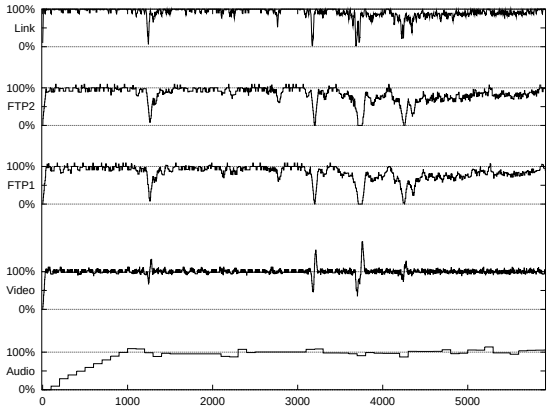

Figure 11: Response to a dynamic real-world error trace.

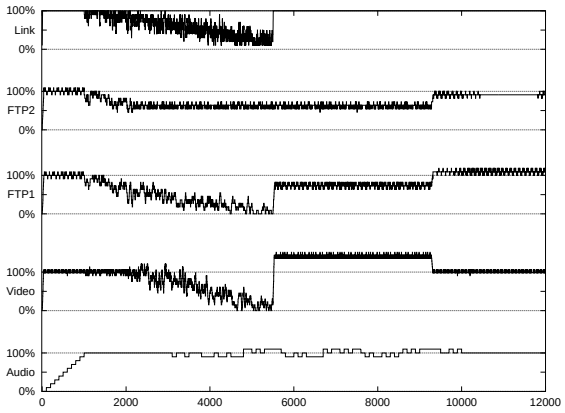

Figure 12: Location-dependent error trace.

Figure 12 is an example of a location-dependent error pattern: 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 pattern is a smoothly increasing frequency of packet loss, ranging from 10% to 90%, surrounded by periods without losses. The video flow's power factor has been lowered to a more-realistic 140%. 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. 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 effort-limited prioritization of reserved flows over best-effort flows coupled with effort-limited outcome-based fairness. The scheduler performs intuitively across a wide range of desired flow throughputs (1% to 44% of the link) and error rates (10% to 90%). We have verified these simulation results by inserting the ELF scheduler into our prototype wireless LAN and measuring the throughput obtained by the standard kernel TCP stack [14].

8 Related work

8.1 Transport-layer Adaptation

TCP decoupling [30] is a purely end-to-end technique for classifying packet losses as due to link errors versus congestion. As we discussed in Section 2.1, this general approach raises deployment issues and may suffer from efficiency problems in the face of high error rates. As we saw in Section 2.2, several groups have examined transport-level "indirect" or "split" approaches to improving end-to-end performance. Our experience suggests that TCP, and other transport protocols, may benefit from a simpler and protocol-blind link-level approach.

8.2 Other Link-Level Retransmission Systems

A WaveLAN-based evaluation of packet shrinking [19] also suggests an architecture including link-level fragmentation and adaptive error coding. An analysis of TCP running over noisy GSM telephone links [22] demonstrates the utility of adaptive frame sizing and the benefits of (persistent) link-level retransmission.

The IEEE 802.11 wireless LAN standard [16] 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.

Our approach to local retransmission is similar in spirit to that of MACA [18]. The MACAW [5] protocol design considers multiple flows per device and presents evidence that a post-packet acknowledgement increases TCP throughput.

A comparison [4] 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. Because our control loop is tighter and our retransmission scheme does not re-order packets, we achieve good performance without depending on TCP-specific protocol information.

A more-detailed evaluation of our link-level error control implementation may be found in [12].

8.3 Wireless link scheduling

Other researchers have investigated wireless-aware WFQ schedulers [20, 21, 25]; these and other error-sensitive wireless link schedulers are evaluated according to a unified framework [24]. These schedulers embody many attractive features, such as slot swapping based on error prediction, burst-averse scheduling, and formal delay bounds, but most share a fundamental philosophical difference with ELF. We believe that wireless link schedulers produce preferable outcomes when they consider not only flow weights, flow lags, and predicted channel error states but also each flow's tolerance for throughput variation. Two flows with equal weight, equal lag, and the same predicted channel error state will receive the same service from most wireless extensions of WFQ, but an ELF scheduler could preferentially schedule a video flow at the expense of an FTP flow based on differing power factors.

The *Server-Based Fairness Approach* [27] 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. While this approach can implement a wide universe of policies, we believe that this very generality calls for a simple and intuitive fairness model such as our power-factor approach.

Utility-fair bandwidth allocation [6] 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.

More detail about the particular ELF scheduler we evaluated may be found in [14].

9 Conclusion

We have argued and demonstrated that "pure" or "non-snooping" adaptive link-level error control vastly increases link throughput in the face of the high and dynamic error rates often found in wireless networks. While TCP can use a high fraction of the resultant link capacity, our work is not tied to TCP in particular, which suggests that existing Internet applications and infrastructure can be deployed across noisy wireless links.

To balance the needs of throughput-sensitive flows, fairness, and link efficiency in the presence of severe, time-varying, and location-dependent link capacity loss, we have proposed the "effort-limited fair" (ELF) scheduling approach. An ELF scheduler strives to achieve the outcomes envisioned by users (e.g., weighted link sharing or fixed-rate reservations) while limiting the effort spent on a flow according to a per-flow parameter called the power factor. This extension to WFQ scheduling can implement a variety of fairness and efficiency policies.

Trace-based simulations and measurements of an experimental prototype suggest that an Internet-style approach comprising adaptive link-level error control and error-sensitive scheduling produces an environment friendly to both traditional best-effort data traffic and applications with specific throughput needs.

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. Hui Zhang and Ion Stoica

provided extensive comments on our scheduling proposals. The responsibility for any inaccuracies or deficiencies in this work is, of course, our own.

References

[1] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

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

[18] 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.

[19] 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.

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

[21] 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.

[22] 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)*.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

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

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

[31] 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.

David A. Eckhardt is a Ph.D. candidate in the School of Computer Science at Carnegie Mellon University, where he has already received a M.S. in Computer Science. He received a B.S. in Computer Science, with a minor in Political Science, from The Pennsylvania State University in 1989. His research interests are in the areas of wireless and high-performance networking. He is a member of the IEEE Communications Society.

Peter A. Steenkiste is an Associate Professor in the School of Computer Science and the Department of Electrical and Computer Engineering at Carnegie Mellon University. He received the degree of Electrical Engineer from the University of Ghent in Belgium in 1982, and M.S. and Ph.D. degrees in Electrical Engineering from Stanford University in 1983 and 1987. His research interests are in the areas of high-performance networking and network quality of service. He is a member of the ACM and the IEEE Computer Society.