PPR: Partial Packet Recovery for Wireless Networks

Kyle Jamieson and Hari Balakrishnan
MIT Computer Science and Artificial Intelligence Laboratory
{jamieson, hari}@csail.mit.edu

ABSTRACT

Bit errors occur in wireless communication when interference or noise overcomes the coded and modulated transmission. Current wireless protocols may use forward error correction (FEC) to correct some small number of bit errors, but generally retransmit the whole packet if the FEC is insufficient. We observe that current wireless mesh network protocols retransmit a number of packets and that most of these retransmissions end up sending bits that have already been received multiple times, wasting network capacity. To overcome this inefficiency, we develop, implement, and evaluate a *partial packet recovery* (PPR) system.

PPR incorporates two new ideas: (1) *SoftPHY*, an expanded physical layer (PHY) interface that provides PHY-independent hints to higher layers about the PHY's confidence in each bit it decodes, and (2) a *postamble* scheme to recover data even when a packet preamble is corrupted and not decodable at the receiver.

Finally, we present *PP-ARQ*, an asynchronous link-layer ARQ protocol built on PPR that allows a receiver to compactly encode a request for retransmission of only those bits in a packet that are likely in error. Our experimental results from a 31-node Zigbee (802.15.4) testbed that includes Telos motes with 2.4 GHz Chipcon radios and GNU Radio nodes implementing the 802.15.4 standard show that PP-ARQ increases end-to-end capacity by a factor of 2× under moderate load.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless Communication*

General Terms

Design, experimentation, measurement

Keywords

Wireless, 802.11, Zigbee, layering, synchronization, ARQ

1. INTRODUCTION

Bit errors over wireless channels occur when the signal to interference and noise ratio (SINR) is not high enough to decode information correctly. In addition to noise, poor SINR arises from the interference caused by one or more concurrent transmissions in the network, and varies in time even within a single packet transmission. Thus a tension arises between permitting concurrent transmissions to increase spatial reuse, and receiving those transmissions correctly. Even with a variety of physical layer (PHY) techniques such as spread-spectrum and OFDM modulation, channel coding, and the like, current systems rely heavily on link-layer retransmissions to recover from bit errors and achieve high capacity. Since wireless channels are hard to model and predict, designing an error-free communication link generally entails sacrificing significant capacity; instead, a design that occasionally causes errors to occur fares better in this regard. Retransmissions allow a receiver to recover from lost packets.

Retransmitting entire packets works well over wired networks where bit-level corruption is rare and a packet loss implies that all the bits of the packet were lost (*e.g.*, due to a queue overflow in a switch). Over radio, however, all the bits in a packet don't share the same fate: very often, only a small number of bits in a packet are in error; the rest are correct. Thus, it is wasteful to re-send the entire packet: our goal is to eliminate this waste.

There are several challenges in realizing this goal. First, how can a receiver tell which bits are correct and which are not? Second, since most PHYs require the receiver to synchronize with the sender on a preamble before decoding a packet's contents, wouldn't any corruption to the preamble (caused, for instance, by a packet collision from another transmission) greatly diminish the potential benefits of the proposed scheme? Third, how can higher layer protocols use partial packets to improve end-to-end performance?

This paper presents the design, implementation, and evaluation of *PPR*, a *Partial Packet Recovery* system that improves aggregate network capacity by greatly reducing the number of redundant bits transmitted. Our key insight is to use information from the physical layer to improve error resilience. PPR incorporates the following two novel techniques, to meet the challenges mentioned above:

The SoftPHY interface (Section 2) allows the receiver to determine, with no additional feedback or information from the sender, which bits are likely to be correct in any given packet reception using hints from the PHY. The key insight in SoftPHY is that the PHY should pass up information about how close each received symbol or codeword was to the symbol or codeword the PHY decided upon. The higher layer can then use this information as a hint, independent of the underlying details in the PHY.

Postamble decoding (Section 3) allows a receiver to receive and decode bits correctly even from packets whose preambles are corrupted by other transmissions or noise. The main idea here is to replicate the information in the preamble and packet header in a postamble and a packet trailer, allowing a receiver to lock on the

This work was supported by the National Science Foundation under Award Numbers CNS-0520032 and CNS-0205445.

SIGCOMM'07, August 27–31, 2007, Kyoto, Japan.

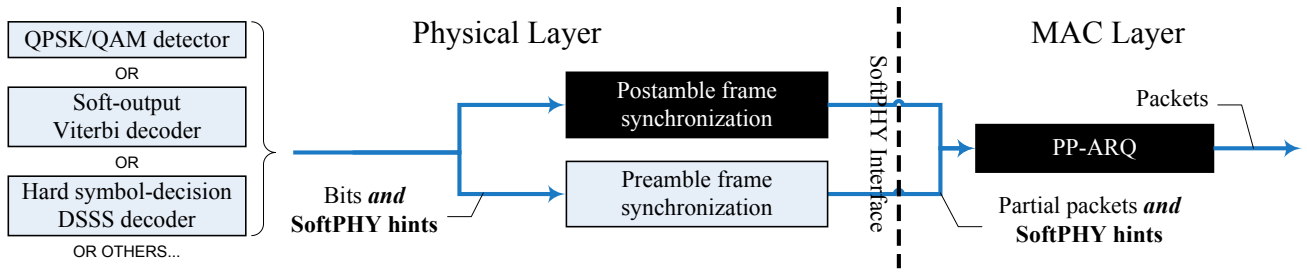

Figure 1: Block diagram of the PPR system; dark blocks and the SoftPHY interface are the contributions of this paper. Our contributions fit above one of many different types of receiver structure, modified to pass up SoftPHY *hints* (Section 2) to the MAC layer. Postamble decoding, described in Section 3, increases the number of opportunities for recovering partial packets from the receiver. SoftPHY hints propagate to PP-ARQ, the partial packet retransmission layer, described in Section 4.

postamble and then "roll back" in time to recover data that was previously impossible to decode.

Using PPR, we have designed **PP-ARQ** (Section 4), a link-layer retransmission protocol in which the receiver compactly requests the retransmission of only the select portions of a packet where there are bits likely to be wrong. In response, the sender retransmits the bits and checksums for those ranges, so that the receiver can eventually be certain that all the bits in the packet are correct. The receiver's request encoding uses a dynamic programming algorithm that minimizes the expected bit overhead of communicating this feedback, balancing that against the cost of the sender retransmitting bits already received correctly.

We have implemented each of the three above ideas for 802.15.4, the Zigbee standard. Our implementation is compatible with that specification. The SoftPHY and postamble decoding steps running at the receiver can recover partial packets from unmodified Zigbee senders, while PP-ARQ requires sender-side modifications. For additional insight, we have implemented PPR in an uncoded DQPSK receiver. Section 6 gives the details of all our implementation work.

The underlying premise in PPR is that significant performance gains can be obtained by the combination of a more aggressive, higher-rate PHY and being more flexible about the granularity of error recovery in wireless networks. Our techniques can improve performance in both access point-based networks and wireless mesh networks. Section 7 shows several experimental results that confirm this premise: in that section, we describe a 31-node indoor testbed consisting of telos motes with 2.4 GHz Zigbee radios from Chipcon and six GNU Radio nodes. Our results show factor-of-two gains over the status quo in aggregate end-to-end throughput using PP-ARQ. Our gains are even higher (4× better aggregate end-to-end throughput) under heavy load, which causes a number of links to have marginal quality. Even at light load we find that on the links with the lowest loss rates (which would be the ones selected by routing protocols), the raw success rate improves by 1.6×. We also compare PPR to other ways of determining which bits are likely to be correct, such as fragmented packet checksums.

2. SOFTPHY INTERFACE AND DESIGN

In current receivers, the PHY outputs only a sequence of bits after demodulation and channel decoding. This interface is rather limiting, since higher layers have no easy way of assertaining the certainty that the PHY has in each bit it outputs. The PHY, however, has this information, and in this section we discuss how it can annotate each group of bits output with the confidence it has in those bits' correctness. For three common receiver designs spanning the PHY design space, we specify this information and discuss

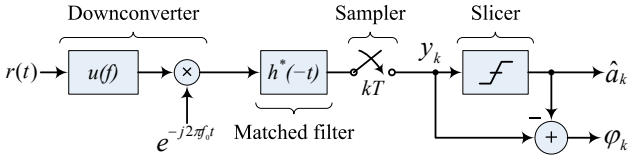

Figure 2: SoftPHY in a standard demodulator. For each output symbol $\hat{a}_k$, the demodulator produces SoftPHY hint φ_k.

implementation methods. In the following sections, we consider SoftPHY in an uncoded demodulator, a DSSS receiver with hard-decision symbol decoding, and soft-decision decoders. We begin with a key observation regarding the PHY's digital abstraction.

2.1 SoftPHY architecture

One benefit of the current layered receiver architecture is that the PHY provides a digital abstraction, which isolates layers above the PHY from implementation details of the PHY itself. While a variety of PHY implementations can provide the SoftPHY interface, the semantics of SoftPHY hints are tied to the details of the PHY, potentially violating this digital abstraction.

In PPR, layers above the PHY are not aware of how SoftPHY hints are calculated. Instead, they adapt their decisions on how to handle each bit based on observation. For example, the MAC layer could observe the correlation between a particular threshold and the correctness of the hint, and adapt the threshold dynamically. This approach can be used as long as the PHY simply provides a "monotonicity" contract; *i.e.*, given any two SoftPHY hint values, h_1 and h_2, $h_1 < h_2$ always implies that the PHY has a higher confidence in the bits associated with h_1 than with h_2 (or vice-versa).

Thus, while SoftPHY hints themselves are PHY-*dependent*, layers above the PHY use SoftPHY hints in a PHY-*independent* manner, retaining the benefits of the PHY's digital abstraction. At the same time, from an information-theoretic perspective, the SoftPHY design necessitates no loss of information: the bits and the hints can contain the same amount of information as the raw signal samples. We now briefly delve into the PHY, describing how three common designs can yield SoftPHY hints.

2.2 SoftPHY for an uncoded channel

Figure 2 gives a picture of a rudimentary digital receiver that has been agumented to return SoftPHY hints. Once the incoming signal $r(t)$ has been downconverted to baseband, the structure that maxi-

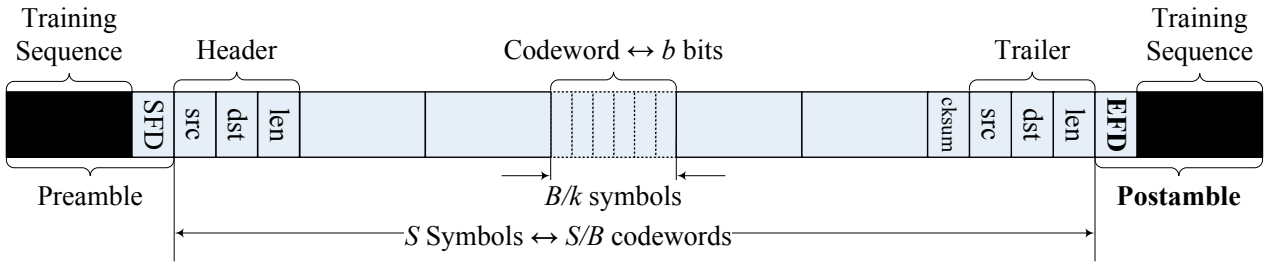

Figure 3: A frame is composed of S *symbols*, preceeded by a start-of-frame delimiter (SFD) and followed by an end-of-frame delimeter (EFD). Using the notation of Section 2.3, symbols are organized into S/B *codewords* of B/k symbols each by channel coding or direct-sequence spread spectrum. Each codeword encodes b data bits. Our novel contribution to the frame layout, the *postamble*, is shown in bold.

mizes SNR at the output of the receiver [7] is a filter $h^*(-t)$ matched to the shape of the transmitted signal as seen through the channel, $h(t)$. After sampling, the key element in the receiver is the *slicer*, which quantizes the sampled signal to one of a few symbols $\hat{a}_k$. For each quantized output symbol, we obtain PHY hints φ_k as the difference between the sampler output and the slicer output, as shown in Figure 2. This SoftPHY hint can be interpreted as the distance in signal space between $r(t)$'s constellation point and the decoded symbol's constellation point. We have implemented the design in Figure 2 for a DQPSK receiver. In Section 7.4, we evaluate these SoftPHY hints at marginal SNR.

2.3 SoftPHY in a hard-decision block decoder

Our Zigbee implementation, evaluated in Section 7.1, uses a hard decision decoder (HDD). To understand how SoftPHY works in our Zigbee receiver, a conceptual model of the wireless communication system will be helpful. This model also applies to common direct-sequence spread spectrum (DSSS) or OFDM-DSSS radio. In particular, it applies to both 802.15.4 (Zigbee) and 802.11b/g (WiFi), two common standards.

In block coding, the PHY maps groups of b source bits to a B-bit (B-chip) *codeword* as shown in Figure 3. Since there are only 2^b unique b-bit strings, the space of valid codewords is extremely sparse. The sender then groups the codewords into channel symbols encoding $k \geq 1$ bits each, and sends the channel symbols over the air, modulated over some baseband transmit waveform. In Zigbee, $k = 2$, $b = 4$, and $B = 32$, thus each group of four source bits in the original packet gets spread over 32 chips, or $B/k = 16$ channel symbols.

In an HDD design, the demodulator outputs *hard symbol decisions* ($\hat{a}_k$ in Figure 2) for each symbol in turn, independent of other symbols. It then sends that information to the channel decoder, which maps the received codeword to the closest valid codeword. The proximity of this mapping, measured as the Hamming distance between the received word and the codeword (the number of distinct elements between the two words), can serve as a useful confidence hint; we evaluate its performance below in Section 2.5.

2.4 SoftPHY in a decoder with soft decisions

For better performance at low SINR, a decoder can use *soft-decision decoding* (SDD) [7]. The SDD decoder works directly on samples of received symbols y_k, before they are sliced, thus using more information to make its decisions. However, SDD will still produce incorrect codewords at very low SINR, and does not recover correct bits particularly well during packet collisions.

In a block-based code, the SDD decoder calculates the correlation C between the received samples $\mathbf{Y}$ and each codeword $\mathbf{C_i}$ (whose jth bit is c_{ij}) defined as:

$$C(\mathbf{Y}, \mathbf{C_i}) = \sum_{j=1}^{n} y_j c_{ij}. \quad (1)$$

C can then serve as a SoftPHY hint from the PHY to higher layers.

In the case of a convolutional code, SoftPHY can use the soft output of the Viterbi [16] or BCJR [6] decoder. This output is a measure of how well the received symbol sequence matches with the path through the coding trellis associated with the chosen codeword.

In some ways, SoftPHY might seem analogous to soft-decision decoding, but there is a crucial architectural difference. With soft-decision decoding, the demodulator's interface to the decoder is quite different from hard decoding. In the former, the demodulator does not attempt to make a decoding decision, instead propagating received signal samples up to the decoder. In contrast, in the SoftPHY design, the PHY doesn't simply pass up all its raw information to the higher layer. The PHY still makes "hard" decisions, thus preserving layering boundaries. This architecture preserves a clean decomposition between PHY and higher layers while enabling performance gains.

2.5 SoftPHY experiments

We have conducted preliminary experiments with the HDD and SDD schemes described above. We found that our bit errors were mostly attributable to collisions, and in this case, the difference between HDD and SDD was not significant. Because the HDD implementation was conceptually simpler, we developed a complete implementation of that idea and conducted several experiments with it (described in detail in Section 7.1). Here, we give brief experimental results showing that Hamming distance is a good SoftPHY hint.

We first take a detailed look at a particular partial packet reception, showing the receiver's view of each codeword. Figure 4 shows a receiver's view of a packet sent from one sender, at two different codeword synchronization offsets. The packet contains a known bit pattern, against which we test each received codeword for correctness. The result of each test is indicated by the presence or absence of a triangle in the figure.[1] The upper plot in Figure 4 shows the packet arriving at the receiver at time[2] 0, and achieving synchronization at time 10 (lower plot). When the PHY syn-

[1]For clarity, we show the result of every fourth codeword-correctness test.

[2]Measured in units of codeword-time, 16 μs in our radios.

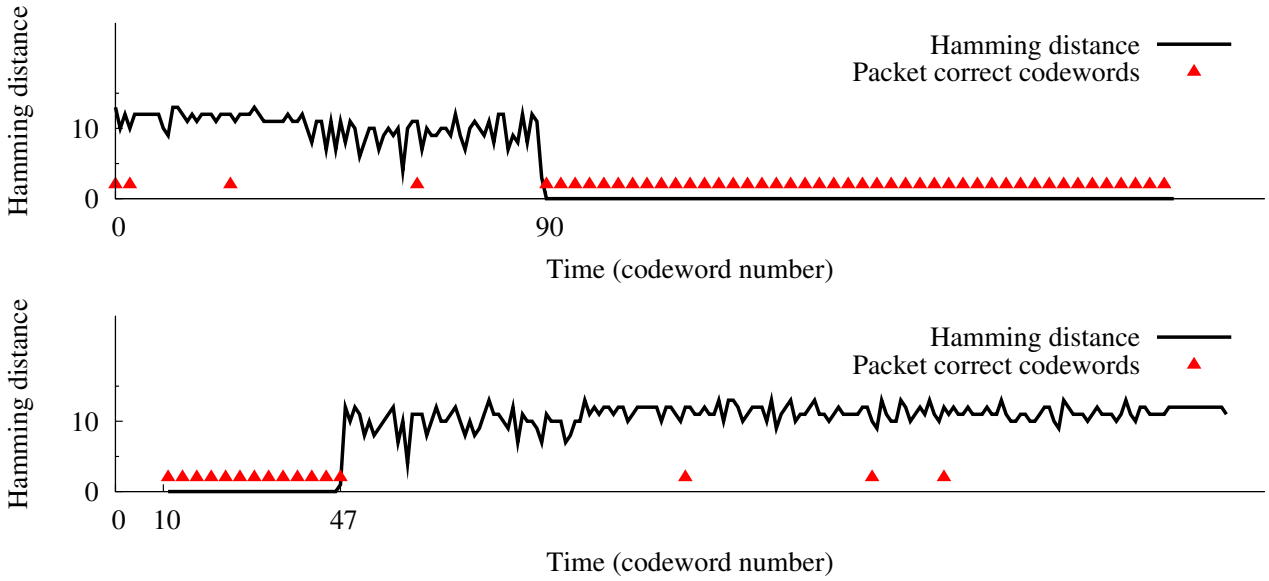

Figure 4: Partial packet reception at two different codeword synchronization offsets during a loss in codeword/symbol synchronization: codeword correctness (triangle indicators) and each codeword's associated Hamming distance (curves). Despite uncertainty in PHY codeword timing recovery, Hamming distance indicates the correct parts of the packet to higher layers.

chronizes on the packet, symbol timing recovery succeeds and the receiver decodes approximately 40 codewords correctly (including the preamble) before losing symbol or codeword synchronization. We see that Hamming distance remains at 0 for the duration of the correct codeword decisions, and rises at time 47 when the burst of errors occurs. The PHY passes these Hamming distance hints up to the ARQ layer along with all the codewords in the packet.

Later, at time 90 at the other synchronization offset (upper plot), the receiver successfully synchronizes on and decodes a run of codewords extending to the end of the first packet. Since this packet data is at a different synchronization offset to the preamble, it relies on its postamble in order to frame-synchronize and pass up the partial packet reception and associated SoftPHY hints.

We perform the next experiment in a 31-node Zigbee/software radio testbed described below in Section 7. All but four of the nodes send packets containing a known test pattern, at a constant rate. There are four receivers, each able to hear and decode some subset of the senders. Figure 5 shows the distribution of Hamming distance across each received codeword, separated by whether the codeword was correctly or incorrectly received (we know this from the test pattern). Conditioned on a correct decoding, only about one in 100 codewords have a Hamming distance of two or more. Conversely, fewer than one in 10 incorrect codewords have a distance of two or less.

This result shows that the higher layer can interpret this SoftPHY hint with a threshold rule. We denote the threshold by η, so that the higher layer labels groups of bits with $d \leq \eta$ "good" and groups of bits with $d > \eta$ "bad." Under the threshold rule then, the two curves in Figure 5 also show the probability of misclassification for correct and incorrect symbols, respectively. We analyze these results further in Section 7.2.

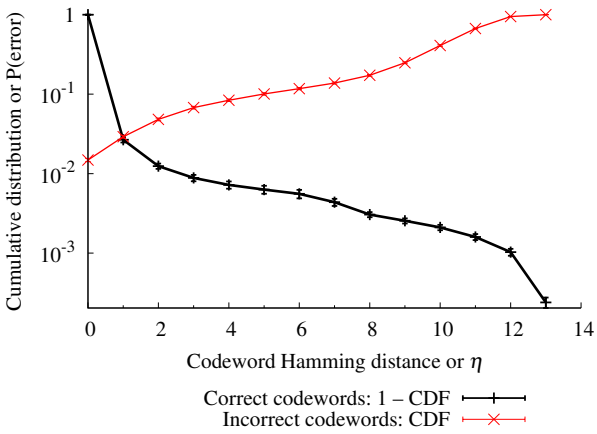

Figure 5: Distributions of Hamming distances for every codeword in every received packet, separated by whether the received codeword was correctly or incorrectly decoded. Under the threshold rule, these CDFs can be reinterpreted as curves plotting probability of misclassification.

3. POSTAMBLE PACKET DECODING

When many errors occur in the preamble due to collisions or noise, current radio receivers will not be able to synchronize with the incoming transmission and decode any bits. In that case, the potential benefits of SoftPHY will go largely unrealized. We need a way to mitigate the effects of preamble loss, for example, in the multi-packet collision shown in Figure 6. In this example, a receiver would not be able to decode any part of packet *P4*, since its preamble was corrupted, while packet *P3* would be received and discarded due to a bad checksum. SoftPHY may help with *P3*, but we are interested in *P4* as well.

Our approach to synchronizing on packets without an intelligible preamble is to add a *postamble* to the end of each packet on which a receiver can also synchronize. The postamble has a well-known sequence attached to it that uniquely identifies it as the postamble, and differentiates it from a preamble ("EFD" in Figure 3). In

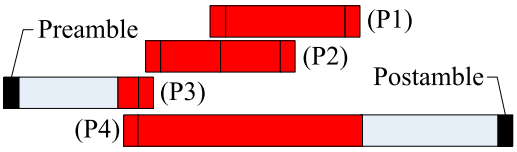

Figure 6: A four-packet collision. We use *P4*'s postamble to partially decode that packet, and the techniques described in Sections 2 and 4 to detect the incorrect parts of both *P3* and *P4* and request retransmission of just those parts.

addition, we add a *trailer* just before the postamble at the end of the packet, also shown in Figure 3. The trailer contains the packet length, source, and destination addresses. Just as with header data, the receiver uses the SoftPHY interface to check the correctness of the trailer.

To recover the payload after hearing just a postamble, the receiver maintains a circular buffer of samples of previously-received symbols even when it has not heard a preamble. In our implementation, we keep as many sampled symbols as there are in one maximally-sized packet. When the receiver detects a preamble, the behavior is the same as in the status quo. If not, then if the receiver detects a postamble, it takes the following steps:

1. "Roll back" as many symbols as are in the packet trailer.
2. Decode and parse the trailer to find the start of the entire packet, and the sender and receiver identities.
3. "Roll back" in time as many symbols as are in the entire packet, to decode as much of the packet as possible.

The main challenge of postamble decoding is addressing how a receiver can keep a modest number of samples of the incoming packet in a circular buffer while still allowing the various receiver subsystems to perform their intended functions. These functions include carrier recovery, symbol timing recovery, and equalization. We meet each of these challenges in our implementation, as briefly outlined below.

Most receivers need to perform symbol timing recovery [7, Chp. 16] to determine when (*i.e.*, with which frequency and phase) to sample the incoming signal such that the probability of detection is maximized. In our system, we use the popular decision-directed timing recovery algorithm [24]. Next, the demodulator may[3] need to perform carrier recovery [7, Chp.15] to estimate the incoming carrier's time-varying frequency and phase. A number of techniques for countering inter-symbol interference rely on estimating the channel impulse response (equalization) [7, Chp. 8]. Typically the preamble includes a known *training sequence* to enable the equalizer to quickly estimate the channel's response during synchronization. We can therefore include the same training sequence in the postamble (see Figure 3) and post-process the samples of the signal in the body of the packet afterwards, using standard signal processing techniques [17].

4. PP-ARQ: PPR + RETRANSMISSIONS

SoftPHY and postamble detection together allow higher layers to discover which received codewords are likely to be correct and which are not. We now examine the problem of how the receiver can most efficiently communicate this information back to the sender, to improve the performance of link-level retransmissions.

[3]Some modulation techniques permit the use of non-coherent detection where carrier recovery is not necessary.

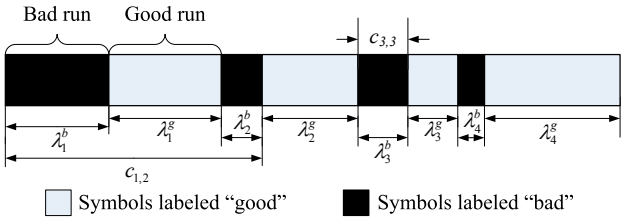

Figure 7: After computation of *run-length representation* of a received packet, the first step in PP-ARQ at the receiver. Run lengths $\lambda_i^{b,g}$ are as defined in expression 2. Chunk $c_{1,2}$ is defined in expression 3.

The naïve way to provide feedback is for each receiver to send back the bit ranges of each part of the packet believed to be incorrect. Unfortunately, doing that may consume a large number of bits, because encoding the start of a range and its length can take up to on the order of $\log S$ bits for a packet of size S. Hence, we need to develop a more efficient feedback scheme.

After the receiver has decoded a packet, it has a list of received symbols S_i, $1 \le i \le N$, and a list of associated PHY layer hints φ_i where φ_i is the confidence the PHY has in symbol S_i. Then it uses the threshold test on each confidence φ_i, and labels each symbol "good" or "bad."[4] Next, it computes alternating *run lengths* λ_j^g, λ_j^b, $1 \le j \le L$ of good and bad symbols, respectively, to form the *run-length representation* of the packet as shown in Figure 7. This representation has the form:

$$\lambda_1^b \lambda_1^g \lambda_2^b \lambda_2^g \cdots \lambda_L^b \lambda_L^g \tag{2}$$

Here, λ_j^g is the count of symbols in the jth run of symbols all rated "good" by SoftPHY, shown with light shading in the figure. Similarly, λ_k^b is the size of the kth run of symbols rated "bad" by SoftPHY, shown with dark shading in the figure.

The receiver forms a list of *chunks* $c_{i,j}$: groups of runs that it will ask the sender to retransmit. Chunk $c_{i,j}$ contains all the bad and good runs in between and including bad run i and bad run j, so each chunk starts and ends with bad runs. For example, chunks $c_{1,2}$ and $c_{3,3}$ appear in Figure 7. Note that chunk $c_{i,j}$ does not include λ_j^g, the last run of good symbols in the chunk:

$$c_{i,j} = \lambda_i^b \lambda_i^g \lambda_{i+1}^b \lambda_{i+1}^g \cdots \lambda_j^b \tag{3}$$

Once the receiver has made the choice of which chunks to request from the sender, it sends a *feedback packet* communicating this information. We next show that each chunk can be assigned a cost function, and that the problem of which chunks to request exhibits the "optimal substructure" property in that the cost for an entire chunk is easily derived from the cost of two suitably divided portions. When the sender responds to the receiver's feedback packet, it also sends the checksums of the good chunks so that the receiver can verify that they are correct.

4.1 Dynamic programming to find the best feedback strategy

If λ_k^g, $i \le k \le j$ are all small and $j - i$ is large, we would favor requesting that the entire chunk $c_{i,j}$ be retransmitted, because the

[4]We note that this "slicing step" is analogous to hard-decision decoding, and leave as future work improvements to PP-ARQ that take into account the values of the SoftPHY hints themselves, although we perform an analysis of how often SoftPHY hints are wrong in Section 7.2.

additional bits it would take for the receiver to describe each of the $j - i$ individual chunks would far exceed that needed to retransmit the good symbols associated with chunk $c_{i,j}$. If, on the other hand, some of the λ_k^g, $i \leq k \leq j$ are large, and/or $j - i$ is small, we would favor asking for the individual chunks $c_{k,k}$ for each $k \in [i, j]$ for the converse reason.

We define the *cost* of a chunk as follows ($i \neq j$):

$$C(c_{i,i}) = \log S + \log \lambda_i^b + \min\left(\lambda_i^g, \lambda_C\right) \quad (4)$$

$$C\left(c_{i,j}\right) = \min\left\{2\log S + \sum_{l=i}^{j-1} \lambda_l^g, \min_{i \leq k \leq j-1}\left\{C\left(c_{i,k}\right) + C\left(c_{k+1,j}\right)\right\}\right\} \quad (5)$$

For the receiver to describe the length and offset of the ith bad run to the sender, it takes approximately $\log S + \log \lambda_i^b$ bits, where S is the packet length. The receiver also sends the ith good run or a checksum of it to the sender, so that the sender can verify that it received the good run correctly. This takes $\min\left(\lambda_i^g, \lambda_C\right)$ bits, where λ_C is the length of the checksum. These two terms form the base case cost of a chunk in Equation 4.

The receiver then runs the recursive steps of the DP algorithm on the run-length representation of the packet. Equation 5 describes this computation. The outer min chooses between leaving chunk $c_{i,j}$ intact (thus resending all good runs within the chunk), or splitting the chunk into two smaller chunks and thus diving deeper into the recursive computation. The innermost min operator chooses how to make the split, if one is needed.

We compute the optimal chunking bottom-up using a table to memoize the costs of each possible chunking. Note that because the chunking algorithm operates on chunks, the table has as many entries as there are chunks in the packet, L. To analyze the computational complexity of this algorithm, we note that it can be implemented in a bottom-up fashion using a table to memoize the costs of each possible chunking. This results in an $O(L^3)$ implementation.

4.2 The streaming ACK PP-ARQ protocol

The receiver-side dynamic programming algorithm described above chooses chunks such that each chunk "covers" all the bad runs in the packet, and may cover some good runs, if they are short enough. We now describe the complete PP-ARQ protocol between sender and receiver.

1. The sender transmits the full packet with checksum.
2. The receiver decodes the packet (possibly partially), and computes the best feedback as described in Section 4.1.
3. The receiver encodes the feedback set in its reverse-link acknowledgement packet (which may be empty, if the receiver can verify the forward link packet's checksum).
4. The sender retransmits only (a) the contents of the runs the receiver requests, and (b) checksums of the remaining runs.

This process continues, with multiple forward-link data packets and reverse-link feedback packets being concatenated together in each transmission, to save per-packet overhead.

5. OTHER APPLICATIONS OF PPR

PPR has the potential to improve the performance of mesh network protocols such as opportunistic routing [10] and network coding. Using PPR, nodes need only forward or combine the bits likely to be correct in a packet that does not pass checksum, thus improving network capacity. Rather than use PP-ARQ, the integrated MAC/link layer that implements ExOR or network coding would directly work with SoftPHY's output. Alternatively, PP-ARQ could operate in the "background" recovering erroneous data, while the routing protocol sends the correct bits forward.

PPR also has the potential to improve the performance of multi-radio diversity (MRD) schemes [23] in which multiple access points listen to a transmission and combine the data to recover errors before forwarding the result, saving on retransmissions. Avudainayagan [5, 35] *et al.*develop a scheme in which multiple nodes (*e.g.*, access points) exchange soft decision estimates of each data symbol and collaboratively use that information to improve decoding performance. For this application, PPR's SoftPHY hints would provide a way to design a protocol that does not rely on the specifics of the PHY, unlike this previous work. Thus, with PPR, we may be able to obtain the simpler design and PHY-independence of the block-based combining of [23], while also achieving the performance gains of using PHY information.

6. IMPLEMENTATION

Zigbee sender. Each Zigbee sender node is a telos mote with a Chipcon CC2420 radio [33]. Senders run TinyOS[5] on the telos's TI MSP430 microprocessor. The CC2420 radio is a 2.4 GHz RF transceiver that uses direct-sequence spread spectrum (DSSS) at a rate of 2 Msymbols/s with $B = 32$ symbol codewords.[6] Each of the 16 codewords encodes $b = 4$ bits, implying a peak link data rate of 250 Kbits/s when there are no other transmissions in progress. The radio's underlying modulation is O-QPSK with half sine pulse shaping, also known as min-shift keying (MSK) [25].

Zigbee receiver. Each of the Zigbee receivers in the following experiments is a computer connected to a software-defined radio. The hardware portion of the receiver is a Universal Software Radio Peripheral (USRP) [13] with a 2.4 GHz daughterboard; the remainder of the receiver's functionality (demodulation and block decoding as described in Section 2.3) is implemented in software. The DSSS despreading function was written in C++ in the GNURadio [15] framework by the authors, with parts derived from code written by Schmid [29]. We implemented preamble and postamble frame synchronization in C++.

DQPSK transceiver. We have implemented a software-defined radio transmitter and receiver using a combination of the USRP hardware with CppSim [26] and Matlab. The transmitter uses differentially-coded QPSK with square-root raised cosine pulse shaping, for an aggregate data rate of 1.33 MBps. Apart from the differential coding, there is no channel coding layer in this radio.

PP-ARQ. We have implemented PP-ARQ both in trace-driven simulation and in the GNURadio framework. The PP-ARQ receiver uses SoftPHY to compute the run-length representation of the packet as defined in Section 4, and runs the dynamic programming algorithm described in Section 4.1 on the run-length representation of the packet. It then sends a feedback packet to the sender summarizing which runs need retransmitting. At the sender, our PP-ARQ implementation parses the recevier's feedback packet, computes checksums of each run needing retransmission, packs the runs into a fragmented CRC packet (with variable sized fragments), and transmits the fragmented CRC packet to the receiver.

7. EVALUATION

We now describe our experimental evaluation of the PPR system and PP-ARQ. We begin in Section 7.1 with an evaluation of PPR capacity improvements in a busy network where collisions cause SINR to fluctuate. In Section 7.2 we continue the experiments of Section 2.5, taking a closer look at SoftPHY hints. In Section 7.3

[5]See http://tinyos.net.

[6]We use the notation of Section 2.3.

Experiment (Radio)	Section	Conclusion
PPR in a busy network	7.1	PPR (SoftPHY and postamble decoding) improve four-fold the amount of correct bits the PHY delivers to higher layers.
SoftPHY hints in a busy network (DSSS/MSK)	7.2	The pattern of "misses" and "false alarms" under the SoftPHY hints we propose enable partial packet recovery in a busy network.
PP-ARQ in a busy network (DSSS/MSK)	7.3	PP-ARQ improves aggregate end-to-end throughput over the status quo by more than 4× under high load and 2× under moderate load. SoftPHY improves capacity even more than fragmented CRC, without needing performance tuning.
PPR at marginal SNR (DQPSK)	7.4	SoftPHY hints work best at low SNRs (BER less than 10^{-6}) but well for BERs as high as 10^{-3}. Coding further improves SoftPHY hint efficacy.
PP-ARQ implementation (DSSS/MSK)	7.5	PP-ARQ acheives significant end-to-end savings in retransmission cost, a median factor of 50% reduction.

Table 1: A summary of the major experimental contributions of this paper.

Figure 8: Experimental Zigbee testbed layout: there are 31 nodes in total, spread over 11 rooms in an indoor office environment. Each unnumbered dot indicates a Zigbee node. Software radio nodes are shown dark, labeled with numbers.

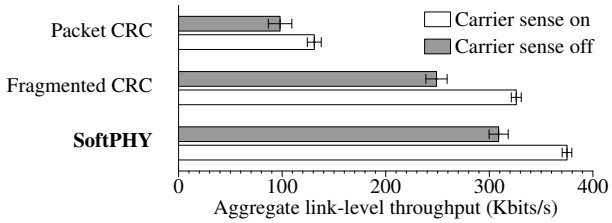

Figure 9: The impact of carrier sense on aggregate link-level throughput. Carrier sense improves throughput under each scheme, but PPR techniques yield further improvements. Postamble decoding is off in this experiment.

we evaluate PP-ARQ using trace-driven simulation. In Section 7.4 we evaluate PPR's efficacy in a quiet network on links close to the SNR threshold. Finally, in Section 7.5, we present a preliminary evaluation of a real PP-ARQ implementation. We summarize our experimental findings in Table 1.

7.1 PPR in a busy network

Each sender in the following experiments is a moteiv tmote sky wireless sensor node, equipped with an DSSS radio as described in Section 6. We have deployed 25 sender nodes over eleven rooms in an indoor office environment, as shown in Figure 8. Each receiver is a Zigbee software radio, also as described in Section 6. We also deployed six receivers among the senders; in the absence of any other traffic, each receiver could hear between four and ten sender nodes, with the best links having near perfect delivery rates.

We now present trace-driven channel capacity results evaluating how well the combination of SoftPHY (with the Hamming distance hint described in Section 2) and postamble decoding performs against the fragmented CRC scheme described in Section 7.1.2. In this experiment each node sends a stream of bits, which are formed into traces and post-processed to simulate a range of packet sizes realistic for a mesh network [30] (this technique is accurate in the busy, collision-dominated network that we evaluate in this section). In the underlying experiments, all senders transmit at the same time, offering 6.9 Kbits/s/node unless otherwise noted. Data points represent averages of 14 runs unless otherwise noted, and all error bars indicate 95% confidence intervals.

Summarizing each scheme:

1. **Packet CRC** computes a 32-bit CRC check over the received packet payload and discards the packet if it does not pass.
2. **Fragmented CRC**, described in Section 7.1.2, breaks the packet into fragments, appending to each a 32-bit CRC. Fragmented CRC delivers only those fragments with matching checksums, discarding the remainder.
3. **SoftPHY** delivers the high-confidence bits: exactly those bits in the packet whose codewords had a Hamming distance less than $\eta = 2$.
4. **PP-ARQ** is an implementation of the full PP-ARQ protocol (as described in Section 4) using PPR (SoftPHY and postamble decoding).

7.1.1 The impact of carrier sense

One potentially confunding factor in PPR's evaluation is the use and efficacy of carrier sense in the senders' CC2420 radios: carrier sense can fail due to hidden terminals or backoff slots smaller than the transmit-to-receive turnaround time [12]. To address this factor, we examine aggregate throughput for each scheme, with and without carrier sense. In Figure 9 we see that carrier sense improves throughput by a statistically significant amount over the status quo ("Packet CRC" with postamble decoding off). Noting that carrier sense yields additive improvements for each scheme, we narrow the design space of our evaluation to only include carrier sense on in the remaining experiments.

7.1.2 An alternative: per-fragment checksum

We will show next that the PPR improves performance significantly, but one might ask whether it is necessary to achieve similar gains. One way to approximate SoftPHY is to adopt a technique similar to that proposed by Ganti *et al.* [14], splitting the packet into fragments, and sending multiple checksums per packet, one per fragment, as shown in Figure 10. This scheme allows the re-

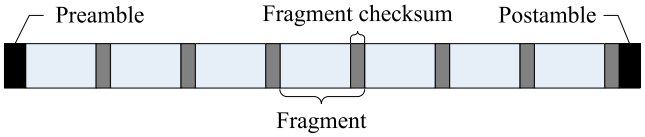

Figure 10: The per-fragment checksum approach: the packet includes multiple checksums, with each checksum taken over a different fragment of the packet.

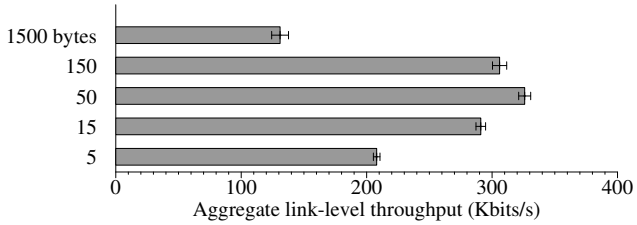

Figure 11: The impact of fragment size on the performance of the fragmented checksum scheme. Postamble decoding is off, carrier sense on in this experiment.

ceiver to identify entire fragments that are correct. If bit errors are concentrated in only a few bursts, then entire fragments will checksum correctly, and the receiver would then only have to recover the erroneous fragments from the sender.

How big must a fragment, c, be? In an implementation, one might place a checksum every c bits, where c varies in time. If the current value leads to a large number of contiguous error-free fragments, then c should be increased; otherwise, it should be reduced (or remain the same). Alternatively, one might observe the symbol error rate (or bit error rate), assume some model for how these errors occur, and derive an analytically optimal fragment size (which will change with time as the error rate changes). In either case, the fragmented checksum needs tuning for the optimal fragment size.

To find the optimal chunk size for the fragmented CRC scheme, we conducted experiments comparing aggregate throughput as fragment size varies. The results are shown in Figure 11. We see that when chunk size is small, checksum overhead dominates; while large chunk sizes lose throughput because collisions and interference wipe out entire whole fragments. We therefore choose a fragment size of 50 bytes (corresponding to 30 fragments per packet) for the following experiments.

7.1.3 PPR raw throughput

To gain further insight about PP-ARQ's performance gains, we first look one layer deeper, at the throughput achieved at the SoftPHY interface. Figure 12 compares the per-link distribution of throughputs at medium offered load for each scheme. Since the postamble and the preamble usually share fate in the packet-level CRC scheme, performance with or without postamble decoding is very close, and so for clarity, we omit the curve for packet-level CRC with postamble decoding. Per-link, we see that fragmented CRC yields a substantial throughput gain over the status quo, and that SoftPHY yields a small gain over fragmented CRC without the need for tuning the fragment size, as noted above. Furthermore, postamble decoding yields another small and additive raw throughput gain over each scheme.

The scatter plot in Figure 13 compares the end-to-end throughput for fragmented CRC on the x-axis with either SoftPHY (top half)

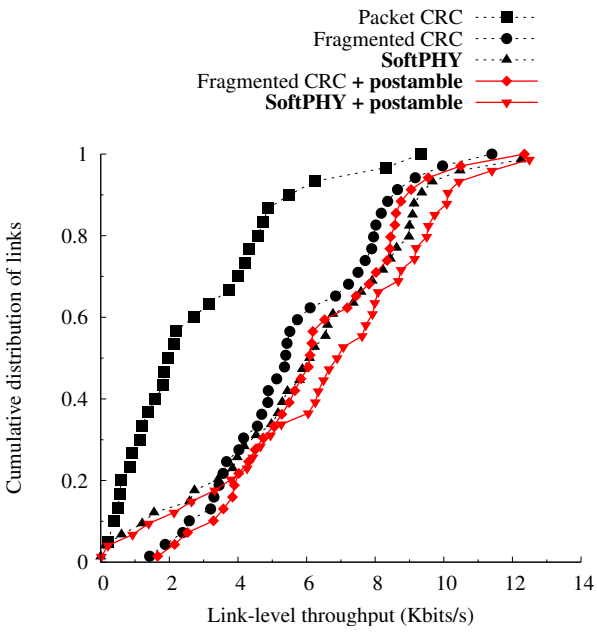

Figure 12: Per-link throughput distribution achieved at the SoftPHY interface. The offered load is 6.9 Kbits/s/node, close to channel saturation.

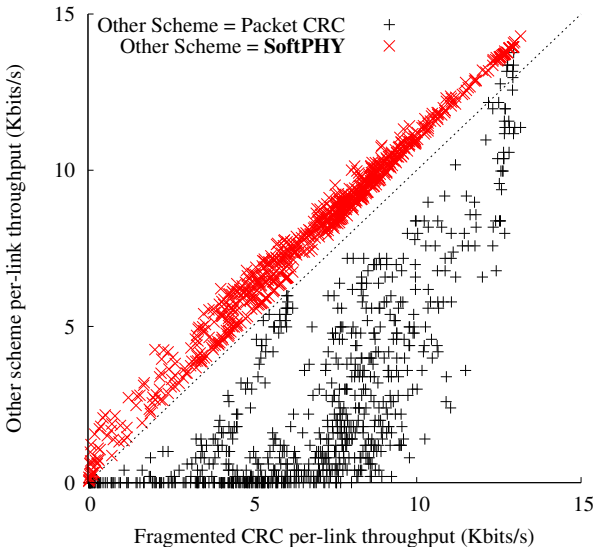

Figure 13: Link-by-link comparison of throughput through the SoftPHY interface. Each data point represents one link in one experimental run. Upper half: SoftPHY vs. fragmented CRC. Lower half: packet-level CRC vs. fragmented CRC.

or packet-level CRC (bottom half). The first comparison we can draw from this graph is the per-link throughput of SoftPHY compared with fragmented CRC (top-half points). We see that SoftPHY improves per-link performance over fragmented CRC by roughly a constant factor. This factor is related to the fragment size, and may be attributed to fragmented CRC's need to discard the entire fragment when another transmission corrupts part of it.

The bottom-half points in Figure 13 compare fragmented CRC with packet-level CRC. We see that fragmented CRC far out-

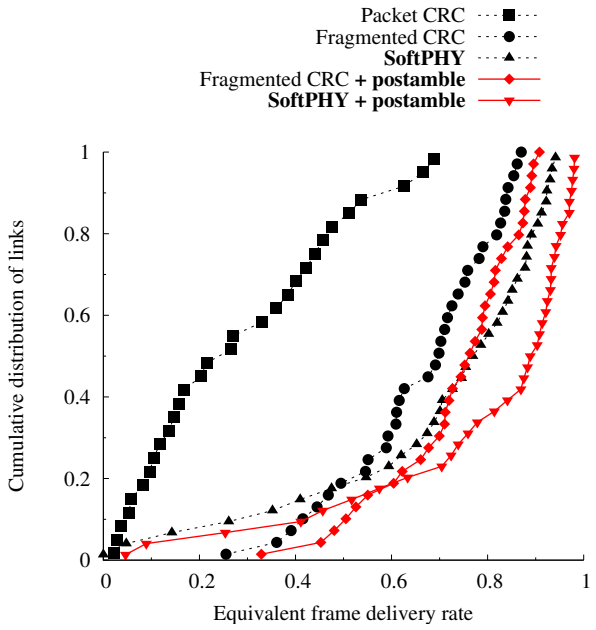

Figure 14: Per-link equivalent frame delivery rate with carrier sense enabled, at moderate offered load (3.5 Kbits/s/node).

performs packet CRC, because it only has to discard a small fragment instead of the entire packet when that fragment is corrupted. The fact that the circle points are dispersed on the y-axis and not on the x-axis means that the spread in the link quality distribution decreases when moving to smaller fragment sizes or SoftPHY. This is probably again because collisions do not occur over the entire packet, but rather often over a small piece of it.

7.1.4 PPR equivalent frame delivery rate

We now examine the rate at which each scheme described above delivers bits to higher-layers, once it has succesfully acquired a packet (i.e., the PHY has detected either a preamble or a postamble). We term this rate the *equivalent frame delivery rate*, because it measures how efficient each scheme is at delivering bits to higher layers once the PHY layer successfully synchronizes.

Figure 14 shows the per-link distribution of equivalent frame delivery rate in our network when there is a moderate offered load (3.5 Kbits/s/node). Even when carrier sense and postamble decoding are enabled, we see a large proportion of extremely poor links in the status quo network, but PPR techniques increase frame delivery rate substantially. For both SoftPHY and fragmented CRC, postamble decoding increases median frame delivery rate by the fraction of bits that come from packets whose preamble was undetectable, roughly 10%. Comparing packet-level CRC with fragmented CRC, we see a large gain in frame delivery rates because fragmented CRC does not throw away the entire packet when it detects an error. PPR improves on frame delivery rates even more by identifying exactly which portions of the frame are correct and passing exactly those bits up.

7.2 SoftPHY hints in a busy network

In Section 2 we introduced the SoftPHY hints that we use in our experimental evaluation; in Section 2.5 we saw that SoftPHY hints were a good predictor of correct decoding during an example packet reception and in a busy network. We now examine the statistics of the Hamming distance hint in further detail.

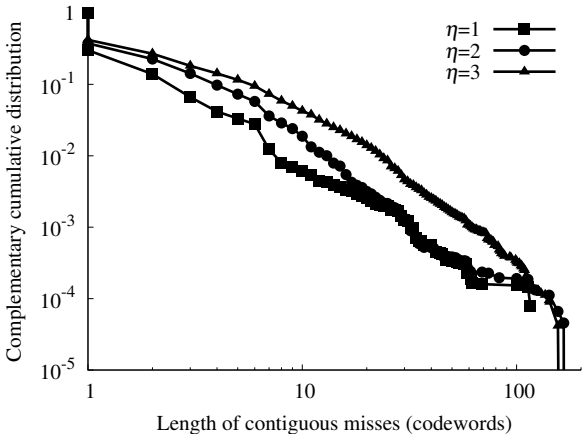

Figure 15: The distribution of lengths of contiguous misses in every received packet for various thresholds η.

Recall that we label a codeword "good" when its Hamming distance is less than or equal to η. Therefore the cumulative distribution function of incorrect codewords in Figure 5 is also the fraction of incorrect codewords that we incorrectly label good, and for which the CRC check on the resulting packet or partial packet fails. We call this fraction the *miss rate* at threshold η, the rate at which we "miss" labeling a codeword bad at Hamming distance thresold η. We see from the figure that the miss rate is one in ten codewords at $\eta = 6$, initially a cause for concern. The saving grace is that when misses occur, it is highly likely that there are correctly-labeled incorrect codewords around the miss, and so PP-ARQ will choose to retransmit the missed codewords. Figure 15 verifies this intuition, showing the complementary CDF of contiguous miss lengths at various thresholds η. We see that a significant fraction of misses are of length one, and that long runs of misses are extremely rare.

In Figure 5, we see the complementary cumulative distribution of correct codewords' Hamming distances. Since we label a codeword "bad" when its distance exceeds η, this complementary CDF is also the fraction of correct codewords that we incorrectly label "bad" (and which PP-ARQ retransmits) at threshold η. Noting that the overhead of this event is low—just one unnecessarily transmitted codeword—we see that its occurence is also low.

7.3 PP-ARQ in a busy network

We now present results for an implementation of the PP-ARQ protocol as described in Section 6. Our results in this section are from trace-driven simulation of PP-ARQ, using traces from the testbed of Figure 8. Using trace-driven simulation, we simulate a range of different packet sizes realistic for a mesh network [30], attaching a 24-byte preamble [2] to each packet.

Figure 16 shows the aggregate received thoughput across all links in the testbed for packet-level CRC (the status quo), fragmented CRC, and PP-ARQ. We see that PP-ARQ achieves roughly a 2× capacity improvement over the status quo, **without** needing the fragment-size tuning described in Section 7.1.2.

One significant cause of our performance improvements over the status quo is the avoidance of retransmitting data that reached the receiver, but was discarded due to a bad checksum. Figure 17 quantifies this intuition. In the status quo ("Packet CRC" in the figure), retransmissions are always packet-sized in length, and so we see only the modes of the packet distribution in the retransmit-size distribution. Fragmented CRC tuned with a fragment size of 50 bytes

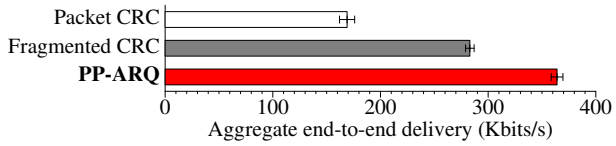

Figure 16: Comparison of the aggregate end-to-end delivery rate between packet-level CRC, fragmented CRC, and the PP-ARQ implementation. Postamble decoding is on in this experiment.

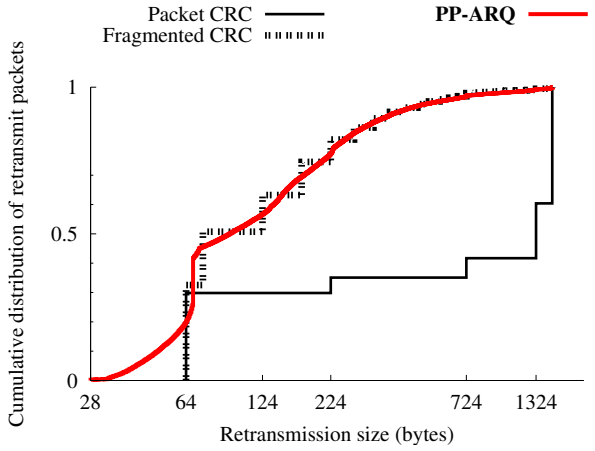

Figure 17: Comparison of the distribution of retransmission sizes for packet-level CRC, fragmented CRC, and the PP-ARQ implementation. Note PP-ARQ's long tail of short retransmit sizes.

breaks the retransmissions down into fragments of size $50 \times k$ for positive integers k, resulting in the stair-step pattern in the figure. However, fragmented CRC transmits no fragments smaller than 64 bytes. In contrast, PP-ARQ transmits a significant fraction of very small packets (less than 64 bytes), the cause of its significant performance gains. Note from Section 4 that PP-ARQ batches its retransmissions to avoid preamble overhead on each of the smaller retransmissions.

Figure 18 shows how end-to-end delivery rate changes when we increase the offered load to in the network. As well as raw offered load, we show the percentage of link capacity each node offers in the figure. At higher offered loads we see packet-level CRC performance degrading substantially. There have been several recent studies that attempt to elucidate the causes of this loss [3, 31, 32]. PP-ARQ's end-to-end throughput increases despite the overload, suggesting that only relatively-small parts of frames are actually being corrupted in overload conditions in the status quo.

7.4 SoftPHY hints at marginal SNR

We now turn from networks with significant interfering transmissions to an evaluation of SoftPHY hints in a quiet channel at marginal SNR. To perform these experiments, we utilized a frequency band that does not overlap with 802.11 [32], the dominant source of RF interference in our environment.[7] The experiments in this section use a software radio-based DQPSK transmitter and receiver pair, whose implementation is described above in Section 6.

[7]We used GNURadio tools to check for significant interference in our channel between runs of these experiments.

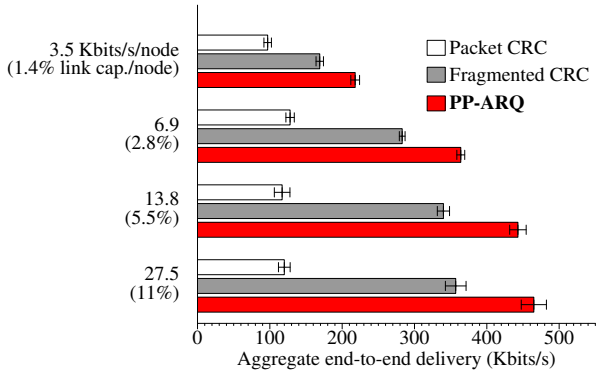

Figure 18: Comparison of end-to-end delivery rate in overload conditions; PP-ARQ scales favorably compared to the status quo. Postamble decoding is enabled in this experiment.

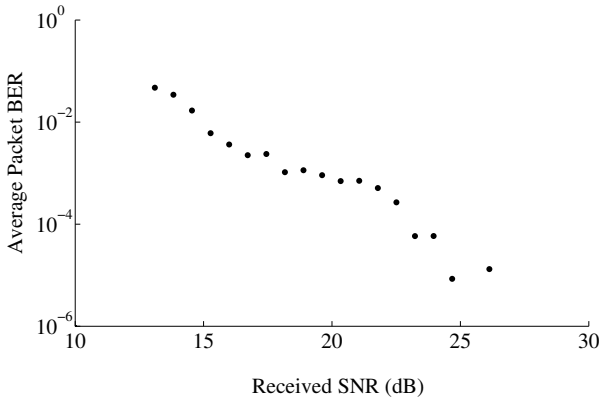

Figure 19: Bit error rate (BER) v. received signal-to-noise ratio for a DQPSK transmitter-receiver pair in a quiet network.

To simulate links with varying amounts of path loss, we send a stream of packets between the two radios, modulating the transmit power of the stream of packets (we hold transmit power constant for the duration of each packet). At the receiver, we calculate the average received SNR for each packet and check the correctness of each bit in the packet. Figure 19 shows the resulting BER-SNR curve. We note the high BER for relatively-high SNR, hypothesizing that better clock-recovery algorithms and of course coding would shift the curve left as is commonly seen in commercial radio receivers.

We are now in a position to examine SoftPHY hints at low SNRs. We partition the data into "good," "grey-zone," and "bad" transmissions according to average BER, as in Table 2. Figure 20 shows the distribution of SoftPHY hints in each regime. We see that SoftPHY hints are a good predictor of symbol correctness in the good regime, but an increasingly poorer predictor of symbol correctness

Label	SNR	BER
Good	SNR $\geq$ 21	BER $\leq 10^{-3}$
Grey-zone	13 $<$ SNR $<$ 21	$10^{-3} <$ BER $< 10^{-2}$
Bad	SNR $\leq$ 13	BER $\geq 10^{-2}$

Table 2: Regimes for evaluating SoftPHY at marginal SNR.

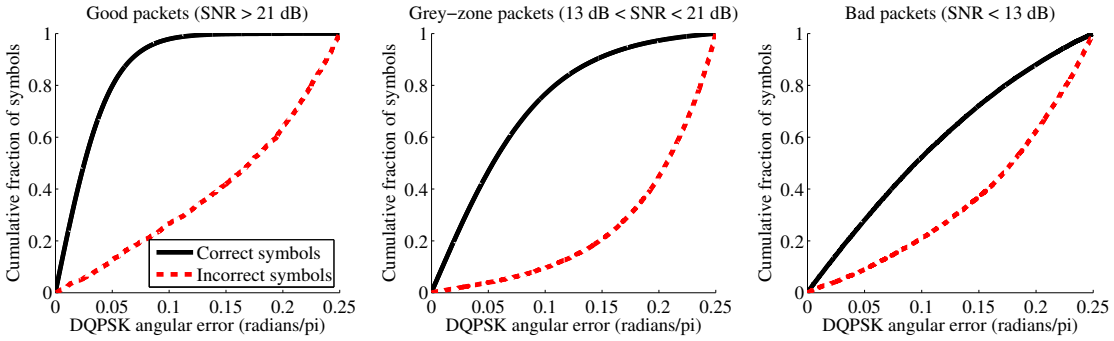

Figure 20: The effects of marginal SNR on SoftPHY hints with an uncoded DQPSK receiver in a quiet network.

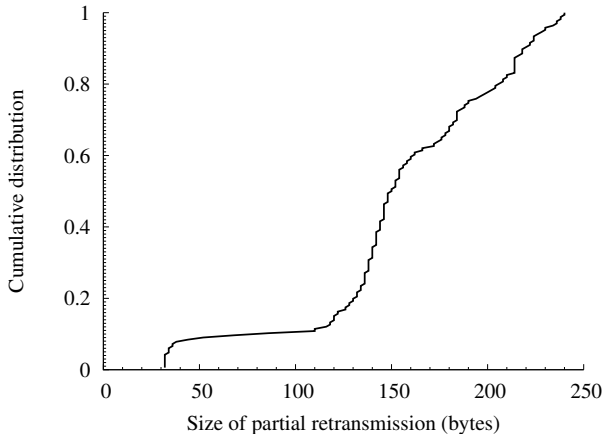

Figure 21: Packet sizes of partial retransmissions between a pair of nodes transferring data with PP-ARQ. Here each packet size is 250 bytes.

as SNR decreases, as expected. We note that the SoftPHY hint we use here, per-symbol angular difference from the hard decision, is based on an uncoded modulation considering each symbol independently, and we have achieved significantly better results in coded modulations (see Figure 5). These results illustrate the diminishing utility of SoftPHY hints as SNR decreases, and provide a proof-of-concept in an alternative modulation.

7.5 PP-ARQ

We now present some preliminary results from our real-time implementation of PP-ARQ. In this single-link experiment, one software radio-based DSSS transmitter sends 250 byte data packets back-to-back to a software radio-based receiver.

Figure 21 shows the sizes of each retransmission packet that the PP-ARQ sender sent to the PP-ARQ receiver. We see that even in this implementation, which has yet to be performance-tuned, the median retransmission size is approximately half the full packet size. This implies that in a busy network, PP-ARQ may need to retransmit at most half the data, on half the total retransmissions.

8. RELATED WORK

While each of the three ideas in PPR—SoftPHY, postamble decoding, and PP-ARQ—is novel, as is their synthesis into a single system, these individual ideas are related to and inspired by much previous work. We survey closely related work in this section.

Physical-layer algorithms for sequence decoding can yield decision confidences to higher layers in several ways. The BCJR algorithm [6] yields a posteriori probabilities directly, with only a modest increase in complexity from the Viterbi alogorithm. The Viterbi algorithm has itself been extended with decision confidence outputs; the resulting "soft-output" Viterbi algorithm (SOVA) [16] or BCJR may be used in a Turbo decoder [8]. The former work proposes the use of SOVA in the decoding of outer codes and is contained within the physical layer, whereas our focus is on constructing and propagating hints to higher layers. The two techniques are complementary, because layered coding could be used in conjunction with SoftPHY as long as the outer-layer code outputs a confidence metric.

Rate selection algorithms have been extensively studied in 802.11 wireless networks [9, 18, 28]. As with adjusting the amount of coding on a wireless link, it is hard to predict how much redundancy a link will need in highly-variable conditions. PPR mitigates the need for choosing the correct rate by allowing receivers to recover partially-received frames and efficiently retransmit only the parts missing. Moreover, SoftPHY hints can potentially be used to perform rate adaptation at finer time-scales than before, because it is now possible for the MAC layer to estimate the symbol error rate for different rates and modulations more directly than before.

Ahn et al. [4] propose an adaptive FEC algorithm which dynamically adjusts the amount of FEC coding per packet based on the presence or absence of receiver acknowledgements.

Hybrid ARQ is a way of combining FEC and ARQ. Type I hybrid ARQ schemes [19] retransmit the same coded data in response to receiver NACKs. Chase combining [11] improves on this strategy by storing corrupted packets and feeding them all to the decoder. Type II hybrid ARQ schemes [19] forego aggressive FEC while the channel is quiet, and send parity bits on retransmissions, a technique called incremental redundancy [21]. Metzner [22] and later Lin and Yu [20] have developed type II hybrid ARQ schemes based on incremental redundancy.

PP-ARQ takes a different approach from the above work: instead of using stronger codes for the entire packet on retransmit, it uses hints from the physical layer about which codewords are more likely to be in error, and retransmits just those codewords.

Techniques such as coding with interleaving [7, Chp. 12] spread the bursts of errors associated with collisions and deep fades across many codewords so that they can be corrected. This technique is complementary to partial packet recovery, but not easy to implement, because it is necessary to know the channel conditions a priori in order to provision the amount of coding required to acheieve high throughput.

Whitehouse et al. [34] and Priyantha [27] propose avoiding "un-

desirable capture" in packet-based wireless networks in time. Undesirable capture occurs when the weaker of two packets arrives first at a receiving node, so that the stronger, later packet corrupts the weaker, earlier packet, resulting in neither being decoded correctly. This can be viewed as a special case of postamble decoding, which we fully explore in the present work.

9. CONCLUSION

We believe that PPR has the potential to change the way PHY, link, and MAC protocol designers think about protocols. Today's wireless PHY implementations employ significant amounts of redundancy to tolerate worst-case channel conditions. If noise during one or more codewords is higher than expected, existing PHY layers will generate incorrect bits, which will cause packet-level CRCs to fail and require retransmission of the whole packet. Since noise fluctuations are often large, and the penalty for incorrect decoding is also large, PHY layers tend to conservatively include lots of redundancy in the form of a high degree of channel coding or conservative modulation. Similarly, MAC layers tend to be quite conservative with rate adaptation because the consequences of errors are considered dire. The mind-set seems to be that bit errors are bad, and must be reduced (though eliminating them is impossible). As a result, they operate with comparatively low payload bit-rates.

PPR reduces the penalty of incorrect decoding, and thus for a given environment allows the amount of redundancy to be decreased, or equivalently the payload bit-rate to be increased. Put another way, with SoftPHY and PPR, it may be perfectly fine for a PHY to design for one or even two orders-of-magnitude higher BER, because higher layers need no longer cope with high packet error rates, but can decode and recover partial packets correctly.

In addition to investigating the above idea, our plans for future work include implementing other ways of obtaining confidence information (as outlined in Section 2, developing a PHY-independent SoftPHY interface and showing how a PP-ARQ link layer can use different SoftPHY implementations without change, performing a broader set of experiments in more settings, and using SoftPHY information to improve the performance of routing protocols such as opportunistic routing [10].

Acknowledgments

We thank Bret Hull, Robert Morris, our shepherd Stefan Savage, David Wetherall, and the anonymous reviewers for their constructive feedback.

10. REFERENCES

[1] Boulder, CO, Nov. 2006.
[2] Wireless LAN Medium Access Control and Physical Layer Specifications, August 1999. IEEE 802.11 Standard.
[3] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-Level Measurements From an 802.11b Mesh Network. In *Proc. ACM SIGCOMM Conf.*, pages 121–132, Portland, OR, Aug. 2004.
[4] J.-S. Ahn, S.-W. Hong, and J. Heidemann. An Adaptive FEC Code Control Algorithm for Mobile Wireless Sensor Networks. *Journal of Communications and Networks*, 7(4):489–499, 2005.
[5] A. Avudainayagam, J. M. Shea, T. F. Wong, and L. Xin. Reliability Exchange Schemes for Iterative Packet Combining in Distributed Arrays. In *Proc. of the IEEE WCNC Conf.*, volume 2, pages 832–837, Mar. 2003.
[6] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate. *IEEE Trans. Info. Theory*, 20(2):284–287, Mar. 1974.
[7] D. Barry, E. Lee, and D. Messerschmitt. *Digital Communication*. Springer, New York, NY, 3rd. edition, 2003.
[8] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes. In *Proc. of the IEEE ICC Conf.*, pages 54–83, May 1993.
[9] J. Bicket. Bit-Rate Selection in Wireless Networks. Master's thesis, Massachusetts Institute of Technology, Feb. 2005.
[10] S. Biswas and R. Morris. ExOR: Opportunistic Multi-hop Routing for Wireless Networks. In *Proc. ACM SIGCOMM Conf.*, pages 133–144, Philadelphia, PA, Aug. 2005.
[11] D. Chase. Code Combining: A Maximum-Likelihood Decoding Approach for Combining an Arbitrary Number of Noisy Packets. *IEEE Trans. on Comm.*, 33(5):385–393, May 1985.
[12] S. Eisenman and A. Campbell. Structuring Contention-Based Channel Access in Wireless Sensor Networks. In *Proc. of ACM/IEEE IPSN Conf.*, pages 226–234, Nashville, TN, Apr. 2006.
[13] M. Ettus. Ettus Research, LLC. http://www.ettus.com.
[14] R. Ganti, P. Jayachandran, H. Luo, and T. Abdelzaher. Datalink Streaming in Wireless Sensor Networks. In *Proc. of the SenSys Conf.* [1], pages 209–222.
[15] The GNU Radio Project. http://www.gnu.org/software/gnuradio/.
[16] J. Hagenauer and P. Hoeher. A Viterbi Algorithm with Soft-Decision Outputs and its Applications. In *Proc. of the IEEE GLOBECOM Conf.*, Dallas, TX, Nov. 1989.
[17] F. Harris. *Multirate Signal Processing for Communication Systems*. Prentice Hall PTR, Upper Saddle River, NJ, 2004.
[18] G. Holland, N. Vaidya, and P. Bahl. A Rate-Adaptive MAC Protocol for Multihop Wireless Networks. In *Proc. ACM MobiCom Conf.*, pages 236–251, Rome, Italy, July 2001.
[19] S. Lin and D. J. Costello. *Error Control Coding*. Prentice Hall, Upper Saddle River, NJ, 2nd. edition, 2004.
[20] S. Lin and P. S. Yu. A Hybrid ARQ Scheme with Parity Retransmission for Error Control of Satellite Channels. *IEEE Trans. on Comm.*, 30(7):1701–1719, July 1982.
[21] D. Mandelbaum. An Adaptive-Feedback Coding Scheme Using Incremental Redundancy (Corresp.). *IEEE Trans. on Information Theory*, 20(3):388–389, May 1974.
[22] J. Metzner. Improvements in Block-Retransmission Schemes. *IEEE Trans. on Comm.*, 27(2):524–532, Feb. 1979.
[23] A. Miu, H. Balakrishnan, and C. E. Koksal. Improving Loss Resilience with Multi-Radio Diversity in Wireless Networks. In *Proc. MobiCom Conf.*, pages 16–30, Cologne, Germany, Aug. 2005.
[24] K. Mueller and M. Müller. Timing Recovery in Digital Synchronous Data Receivers. *IEEE Trans. on Comm.*, 24(5), May 1976.
[25] S. Pasupathy. Minimum Shift Keying: A Spectrally-Efficient Modulation. *IEEE Communications Magazine*, 7(4):14–22, July 1979.
[26] M. Perrott. The CppSim Behavioral Simulator. http://www-mtl.mit.edu/researchgroups/perrottgroup/tools.html.
[27] N. B. Priyantha. *The Cricket Indoor Location System*. PhD thesis, MIT, May 2005.
[28] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly. Opportunistic Media Access for Multirate Ad Hoc Networks. In *Proc. ACM MobiCom Conf.*, pages 24–35, Atlanta, GA, Sept. 2002.
[29] T. Schmid. Personal communication.
[30] R. Sinha, C. Papadopoulos, and J. Heidemann. Internet Packet Size Distributions: Some Observations. http://netweb.usc.edu/~rsinha/pkt-sizes.
[31] D. Son, B. Krishnamachari, and J. Heidemann. Experimental Analysis of Concurrent Packet Transmissions in Low-Power Wireless Networks. In *Proc. of the SenSys Conf.* [1], pages 237–250.
[32] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis. Understanding the Causes of Packet Delivery Success and Failure in Dense Wireless Sensor Networks. Technical Report SING-06-00, Stanford Univ., 2006.
[33] TI/Chipcon Products CC2420 Data Sheet. http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf.
[34] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the Capture Effect for Collision Detection and Recovery. In *IEEE EmNets Workshop*, Sydney, Australia, May 2005.
[35] T. F. Wong, L. Xin, and J. M. Shea. Iterative Decoding in a Two-Node Distributed Array. In *Proc. of the IEEE MILCOM Conf.*, volume 2, pages 1320–1324, Oct. 2002.