Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2007-008 February 2, 2007

PPR: Partial Packet Recovery for Wireless Networks

Kyle Jamieson and Hari Balakrishnan

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

PPR: Partial Packet Recovery for Wireless Networks

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

Abstract

Bit errors occur over wireless channels when the signal isn't strong enough to overcome the effects of interference and noise. Current wireless protocols may use forward error correction (FEC) to correct for 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 three new ideas: (1) *SoftPHY*, an expanded physical layer (PHY) interface to provide hints to the higher layers about how "close" the actual received symbol was to the one decoded, (2) a *postamble* scheme to recover data even when a packet's preamble is corrupted and not decodable at the receiver, and (3) *PP-ARQ*, an asynchronous link-layer retransmission protocol that allows a receiver to compactly encode and request for retransmission only those portions of a packet that are likely in error.

Our experimental results from a 27-node 802.15.4 testbed that includes Telos motes with 2.4 GHz Chipcon radios and GNU Radio nodes implementing the Zigbee standard (802.15.4) show that PPR increases the frame delivery rate by a factor of 2× under moderate load, and 7× under heavy load when many links have marginal quality.

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. Poor SINR is usually the result of noise caused by sources external to the network, and interference caused by one or more other concurrent transmissions in the network, and varies in time even within a single packet transmission. Even with a variety of physical layer (PHY) techniques such as soft-decision decoding [10], spread spectrum modulation, channel coding, and the like, current systems rely heavily on link-layer retransmissions to recover from bit

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

errors and achieve high throughput. Because 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. Link-layer 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). Over radio, however, all the bits in a packet don't share the same fate: wery often, only a small number of bits in a packet are in error and others are correct. Thus, it is wasteful to re-send the entire packet when only a small part of it is lost. Our goal is to eliminate this waste, ideally by never re-sending those portions of the packet that have already been received correctly.

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 PHY schemes 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 bit transmissions. It incorporates three new techniques to meet the challenges mentioned above:

1. *SoftPHY* allows a receiver to determine, with no additional feedback or information from the sender, which groups of bits (*symbols*) are likely to be correct in any given packet reception using hints from the PHY. The key insight in SoftPHY is that the PHY maintains information about how close the received symbol was to what it decoded and reported to the higher layer. The higher layer can use this information, propagated from the PHY, as a hint. Section 3 develops this idea further.
2. *Postamble decoding* allows a receiver to receive and decode bits correctly even from packets whose preambles are corrupted. The main idea here is to replicate the preamble and packet headers in a postamble and a packet trailer, allowing a receiver to lock on the postamble and then "roll back" in time to recover data that was previously impossible to obtain. Section 4 describes the details of this scheme.
3. *PP-ARQ*, a link-layer retransmission protocol in which the receiver compactly requests and encodes the retransmission of only select portions of a packet. This compact encoding is done using 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. Section 5 describes PP-ARQ.

We have implemented all three components on the GNU Radio platform for 802.15.4, the Zigbee standard. Our implementation is compatible with that specification (see Section 6). The SoftPHY and postamble decoding steps running at the receiver can recover partial packets from unmodified Zigbee senders, while PP-ARQ requires modifications to the sender as well.

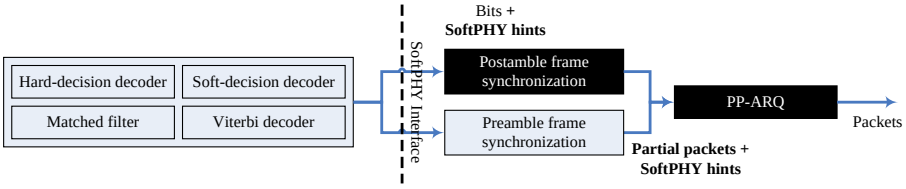

Figure 1: Block diagram of the PPR system; dark blocks and the SoftPHY interface are the contributions of this paper. PPR sits above one of many different types of receiver structure, as discussed in Section 3. The SoftPHY interface passes up hints from the receiver to PP-ARQ, the partial packet retransmission layer, described in Section 5. Postamble decoding, described in Section 4, increases the number of opportunities for recovering partial packets from the receiver.

We have conducted several experiments over a 25-node indoor testbed consisting of Telos motes with 2.4 GHz Zigbee radios from Chipcon and four GNU Radio nodes. These results, described in Section 7, show factor-of-two gains in the packet delivery rate and per-link throughput with PPR over the status quo under moderate load. We also compare PPR to other ways of determining which bits in a packet are likely to be correct, such as fragmented CRCs on packets. There 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×. These gains are even higher (7× better per-link throughput) under heavy load, which causes a number of links to have marginal quality.

2 Design Overview

The PPR system allows the link/MAC layer to estimate which bits of any given reception are likely to be correct and use that information to improve packet delivery rates and throughput. To understand how its three components (SoftPHY, postamble decoding, and PP-ARQ) work and interface with each other, a conceptual model of the wireless communication system will be helpful. This model is intentionally simple and doesn't capture the breadth of designs where PPR can be useful, but describes a typical instance of a direct-sequence spread spectrum (DSSS) radio. This model applies to both 802.15.4 (Zigbee) and 802.11b/g (WiFi), two common standards.

At the sender, the network layer passes packets to the link/MAC layer, which places a header that includes the link-layer source address, link-layer destination address, and a CRC covering the entire link-layer packet's contents at the end of the packet.[1] The link layer then passes a stream of bits to the PHY.

There are many ways to implement a DSSS sender. In one approach, the PHY maps groups of b bits to a B-bit *codeword*. In Zigbee, for instance, $b = 4$ and $B = 32$. Since there are only 2^b b-bit strings, the space of valid codewords is extremely sparse. Each codeword (actually, the sequence of codewords) is broken up into groups of $k \geq 1$, each

[1] Later, we will investigate the performance of fragment-CRC schemes that place multiple CRCs in the packet.

group mapping to a *channel symbol* or *chip*, and sent over the air modulated over some waveform (in Zigbee, $k = 1$). Thus, each bit in the original packet gets spread over many chips.

At the receiver, in current systems, the PHY produces a sequence of bits after demodulating incoming waveforms and decoding symbols. This interface is rather limiting in that the link/MAC layer has no easy way of telling which bits are almost certainly correct, and which bits are more questionable. The PHY, however, has this information; for example, when decoding received waveform samples to a codeword, it knows how close the actual reception was to the closest valid codeword, and can propagate that information up as a confidence. If it uses a maximum-likelihood estimator in demodulation, then it can propagate the likelihood as a confidence too. We use this insight in developing PPR's SoftPHY interface, shown in Figure 1 and described in detail in Section 3.

In the status quo, no processing of a packet payload can occur without the successful receipt of a preamble. Unfortunately, when collisions occur, the preamble is often destroyed. Our postamble decoding scheme, shown in Figure 1 and described in Section 4, overcomes this problem by allowing a receiver to recover data even when the preamble is lost. We also replicate the information in the header in a trailer at the end of the packet. Figure 2 shows the layout of the information in a typical PPR packet.

Finally, once SoftPHY propagates confidence information together with each group of bits to the link/MAC layer, there are several ways to use it. We develop PP-ARQ, a retransmission protocol where the receiver sends a request to the sender to re-send only those bit ranges where there are several bits likely wrong. In response, the sender retransmits the bits from those ranges and sends CRC values for the other ranges, so that the receiver can be certain that the bits in the non-retransmitted portions are correct. Other ways to use SoftPHY information include integrating it with forwarding protocols or opportunistic routing protocols, forwarding only the bits likely to be correct. We do not investigate that possibility in this paper, focusing instead on improving ARQ (see Section 5).

The underlying premise in PPR is that significant performance gains can be obtained by being a little more flexible about the granularity of error recovery in wireless networks. A finer granularity than packets can help improve performance in both access point-based networks and wireless mesh networks. Section 7 shows several experimental results that confirm this premise.

3 SoftPHY

In the abstract, the SoftPHY interface is simple: annotate each group of bits propagated to the higher layer with a *confidence* metric. The details of how the confidence metric is calculated and what it actually means, however, depend on how the PHY works. We believe, however, that most demodulation and decoding methods maintain this information. For three common designs, we outline what this information would be, then discuss the method we implemented, and finally discuss how higher layers should be designed to interpret these hints, given that what the information means depends on what the PHY does.

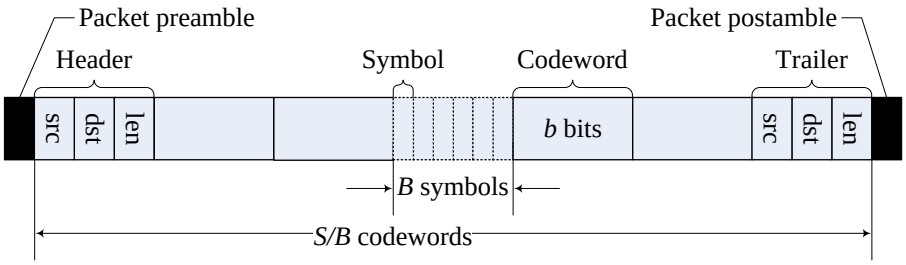

Figure 2: A packet is composed of *symbols*, which may be organized into *codewords* of B symbols each by channel coding or direct-sequence spread spectrum. Each codeword encodes b data bits.

3.1 Options for PHY Hints

At a high level, the PHY generally includes a demodulator and a channel decoder. We propose three ways of obtaining PHY hints that apply to three popular designs.

Hamming distance in hard-decision decoding. In one common PHY design, the demodulator makes a *hard decision* [10, Chp. 8] of what symbol any given sample waveform corresponds to (and therefore what group of $k \geq 1$ bits they correspond to), independent of other receptions. It then sends that information to the channel decoder, which maps the received word to the closest (most likely) codeword. The closeness of this mapping, measured as the Hamming distance between the received work and the codeword (the number of distinct elements between the two words), can serve as a useful confidence hint: a low distance inspires confidence in the correctness of the bits, while a large one does not.

Correlation metric in soft-decision decoding. To cope better with Gaussian noise, a more efficient option than hard decoding is to use *soft-decision decoding* (SDD) [10, Chp. 8].[2] Here, the demodulator passes samples of received symbols. The decoder then makes codeword decisions on groups of received symbols using a correlation metric that takes into account these samples. However, SDD will still produce incorrect codewords if the SINR is low, and does not recover correct bits in the presence of packet collisions particularly well.

With SDD, the receiver calculates a *correlation metric* C between the received samples R and each codeword C_i (whose jth bit is c_{ij}) defined as [10, Ch. 8]:

$$C(R, C_i) = \sum_{j=1}^{n} \left(2c_{ij} - 1\right) r_{ij} \tag{1}$$

This metric can serve as a useful hint from the PHY to higher layers: a larger metric inspires confidence that the bits are correct.

We also note that a particularly interesting instance of a confidence metric when convolutional decoding is used with soft-decision decoding is to use the output of the Viterbi decoder [11]. This output is the likelihood of is a measure of how well the

[2] SDD improves BER by 2 to 3 dB over HDD in the presence of noise.

received sequence of codewords matched with the path through the coding trellis associated with the decided-upon codeword.

Matched filter demodulator output. PHY hints can also be obtained from the demodulator itself in the absence of any channel coding. The receiver structure for optimal detection of signals over an additive white Gaussian noise (AWGN) channel is a filter matched to the shape of the incoming signal [10, Ch. 5]. The output of this matched filter is the correlation function between the received signal and the decided-upon codeword. The SoftPHY hint is then simply the output of the matched filter; a larger value signifies higher confidence.

3.2 Hamming Distance as a Hint

We conducted preliminary experiments with the HDD and SDD schemes described above. We found that our bit errors were mostly due 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). Here, we give a brief summary of one experimental result that shows that the Hamming distance is a good SoftPHY hint.

This experiment was done over a 27-node testbed of 802.15.4 Chipcon and GNU Radios described in Section 7. 23 of the nodes send packets whose payload is a known test pattern at a constant rate. There are four receivers (each receiver is able to hear and decode some subset of the 23 senders). Figure 3 shows the distribution of Hamming distance across each received codeword, demarcated by whether the codeword was correctly or incorrectly received (we know this from the test pattern). Conditioned on a correct decoding, 96% of codewords have a Hamming distance of 1 or less. In contrast, barely 10% of the incorrect codewords have a distance of 6 or less.

This result shows that the higher layer must actually 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". This graph, and later results in Section 7.4, show that a large Hamming distance is an excellent indication of symbol incorrectness, while a small Hamming distance is a good predictor of codeword correctness.

3.3 Architectural Implications of SoftPHY Hints

Although the SoftPHY interface can be provided for a variety of PHY implementations, the semantics of the delivered information is intimately tied to the details of the PHY. One of the benefits of protocol layering is that higher layers are shielded from the details of the lower ones, and SoftPHY has the potential danger of violating this abstraction barrier in the interest of performance. Here, we argue that it is possible to implement higher layers without violating the abstraction boundary.

To do that, the higher layers must not be aware of exactly how the information is calculated. Instead, they should adapt their threshold or other decisions to label groups of bits as "good" or "bad" to actual observation. For example, the link/MAC layer could observe the correlation between a particular threshold and the correctness of the hint,

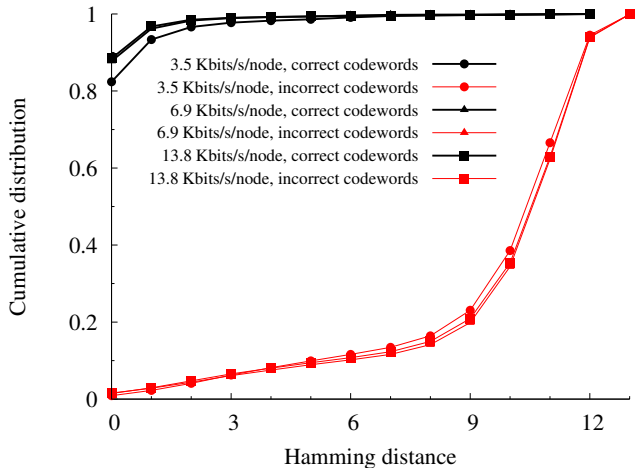

Figure 3: The distribution of hamming distances for every codeword in every received packet, separated by whether the received codeword was correctly or incorrectly decoded. For correctly-decoded codewords, hamming distance is low; for incorrectly-decoded codewords, hamming distance is high.

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 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). Implementing such a contract is straightforward for the PHY, at least for the three options we laid out above.

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 doesn't even attempt to make a decoding decision, instead propagating all received signal samples up to the decoder. In contrast, with our SoftPHY design, the PHY doesn't simply pass up all its raw information to the higher layer, so layering boundaries are preserved and the PHY still makes "hard" decisions. However, the PHY does pass hints upwards about its confidence in each decision it makes. This architecture preserves a clean decomposition between PHY and higher layers, while enabling performance gains.

3.4 An Alternative to SoftPHY: Per-Fragment CRC

We will show later that SoftPHY improves performance significantly, but one might ask whether it is *necessary* to achieve similar gains. One cannot ignore the benefits of the boundaries between the PHY and higher layers in the status quo, so it is important that we investigate how necessary SoftPHY hints are, in addition to how much they improve performance. One way to approximate SoftPHY is to send multiple CRCs per packet, one per fragment of the packet, as shown in Figure 4. This scheme allows

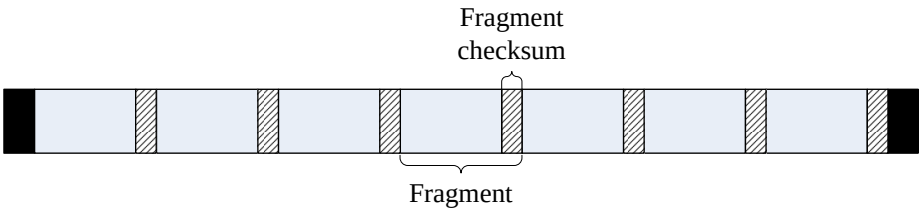

Figure 4: Multiple CRC checksums sent per packet, with each CRC taken over a different fragment of the packet.

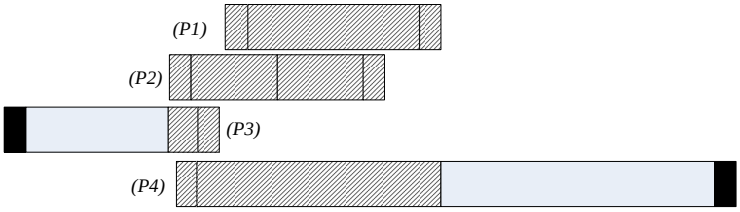

Figure 5: A collision between many packets. We use the postamble to partially decode packets *P3* and *P4*, and the techniques described in Sections 3 and 5 to detect the incorrect parts of each packet and request retransmission of just those parts.

the receiver to identify entire fragments that are correct. If bit errors are concentrated in only a few bursts, then entire fragments will check out 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 CRC 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 Section 7, we investigate the "best case" for CRC fragments, finding *post facto* from traces of errored and error-free symbols what the optimal fragment size is and using that value.

4 Decoding Packets Without a Preamble

When errors occur in the preamble due to collisions or noise, the receiver will not be able to synchronize to the incoming transmission and decode any bits. In that case, the benefits of SoftPHY will go unrealized. We need a way to cope with preamble loss, for example, in the multi-packet collision in Figure 5. In this example, packet *P4* would not be received at all, because its preamble would not have been decoded correctly, while packet *P3* would be received only partially.

Our approach to synchronizing on packets without an intelligible preamble is simple: add a *postamble* to the end of each packet on which a receiver can also synchronize. Like the preamble, the postamble has a well-known sequence attached to it that uniquely identifies it as the postamble, and differentiates it from a preamble. In addition, we add a *trailer* near the postamble at the end of the packet, as shown in Figure 2. The trailer contains the packet length, source, and destination addresses. Just as with the header data, the receiver uses a CRC fragment or PHY-layer confidences to check the correctness of the trailer so that it can request a partial retransmission from the sender, as described in Section 5.

To recover payload symbols (and bits) 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 of postamble decoding in MSK, we keep as many samples of previously-received symbols as there are symbols in one maximally-sized packet in the system. When the receiver detects a preamble, the behavior is the same is in the status quo. When 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, source, and destination addresses.
3. Verify the trailer's checksum.
4. "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, adaptive equalization, automatic gain control (AGC), and symbol timing recovery. We meet each of these challenges in our implementation, as briefly outlined below.

To receive a packet correctly, the demodulator may[3] need to perform carrier recovery [15, Chp.14] to estimate the incoming carrier's frequency and phase. In our MSK implementation, there is no need to perform carrier recovery. For other modulations, preamble-less non-decision-aided techniques for carrier recovery in 16-QAM have also been proposed [8].

A number of techniques for countering inter-symbol interference rely on estimating the channel impulse response, a process called *adaptive equalization* [15, Chp. 9]. Typically the preamble includes a known *training sequence* to enable the adaptive equalizer to acquire the signal. We therefore include the same training sequence in the postamble and post-process the samples of the signal in the body of the packet afterwards. Modulations needing automatic gain control such as MQAM [10, Chp. 6] can utilize similar methods.

Finally, most, but not all demodulators need to perform *symbol timing recovery* [15, Chp. 15] to determine when to sample the incoming signal such that each symbol is correctly decoded. In our system, we use a non data-aided symbol timing recov-

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

ery [21] which permits synchronization at any time during a transmission, allowing us to symbol-synchronize the stored samples without having already heard the postamble.

5 PP-ARQ: Retransmissions Using Partial Packet Recovery

SoftPHY and the postamble scheme 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 should most efficiently communicate this information back to the sender, to form the full partial packet recovery protocol.

The naive way to provide feedback is for each receiver to send back the bit ranges of each chunk believed to be wrong. Unfortunately, doing that takes up a large number of bits, since encoding the start of a range and its length can take up between $\log S$ and $2 \log S$ bits for a packet of size S (depending on the details of the encoding; compression might improve this number, but it is still potentially large). Hence, we need to develop an bit-efficient feedback scheme.

After the receiver has decoded a packet, it has a list of received symbols S_i, $1 \leq i \leq N$, 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 computes alternating *run lengths* λ_j^g, λ_j^b, $1 \leq j \leq L$ of good and bad symbols, respectively, to form the *run-length representation* of the packet as shown in Figure 6. 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's job is to decide which runs to ask the transmitter to re-send. Once the receiver has made that choice, it sends a *feedback packet* to the transmitter communicating this information. If there are many short runs of incorrect bits, encoding their positions will be expensive, and the receiver should ask for one large run containing all the incorrect bits. If, on the other hand, there are relatively-few long runs of incorrect bit, the receiver should ask for each and every long run individually.

This problem can be solved using dynamic programming. We show that any set of bad runs the receiver asks for can be assigned a cost function, and that the problem itself exhibits the "optimal substructure" property in that the cost for the entire packet is easily derived from the cost of two suitably divided portions. The result is a series of segments that the receiver requests the sender to retransmit. Each of those segments will of course have "bad" codewords in them, but may also have some "good" codewords (for otherwise there might have been too many segments to ask for). On the other hand, no segment that is not asked for will have any "bad" codewords. When the sender responds to this carefully constructed requests for segment retransmissions, it also sends the CRCs of the segments not asked for, so that the receiver can verify that all those codewords have been correctly recovered.

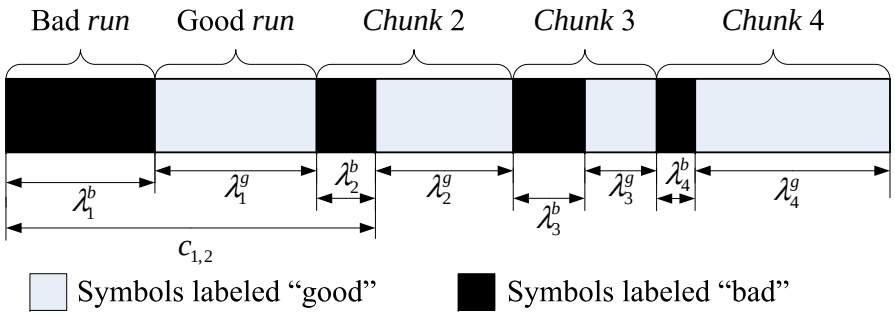

Figure 6: After computation of *run-length representation* of a received packet, the first step in PP-ARQ at the receiver. Run lengths $\lambda_i^{b,g}$ are as defined in expression 2. $c_{1,2}$ refers to the chunk as defined in expression 3.

5.1 Dynamic programming to find the best retransmission strategy

The receiver forms a list of "chunks" $c_{i,j}$: groups of runs that it will eventually 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, chunk $c_{1,2}$ appears in Figure 6. Note that chunk $c_{i,j}$ does not include λ_j^g, the last run of good symbols in the chunk.

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

If λ_k^g, $i \leq k \leq j$ are all small and $j - i$ is large, we would favor requesting that chunk $c_{i,j}$ be retransmitted over chunks $c_{k,k}$ for each and every $i \leq k \leq j$, because the additional bits it would take for the receiver to describe each of the $j - i$ individual chunks would exceed that needed to retransmit the good symbols associated with those chunks. 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 the same reasons.

We assume that the receiver must send a checksum of every good run to the sender, to verify that the bits are correct. We also assume that the receiver will ask the sender to re-transmit at least the bad runs in the packet.

We define the *cost* of a chunk as follows:

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

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

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 outermost 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 can 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. The results in an $O(L^3)$ implementation.

5.2 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 a checksum appended.
2. The receiver decodes the packet (possibly partially), and computes the best feedback set of chunks as described in Section 5.1.
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 those runs* (and their associated CRCs and byte offsets) *that the receiver has requested*.

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.

6 Implementation

Each of the receivers in the following experiments is a computer connected to a GNU Radio [9] software-defined radio. We deployed four receivers among the senders, as shown in Figure 7. The hardware portion of the receiver is a GNU Radio *Universal Software Radio Peripheral* (USRP) with a 2.4 GHz *RFX2400* daughterboard; the remainder of the receiver's functionality is implemented in software. The DSSS despreading functionality was written in C++ by us, with parts derived from code written by Schmid [25]. The preamble and postamble frame synchronization was also implemented in C++.

The sender node Chipcon CC2420 radio [28] is a 2.4 GHz RF transceiver that uses O-QPSK modulation with half sine pulse shaping, also known as min-shift keying (MSK) [22]. The CC2420 uses direct-sequence spread spectrum (DSSS) at a rate of 2 Msymbols/s with $B = 32$ symbol codewords.[4] 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.

6.1 PP-ARQ

We implemented PP-ARQ in Python, above the SoftPHY interface. At the receiver, our implementation parses the received SoftPHY hints, computes the run-length representation of the packet as defined in Equation 2, runs the dynamic programming algorithm described in Section 5.1 on the run-length representation of the packet, and sends a feedback packet to the sender summarizing which runs need retransmitting.

At the sender, our 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. We begin with an evaluation of the predictive power of SoftPHY's PHY-level hints proposed in Section 3. Next we present channel capacity results evaluating the SoftPHY interface and a postamble-enabled physical layer (Section 4). We conclude our evaluation with results combining the above two techniques with PP-ARQ, the partial packet recovery ARQ method described in Section 5. We summarize our experiments and findings in Table 1.

7.1 Experimental method

Each of the senders in the following experiments is a moteiv tmote sky wireless sensor node, equipped with a TI/Chipcon CC2420 radio. We deployed 23 sender nodes over nine rooms in an indoor office environment, as shown in Figure 7. All sender nodes are connected to tmote connect gateways, which interface the senders to a central control computer.

7.2 Partial packet recovery in a busy network

We now present channel capacity results evaluating how well the combination of SoftPHY (with the Hamming distance hint described in Section 3) and postamble decoding performs against the fragmented CRC scheme described in Section 3.4. In this experiment each node sends a stream of bits, which are formed into traces and post-processed to emulate a packet size of 1500 bytes. Summarizing each scheme:

[4] We use the notation defined above in Section 2.

Experiment	Section	Conclusion
PPR capacity	7.2	PPR and fragmented CRC both improve per-link throughput over the status quo by more than 7× under high load, and 2× under moderate load; PPR improves capacity even more than fragmented CRC, by a factor of 2× under high load and 1.6× under moderate load.
Anatomy of a collision	7.3	PPR can recover partial packets even in the presence of uncertainty in PHY symbol-timing recovery.
SoftPHY hints in a busy network	7.4	The pattern of "misses" and "false alarms" under the SoftPHY hints we propose enable partial packet recovery in a busy network.
PP-ARQ	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 7: Experimental testbed layout: there are 27 nodes in total, spread over nine rooms in an indoor office environment. Each dot indicates an 802.15.4 node. GNURadio nodes are labeled $R1$ through $R4$.

Number of chunks	Aggregate throughput (Kbits/s)
1	26
10	85
30	**96**
100	80
300	15

Table 2: Fragmented CRC end-to-end aggregate throughput in our network as a function of chunk size.

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 above in Section 3.4, breaks the packet into chunks, each followed by an associated 32-bit CRC over the preceeding chunk. Fragmented CRC delivers each chunk whose checksum verifies correctly, and discards the remainder.
3. *PPR* delivers exactly those bits in the packet whose codewords had a Hamming distance less than η. Here we choose $\eta = 6$.

7.2.1 Choosing fragmented CRC chunk size

To find the optimal chunk size for the fragmented CRC scheme, we ran experiments comparing aggregate throughput as chunk size varied. The results are shown in Table 2. We see that when chunk size is small, checksum overhead dominates; while large chunk sizes lose throughput because collisions and interference wipe out entire chunks. Based on these results, we picked a chunk size of 50 bytes (corresponding to 30 chunks per packet) for the following experiments.

7.2.2 Equivalent frame delivery rate

We first examine the rate at which each scheme described above delivers bits to higher-layers, once it has succesfully acquired a packet. We term this rate the *equivalent frame delivery rate*, because it measures how efficient each scheme is at delivering packets to higher layers once the PHY layer synchronizes on a packet, either by the presence of a preamble, or by the presence of a postamble.

In the following experiments we measure the amount of traffic each of the 23 CC2420 senders shown in Figure 7 can deliver to the four GNU Radio receivers. In the absence of any other traffic, each sink had between 4 and 8 sender nodes that it could hear, with the best links having near perfect delivery rates. When multiple nodes transmit at the same time, SINR reduces and link quality falls, as shown in the delivery-rate graphs below.

Figure 8 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). In this experiment, the CC2420 senders perform a carrier sense before transmitting each packet. We see that under all partial packet recovery schemes, postamble decoding increases median

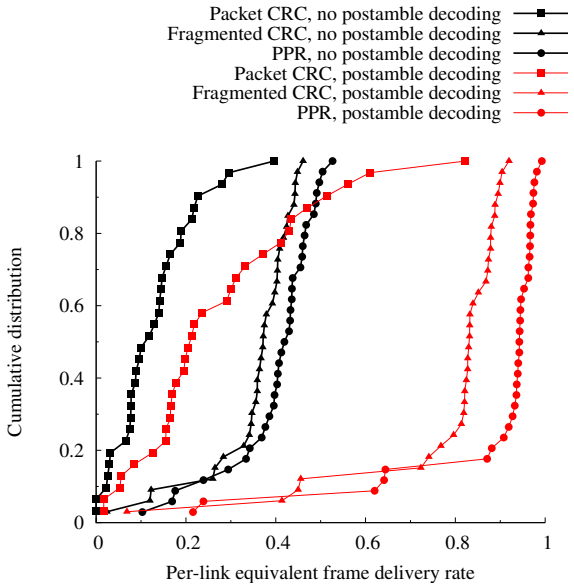

Figure 8: Per-link equivalent frame delivery rate with carrier sense enabled, at moderate offered load (3.5 Kbits/s/node).

frame delivery rate by a factor of two, probably because it decreases the probability of missing an opportunity to synchronize on an incoming transmission. Even when carrier sense and postamble decoding are enabled, we see relatively poor links in the network.

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.

We now repeat the experiment with carrier sense disabled; Figure 9 shows the results. When carrier sense is disabled, at least a small part of the packet is likely to be decoded incorrectly, resulting in a dropped packet in the packet-level CRC scheme. This is reflected in the very poor frame delivery rates of packet-level CRC. However, at moderate offered loads, we see that it is not likely that very much of the packet is involved in a collision, because the frame delivery rates for PPR and fragmented CRC remain roughly unchanged between Figures 8 and 9.

Figure 10 shows how frame delivery rate changes when we increase the offered load to 13.8 Kbits/s/node in the network. 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 [1,26,27]. PPR's frame delivery rate remains high despite the high offered load, suggesting that only relatively-small parts of frames are actually being corrupted. This means that PPR has a large potential to improve end-to-end throughput of the system, and that higher layers, such as PP-ARQ and the routing layer, have a large potential for performance gains resulting from using PPR at the link layer.

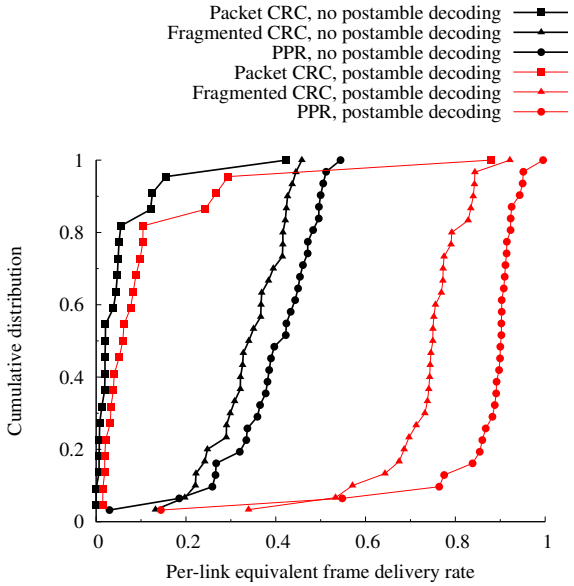

Figure 9: Equivalent frame delivery rate with carrier sense disabled, at an offered load of 3.5 Kbits/s/node.

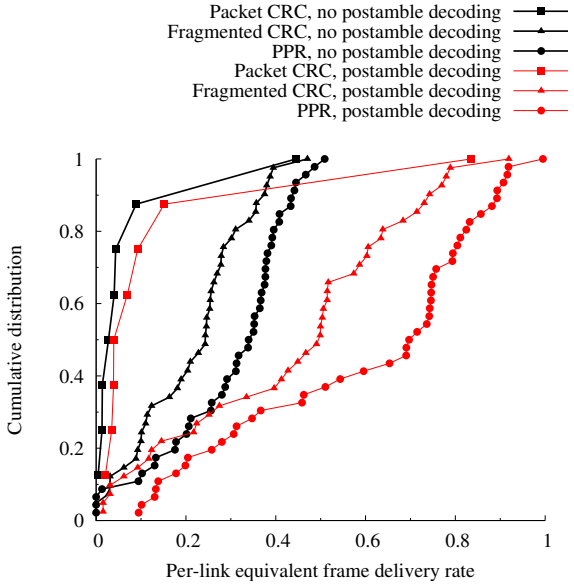

Figure 10: Equivalent frame delivery rate at an offered load of 13.8 Kbits/s/node. Carrier sense is disabled in this experiment.

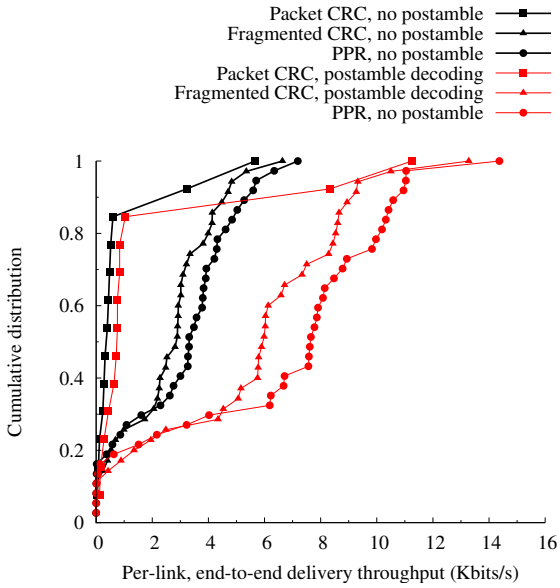

Figure 11: End-to-end per-link throughput at an offered load of 6.9 Kbits/s/node, close to channel saturation. Carrier sense is disabled in this experiment.

7.2.3 End-to-end throughput

We now examine end-to-end throughput from all of the senders to each of the four receivers in our network. These results take into account the performance of all the communication subsystems described above such as modulation, coding, use of spread-spectrum, and synchronization.

Figure 11 compares the per-link distribution of throughputs at medium offered load for each of the three schemes. We see that

The scatter plot in Figure 12 compares end-to-end throughput for fragmented CRC on the x-axis with either PPR (triangle points) or packet-level CRC (circles); we show results for all three offered loads. The first comparison we can draw from this graph is the per-link throughput of PPR compared with fragmented CRC (the triangle points in the graph). From these points we see that PPR improves performance over fragmented CRC by a roughly constant factor. This factor is related to the fragment size, and may be attributable to the fact that fragmented CRC needs to discard the entire fragment when a part of it is corrupted by another transmission.

The circle points in Figure 12 compare fragmented CRC with packet-level CRC. We see that fragmented CRC far out-performs packet-level 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 PPR. This is probably again because collisions do not occur over the entire packet, but rather often over a small piece of it.

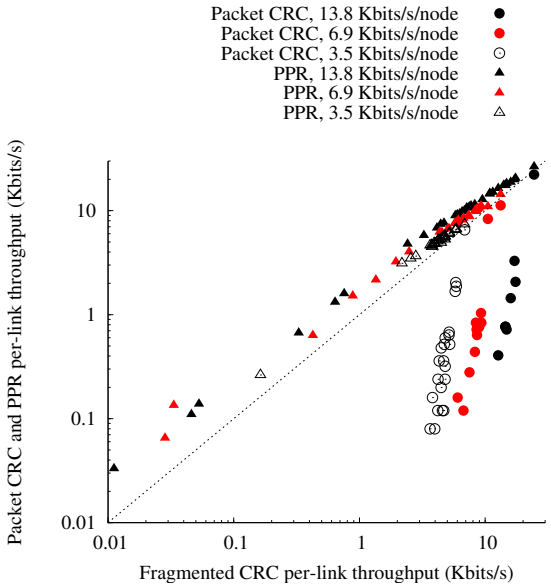

Figure 12: Comparison of end-to-end per-link throughput at various offered loads, with fragmented CRC as the baseline, on the x-axis. Carrier sense is disabled in this experiment.

7.3 SoftPHY hints during a partial packet reception

We now take a detailed look at a partial packet reception, showing the receiver's view of each codeword. Figure 13 shows a receiver's view of two packets sent from each of two senders. Each 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.[5] The upper plot in Figure 13 shows the first packet arriving at the receiver at time[6] 0, and the second packet packet arriving at time 10 (lower plot). When the second packet arrives, symbol timing recovery succeeds and the receiver decodes approximately 40 codewords correctly (including the preamble) before either losing symbol timing synchronization or a too-low SINR results in codeword errors. We see that Hamming distance remains at 0 for the duration of the correct codeword decisions, and rises at time 47 when a burst of codeword 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 (upper plot), the receiver successfully decodes a run of codewords extending to the end of the first packet. Having missed the first packet's preamble, it relies on its postamble in order to frame-synchronize and pass up the partial packet reception and associated SoftPHY hints.

[5]For clarity, we show the result of every fourth codeword-correctness test.
[6]Measured in units of codeword-time, 16 μs in our radios.

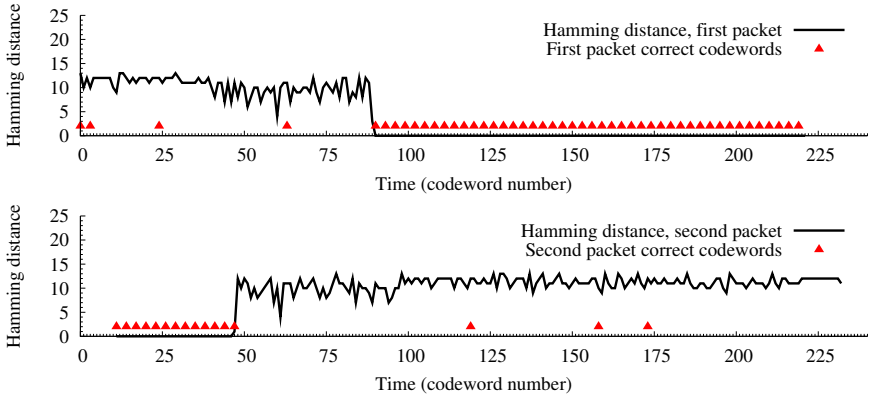

Figure 13: Partial packet reception during two concurrent transmissions: codeword correctness (triangle indicators) and each codeword's associated Hamming distance (curves). Despite uncertainty in PHY codeword timing recovery or SINR, Hamming distance indicates the correct parts of these packets to higher layers.

7.4 SoftPHY hints in a busy network

In Section 3 we introduced the SoftPHY hints that we use in our experimental evaluation; in Section 7.3 we saw that SoftPHY hints were a good predictor of correct decoding during an example packet reception. We now examine the statistics of the Hamming distance hint in more detail.

7.4.1 Miss rate

Recall from Section 3 that we label a codeword "correct" when its Hamming distance is less than or equal to η. Therefore the CDF curves of incorrect codewords in Figure 3 are also the fraction of incorrect codewords that we falsely label correct, and for which the CRC check on the resulting packet or run fails. We call this fraction the *miss rate* at threshold η, the rate at which we "miss" labeling a codeword incorrect at Hamming distance thresold η. We see from the figure that the miss rate is one in ten codewords at $\eta = 6$, 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 14 verifies this intuition, showing the complementary CDF of contiguous miss lengths at various thresholds η. We see that the majority of misses (30%) are of length 1, and that the distribution of miss length decreases faster than an exponential distribution, since the miss-length curve lies below a line on a log-log scale.

7.4.2 False alarm rate

We now examine the distribution of correct codeword Hamming distances more closely. In Figure 15, we plot the complementary cumulative distribution of correct

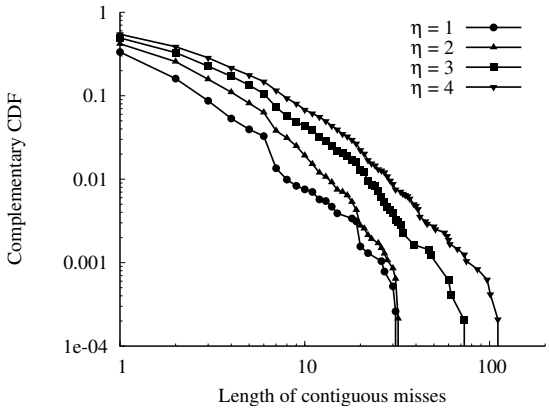

Figure 14: The distribution of lengths of contiguous misses in every received packet for various thresholds η.

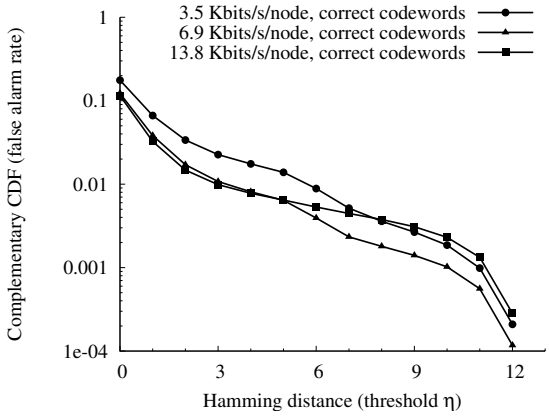

Figure 15: The complementary cumulative distribution of Hamming distances for every correct codeword in every received packet; also the "false alarm" rate for labeling correct codewords incorrect, causing them to be retransmitted.

codewords' Hamming distances: the fraction of correct codewords with Hamming distance greater than the ordinate of the graph. Since we label a codeword "incorrect" when its distance exceeds η, this complementary CDF is also the *false alarm* rate at threshold η: the fraction of correct codewords that we falsely label incorrect (and which PP-ARQ retransmits) at threshold η. Noting that the overhead of a false alarm is low — just one unnecessarily transmitted codeword, we see that the false alarm rate is very low; varying slightly with offered load, on the order of 5 in 1000 codewords at $\eta = 6$.

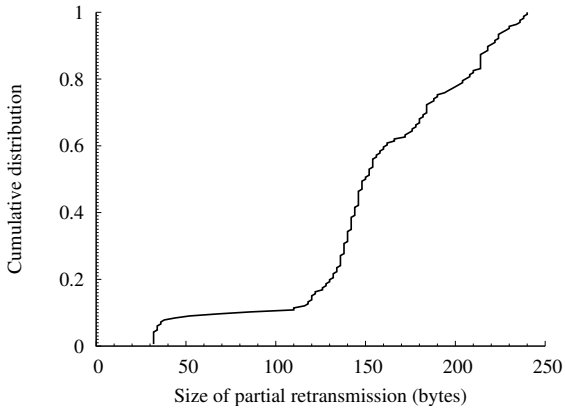

Figure 16: Packet sizes of partial retransmissions between a pair of nodes transferring data with PP-ARQ. Here each packet size is 250 bytes.

7.5 PP-ARQ

We now present some preliminary results from our implementation of PP-ARQ. In this single-link experiment, one GNU Radio transmitter sent 250 byte data packets back-to-back to another GNU Radio receiver using the same DSSS/MSK modulation as in the previous results.

Figure 16 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.

8.1 Work Related to SoftPHY

The Viterbi algorithm for convolutional decoding has been extended with a "soft" output that indicates the reliability of the output decision: the result is the soft-output Viterbi algorithm (SOVA) [11], a component of the Turbo decoder [4]. 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.

8.2 Work Related to Postamble Decoding

Since the body of a DSSS-modulated packet is composed of a known spreading sequence, Jeong and Lehnart propose using all of the packet for acquisition [13]. Our work builds on this idea by using similar postamble-based techniques to not only improve the probability of acquisition, but also recover from partial collisions more efficiently.

Whitehouse et al. [29] and Priyantha [23] propose avoiding "undesirable 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.

8.3 Wireless Error Control

Ahn et al. [2] 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 the combining of the two basic schemes of forward error control (FEC) and automatic repeat request (ARQ). Type I hybrid ARQ schemes [16] retransmit the same coded data in response to receiver NACKs. Chase combining [7] improves on this strategy by storing corrupted packets and feeding them all to the decoder. Type II hybrid ARQ schemes [16] forego aggressive FEC while the channel is quiet, and send parity bits on retransmissions, a technique called incremental redundancy [18]. Metzner [19] and later Lin and Yu [17] 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 [10, Chp. 8] 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 rate a priori in order to provision the amount of coding required to acheieve high throughput.

8.4 High-Throughput Wireless Protocols

Rate selection algorithms have been extensively studied in 802.11 wireless networks [5, 12, 14, 24]. 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.

SoftPHY and postamble decoding together also has the potential to improve the performance of mesh network protocols such as opportunistic routing [6] and network coding. In both cases, nodes need only forward or combine (using XOR, say) symbols (groups of bits) that are likely to be correct, and avoid wasting network capacity on incorrect data. 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.

Similarly, PPR has the potential to improve the performance of multi-radio diversity (MRD) schemes [20] in which multiple access points listen to a transmission and combine the data to recover errors before forwarding the result, saving on retransmissions. Avudainayagan [3, 30] *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 [20], while also achieving the performance gains of using PHY information.

9 Conclusion

This paper described the design, implementation, and experimental evaluation of PPR, a system for partial packet recovery. The motivation for PPR is the observation that wireless bit errors usually don't corrupt all the bits in a packet. PPR incorporates three novel techniques that work in concert: first, SoftPHY, which enhances the physical layer to compute and pass confidence information about each group of demodulated bits, and second, the postamble decoding scheme to recover bits even when a packet's preamble has been corrupted (postamble decoding). The confidence information can help higher layers perform better, as shown in our third technique, PP-ARQ, which shows how a receiver can use this information together with a dynamic programming algorithm to request the sender to re-send ranges of bits, rather than an entire packet.

We have implemented all three components on the GNU Radio platform for 802.15.4, the Zigbee standard, and evaluated the components and system in a 25-node indoor testbed. Our results show a 2× improvement in throughput over the status quo under moderate load, and 7× improvement at high load when many links have marginal quality. Furthermore, our proposed SoftPHY hint, Hamming distance, is a useful measure of bit correctness to PP-ARQ. Finally, PP-ARQ offers considerable savings in retransmission packet size.

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 use significant redundancy to tolerate worst-case channel conditions. If noise or interference during the reception of some codewords is higher than expected, existing PHY implementations will generate incorrect bits, which causing packet-level CRCs to fail. When that happens, entire packets have to be retransmitted. Since SINR depends on other concurrent transmissions and external sources, fluctuations in SINR are often

large. The penalty for incorrect decoding is also large. For these two reasons, PHY layers tend to be conservative with coding and modulation and MAC layers tend to be conservative with rate adaptation. The prevailing mind-set is that the consequences of bit errors are dire, and must be reduced (though eliminating them is impossible). As a result, current systems operate at 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 would be better for a PHY to use parameters that lead to a BER that is one or even two orders-of-magnitude higher than done currently, because higher layers need no longer have to cope with high *packet error rates*—they 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 3, 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 [6].

References

[1] AGUAYO, D., BICKET, J., BISWAS, S., JUDD, G., AND MORRIS, R. Link-level measurements from an 802.11b mesh network. In *Proc. ACM SIGCOMM* (Portland, OR, Aug. 2004).

[2] AHN, J.-S., HONG, S.-W., AND HEIDEMANN, J. An Adaptive FEC Code Control Algorithm for Mobile Wireless Sensor Networks. *Journal of Communications and Networks 7*, 4 (2005), 489–499.

[3] AVUDAINAYAGAM, A., SHEA, J. M., WONG, T. F., AND XIN, L. Reliability exchange schemes for iterative packet combining in distributed arrays. In *Proc. of the IEEE WCNC* (Mar. 2003), vol. 2, pp. 832–837.

[4] BERROU, C., GLAVIEUX, A., AND THITIMAJSHIMA, P. Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes. In *Proc. of the IEEE ICC Conf.* (May 1993), pp. 54–83.

[5] BICKET, J. Bit-rate selection in wireless networks. Master's thesis, Massachusetts Institute of Technology, Feb. 2005.

[6] BISWAS, S., AND MORRIS, R. ExOR: Opportunistic multi-hop routing for wireless networks. In *ACM SIGCOMM 2005* (Aug. 2005).

[7] CHASE, D. Code combining: A maximum-likelihood decoding approach for combining an arbitrary number of noisy packets. *IEEE Trans. on Comm. 33*, 5 (May 1985), 385–393.

[8] EFSTATHIOU, D., FINES, P., AND H, A. A. Preamble-less non-decision-aided (NDA) techniques for 16-QAM carrier phase recovery and gain error correction, for burst transmissions. In *Proc. of the IEEE GLOBECOMM Conf.* (Nov. 1993), pp. 1090–1094.

[9] The GNU Radio Project. http://www.gnu.org/software/gnuradio/.

[10] GOLDSMITH, A. *Wireless Communications*. Cambridge Univ. Press, New York, NY, 2005.

[11] HAGENAUER, J., AND HOEHER, P. A Viterbi Algorithm with Soft-Decision Outputs and its Applications. In *Proc. of the IEEE GLOBECOM Conf.* (Dallas, TX, Nov. 1989).

[12] HOLLAND, G., VAIDYA, N., AND BAHL, P. A rate-adaptive MAC protocol for multi-hop wireless networks. In *ACM Mobicom 2001* (Sept. 2001).

[13] JEONG, Y., AND LEHNERT, J. Acquisition of Packets with a Short Preamble for Direct-Sequence Spread-Spectrum Multiple-Access Packet Communications. In *Proc. of IEEE MILCOM Conf.* (2003), pp. 1060–1064.

[14] LACAGE, M., MANSHAEI, M. H., AND TURLETTI, T. IEEE 802.11 rate adaptation: a practical approach. In *Proc. of the ACM MSWiM Workshop* (Oct. 2004), pp. 126–134.

[15] LEE, E., AND MESSERSCHMITT, D. *Digital Communication*. Kluwer Academic Publishers, Boston, MA, 1988.

[16] LIN, S., AND COSTELLO, D. J. *Error Control Coding*, 2nd. ed. Prentice Hall, 2004.

[17] LIN, S., AND YU, P. S. A Hybrid ARQ Scheme with Parity Retransmission for Error Control of Satellite Channels. *IEEE Trans. on Comm. 30*, 7 (July 1982), 1701–1719.

[18] MANDELBAUM, D. An Adaptive-Feedback Coding Scheme Using Incremental Redundancy (Corresp.). *IEEE Trans. on Information Theory 20*, 3 (May 1974), 388–389.

[19] METZNER, J. Improvements in Block-Retransmission Schemes. *IEEE Trans. on Comm. 27*, 2 (Feb. 1979), 524–532.

[20] MIU, A., BALAKRISHNAN, H., AND KOKSAL, C. E. Improving loss resilience with multi-radio diversity in wireless networks. In *ACM Mobicom* (Aug. 2005).

[21] MUELLER, K., AND MÜLLER, M. Timing Recovery in Digital Synchronous Data Receivers. *IEEE Trans. on Comm. 24*, 5 (May 1976).

[22] PASUPATHY, S. Minimum Shift Keying: A Spectrally-Efficient Modulation. *IEEE Communications Magazine 7*, 4 (July 1979), 14–22.

[23] PRIYANTHA, N. B. *The Cricket Indoor Location System*. PhD thesis, MIT, May 2005.

[24] SADEGHI, B., KANODIA, V., SABHARWAL, A., AND KNIGHTLY, E. Opportunistic media access for multirate ad hoc networks. In *Proceedings of ACM Mobicom 2002* (September 2002).

[25] SCHMID, T. Personal communication. Also see code repository at http://acert.ir.bbn.com/projects/gr-ucla/.

[26] SON, D., KRISHNAMACHARI, B., AND HEIDEMANN, J. Experimental Analysis of Concurrent Packet Transmissions in Low-Power Wireless Networks. In *SenSys* (Boulder, CO, Nov. 2006).

[27] SRINIVASAN, K., DUTTA, P., TAVAKOLI, A., AND LEVIS, P. Understanding the Causes of Packet Delivery Success and Failure in Dense Wireless Sensor Networks. Tech. Rep. SING-06-00, Stanford Univ., 2006.

[28] TI/Chipcon Products CC2420 Data Sheet. http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf.

[29] WHITEHOUSE, K., WOO, A., JIANG, F., POLASTRE, J., AND CULLER, D. Exploiting the Capture Effect for Collision Detection and Recovery. In *IEEE EmNets Workshop* (Sydney, Australia, May 2005).

[30] WONG, T. F., XIN, L., AND SHEA, J. M. Iterative decoding in a two-node distributed array. In *Proc. of the IEEE MILCOM Conf.* (Oct. 2002), vol. 2, pp. 1320–1324.