CodeRepair: PHY-layer Partial Packet Recovery Without the Pain

Jun Huang[†], Guoliang Xing[†], Jianwei Niu[§] and Shan Lin[‡]
[†]Department of Comptuer Science and Engineering, Michigan State University, MI 48824, USA
[‡]Department of Electrical and Computer Engineering, Stony Brook University, NY 11790, USA
[§]State Key Laboratory of Virtual Reality Technology and Systems, Beihang University, Beijing 100191, China

***Abstract*—Prior studies show that repairing partially corrupted packets, instead of retransmitting them in their entirety, holds potential in improving the performance of 802.11 networks. However, the efficiency of existing packet recovery approaches is severely limited by various overhead associated to redundant transmission and repeated channel contention. In this paper, we propose *CodeRepair*, a practical coding-based protocol that recovers partially corrupted 802.11 packets without these pains. The design of CodeRepair is based on two novel ideas. First, CodeRepair pushes the limit of 802.11 PHY to piggyback parities in the *padded bits* of OFDM, obviating the need of transmitting extra information for error correction. Second, CodeRepair corrects errors at the PHY layer, which is significantly more efficient than traditional link-layer approaches. This is due to the fact that a single coded bit usually affects the decoding of a group of data bits in 802.11 convolutional code. As a result, CodeRepair can salvage a partially corrupted packet by correcting a small number of erroneous coded bits using the padded parities. To reduce computational cost of error recovery, CodeRepair employs single parity code for correcting coded bit errors. We propose several techniques to augment the error correcting capability of single parity code without compromising its computation efficiency. Our evaluation shows that CodeRepair recovers an average of 34% partially corrupted packets, and improves the end-to-end link goodput by 59% on lossy 802.11 links.**

I. Introduction

Despite the fast increase of PHY data rate, wireless networks have not seen a commensurate performance improvement [14], especially in challenging environments where wireless links suffer poor reliability caused by time-varying interference and channel fading conditions. For instance, a recent measurement conducted on VanLAN – a testbed of connected buses deployed in the Bay Area – show that wireless links between moving buses and base stations are extremely lossy, and the transport layer suffers a loss ratio up to 10%, even in the presence of link layer error recovery mechanisms [15]. Measurements on off-the-shelf Wi-Fi devices show that about 40% of received packets are partially corrupted when the receiver is moving at walking speed [12].

Prior studies showed that repairing partially corrupted packets, instead of retransmitting them in their entirety, holds substantial potential in improving wireless network reliability [24] [12] [11] [10] [23]. However, the existing packet recovery solutions often imped system throughput due to their high overhead. For example, cooperative protocols [5] [13] [16] [22] need to coordinate multiple receivers over the wired backbone for correcting bit errors. The protocols based on forward error correction (FEC) codes [12] [23] and selective retransmission [24] [11] [10] entail extra bandwidth consumption for transmitting error correction codes or retransmitting parts of the packet.

In this paper, we propose *CodeRepair*, a practical system that recovers partially corrupted packets in 802.11 networks without the above pains. CodeRepair achieves this goal by integrating two key novel designs. First, *CodeRepair pushes the limit of 802.11 PHY to piggyback a small number of parities without incurring extra transmission overhead*. This is realized by exploiting the *padding bits* in OFDM, which is adopted by current 802.11a/g/n standards. Each OFDM symbol carries tens to hundreds of bits, depending on the modulation rate. When the packet size is not an integral number of OFDM symbols, the sender has to pad rubbish bits to fill the last OFDM symbol. CodeRepair salvages these padded bits to carry parity bits, obviating the need of sending redundant information for error correction. Second, to augment the error correction capability of padded parities, *CodeRepair performs error recovery at the PHY layer, which represents a paradigm shift from exiting FEC-based link layer approaches [12] [23]*. Our key insight is that a single coded bit usually affects the decoding of a group of data bits, due to the use of convolutional code in 802.11. Therefore, correcting a single coded bit may improve the success rate of decoding multiple data bits. As a result, CodeRepair can salvage a data packet with many errors by correcting a small number of erroneous coded bits using the padded parities.

The key challenge of realizing the above ideas in 802.11 PHY is to minimize the complexity of error correction and packet recovery. This is particularly crucial due to the delay sensitive nature of 802.11 PHY. For example, the receiver must complete packet recovery within the short inter-frame space (SIFS) defined by 802.11, which is only 10μs in 802.11g. To meet the stringent timing constraints of 802.11 PHY, the computational cost of packet recovery needs to be minimized. To this end, CodeRepair employs the single parity code (SPC), a simple error detection code, for correcting coded bit errors. To improve the error correcting capability of SPC, we propose a novel design which integrates several simple yet effective techniques, including code reversing, selective re-decoding, and parity sampling rate optimization, to augment SPC into a highly efficient outer FEC atop the conventional channel codes without compromising its computation simplicity.

978-1-4799-8381-0/15/$31.00 ©2015 IEEE

We implemented CodeRepair in USRP/GNURadio platform, and extensively evaluated its error correcting efficiency, as well as the impact on end-to-end link performance. We show that CodeRepair recovers an average of 34% erroneous packets, and improves the goodput by 59% on lossy links. Although we primarily focus on 802.11 networks in this paper, the design of CodeRepair can be applied to other OFDM-based radio technologies, such as LTE and ultra-wideband (UWB). Moreover, the augmented SPC can be used as an efficient outer code to improve the performance of channel coding in other delay sensitive environments.

II. Related Work

Corrupted packet recovery for 802.11. Traditionally, 802.11 recovers corrupted packets using link layer protocols. Selective retransmission based protocols [11] [10] reduce packet recovery overhead by only retransmitting the bits that are likely to be wrong. Similarly, Zhang et al. [24] propose μACK, which sends ACKs for every few symbols on a separate feedback channel to enable in-frame recovery. Other protocols, such as ZipTx [12], use link layer FEC to correct bit errors. Unite [23] combines selective retransmission and link layer FECs to minimize error recovery overhead. Leveraging the broadcast nature of wireless channel, cooperative protocols [5] [13] [16] [22] coordinate multiple receivers that hear the same transmission to correct packet errors. However, the efficiency of these approaches is severely limited by the high error reovery overhead, which includes the costs of coordinating multiple receivers, extra bandwidth consumption for sending redundancy information, and the channel access overhead during retransmissions. In comparison, CodeRepair avoids these overhead and explores the efficiency of PHY packet recovery. Moreover, CodeRepair can be integrated with link layer protocols to improve their performance.

Hybrid ARQ and rateless code. 4G wireless standard uses hybrid ARQ for correcting corrupted packets. In hybrid ARQ, data is first encoded using a low-rate mother code, and then in each round of (re)tranmission the coded block is punctured (i.e., only a fraction of the coded bits are chosen) and transmitted, until the packet is successfully decoded or all coded bits are sent. However, if bit errors cannot be completely corrected after sending all codes, a new round of retransmission will be triggered, which causes bandwidth waste just like link layer retransmissions. Moreover, hybrid ARQ only works when the underlying channel code is *rate-compatible* [7]. Unfortunately, 802.11 convolutional code is not rate-compatible [20]. Thus realizing hybrid ARQ in 802.11 requires non-trivial modifications in the default encoder. To optimize link performance under time-varying channel conditions, recent work [6] [17] propose to use rateless code to automatically achieve the optimal transmission rate. However, realizing rateless codes requires significant modifications to the current 802.11 PHY. Moreover, decoding rateless codes incurs multiple rounds of extensive computations [17].

Both hybrid ARQ and rateless codes require a *stateful* PHY, where the decoder has to buffer the coded bits for all ongoing packets until they are successfully decoded. While stateful PHY is employed in celluar networks where a centralized controller schedule all transmissions, it is prohibitively expensive in the unlicensed band, where senders compete with each other for channel access. Since packets from other senders may arrive before the ongoing packet is successfully decoded, it is extremely difficult to maintain packet status at the PHY layer. As a result, 802.11 adopts a stateless PHY where decodings of packets are independent of each other.

Turbo code. CodeRepair uses padded parities as an outer FEC atop of channel code. This structure is similar to that of Turbo code. However, a key feature of CodeRepair is its extremely low overhead. Turbo code parallel y concatenates two FECs, and relies on iterative decoding to reduce error rate. This requires not only significant modifications to the structure of 802.11 encoder, but also extensive computations at the decoder. In comparison, CodeRepair piggybacks the parities of outer code in the padded bits of 802.11 packets, without modifying the default encoder or transmitting extra bits. Moreover, existing FECs typically requires substantial computations for decoding, which will degrade the efficiency of 802.11 PHY. CodeRepair augments the single parity code for error recovery, of which the incurred computational overhead is only 2% to 7% to that of channel decoders. These features make it feasible to incorporate CodeRepair into existing 802.11 standards.

III. Background on 802.11 PHY

In this section, we introduce the background of 802.11 PHY that is necessary for understanding our approach.

A. Convolutional Code

A rate-m/n convolutional code transforms each m-bit message into n-bit codeword. Coded bits are computed based on the current and last k bits using pre-defined generator polynomials, where k is referred to as *constraint length*. The state of the encoder is defined by the values of last k bits. Tab. I shows an example of a rate-1/2 convolutional code with $k = 2$. The two generator polynomials are $c_1 = b_0 \oplus b_1 \oplus b_2$, and $c_2 = b_0 \oplus b_2$, where b_0 is the current bit, $b_1 b_2$ represents the encoder state. The transition table describes how an input bit triggers a state transition inside the encoder, and the output table gives the computation results of generator polynomials. In 802.11, a rate-1/2 code with $k = 6$ is employed as mother code. Higher coding rate can be achieved by puncturing certain coded bits before transmission. For each bit b_i, a codeword of two bits is computed using the following generator polynomials,

$$\begin{aligned} c_{2i} &= b_i \oplus b_{i-2} \oplus b_{i-3} \oplus b_{i-5} \oplus b_{i-6}, \\ c_{2i+1} &= b_i \oplus b_{i-1} \oplus b_{i-2} \oplus b_{i-3} \oplus b_{i-6}, \end{aligned} \tag{1}$$

where b_{i-k} is the bit of round $i-k$, and $b_{i-k} = 0$ if $i < k$.

Fig. 1 shows the process of convolutional encoding using the code given in Tab. I. PATH-1 and PATH-2 show the traces of state transitions when encoding the messages of 00000 and 11010. The goal of decoding is to find the most likely path on the trellis that yields the coded bit sequence

TABLE I
AN EXAMPLE OF A RATE-1/2 CONVOLUTIONAL CODE.

state (b_1b_2)	Transition		Output (c_1c_2)	
	$b_0 = 0$	$b_0 = 1$	$b_0 = 0$	$b_0 = 1$
00	00	10	00	11
01	00	10	11	00
10	01	11	10	01
11	01	11	01	10

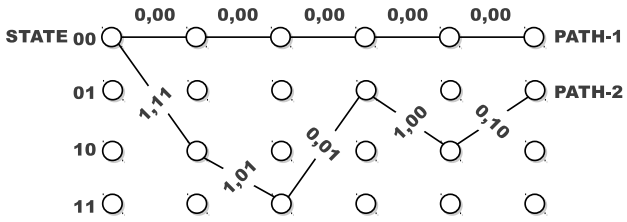


Fig. 1. Illustration of convolutional encoding using the code defined in Tab. I. Each branch is labeled by the input bit that triggers the transition, along with the output codeword.

of minimum *cost* to the received sequence, where the cost is commonly computed using Hamming distance. For example, given a received coded bit sequence of 0100000100, the costs of PATH-1 and PATH-2 are 2 and 5, respectively. In this case, PATH-1 is more likely the correct path.

B. Bit Padding

In OFDM-based wireless systems, when packet size is not an integral multiple of the number of bits carried by an OFDM symbol, rubbish bits will be padded at packet tail to fill the last OFDM symbol. In 802.11, each OFDM symbol carry a total of 24 to 320 data bits, depending on the modulation and channel coding schemes. Let n be the number of data bits carried by each OFDM symbol. A k-bit packet consists of $\lfloor k/n \rfloor$ OFDM symbols. The number of padded bits can be calculated as $n \times \lfloor \frac{k}{n} \rfloor - k$.

IV. A MOTIVATION EXAMPLE

Our approach is driven by the hypothesis that *recovering corrupted 802.11 packets at the PHY layer requires significantly less redundancy than that required by link layer protocols*. The intuition behind is that, by correcting an erroneous coded bit, we can affect the decoding of a sequence of surrounding data bits, leading to higher error correction efficiency. In the following, we demonstrate this intuition using an example based on the code defined in Tab. I.

Tab. II gives an example where a correcting one coded bit recovers 4 data bits. The first three columns show transmitted bits, transmitted and received coded bits, respectively. For PATH-I and PATH-2, the table shows the 2-bit codeword expected by the path, along with the decoding output and the costs accumulated on the path. Assuming a 4-bit message '0000' is transmitted using the code defined in Tab. I, the transmitted and received coded bits are '00000000' and '10011000', respectively. Tab. II shows the case where the decoder compares the costs of the correct path (PATH-1) and

TABLE II
AN EXAMPLE WHERE CORRECTING ONE CODED BIT RECOVERS FOUR DATA BITS.

Tx bit	Tx code	Rx code	PATH-1			PATH-2		
			Code	Bit	Cost	Code	Bit	Cost
0	00	1̲0	00	0	1	11	1	1
0	00	0̲1	00	0	2	01	1	1
0	00	1̲0	00	0	3	10	1	1
0	00	0̲0	00	0	3	10	1	2

a faulty path of 4 bit errors (PATH-2). As the accumulated costs of PATH-1 and PATH-2 are 3 and 2, the decoder will eliminate PATH-1, causing 4 bit errors in decoding output. In this example, correcting one of coded bit errors will reduce the cost of PATH-1 to 2, and increase the cost of PATH-2 to 3, reviving PATH-1 and correcting 4 bit errors.

V. CHALLENGES AND SYSTEM OVERVIEW

Realizing the idea of PHY packet recovery in 802.11 raises two practical challenges.

Maintaining communication efficiency. Recent studies [14] [24] show that the channel contention cost of 802.11 can easily offside the spectrum efficiency of high-rate PHY. To minimize communication overhead, an efficient packet recovery solution should avoid retransmissions whenever it is possible.

Minimizing computational overhead. 802.11 PHY is delay-sensitive. If a sender does not receive an ACK within the window of SIFS (10 μs in 802.11g), it will consider the packet lost and trigger a retransmission. PHY packet recovery requires an extra round of decoding to correct coded bits, leading to higher delay. This delay depends on the complexity of the FEC used for error correction, and must be minimized to meet the stringent 802.11 timing constraints.

To address the above challenges, we propose *CodeRepair*, a practical PHY packet recovery protocol for 802.11. CodeRepair pushes the limit of 802.11 PHY to piggyback a small number of parities on the padded bits of OFDM, therefore avoiding extra communication overhead. During error correction, CodeRepair receiver first uses piggybacked parities to correct a subset of received coded bits, and then re-decodes to recover a the packet without triggering retransmissions.

CodeRepair is designed to be highly computational efficient. To this end, we employ single parity code (SPC) – a simple error detection code – for coded bits recovery. To augment SPC for error correction, CodeRepair integrates three techniques including *code reversing*, *selective re-decoding*, and *parity optimization*, to turn SPC into an efficient outer code atop 802.11 convolutional code without compromising SPC's computation simplicity. CodeRepair can be integrated with other FECs to achieve better error correcting performance, although this comes at the cost of increased complexity.

We now give an overview on the working flow of CodeRepair. On the sender side, CodeRepair generates a small number of parities, where each parity is the modulo-2 sum of some randomly selected coded bits. The parities are then piggybacked by the padded bits of 802.11. Fig. 2 shows the architecture of

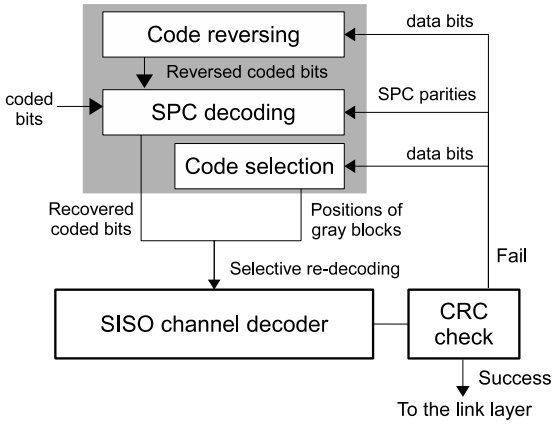


Fig. 2. The architecture of an 802.11 receiver augmented with CodeRepair components (gray area).

an 802.11 receiver augmented with CodeRepair components. For each packet that fails CRC check, CodeRepair receiver extracts padded parities after packet decoding, and then performs SPC decoding to re-estimate a subset of coded bits. To augment SPC for error correction, CodeRepair employs code reversing, a technique that leverages the decoding confidence of data bits to reverse coded bit values. The number of bits sampled by each parity is carefully tuned to improve error correction performance. To correct erroneous data bits, CodeRepair selectively re-decodes a group of low confidence bit blocks using the re-estimated coded bit sequence. In the next section, we will introduce the design of CodeRepair in detail.

VI. CodeRepair Design

In this section, we introduce the coding of SPC, describe the design of CodeRepair sender and receiver, and then discuss how to improve the utilization of padded parities.

A. Single Parity Code

CodeRepair augments single parity code (SPC) – a simple error detection code – to an efficient outer FEC atop 802.11 convolutional code. The key motivation driven this design is to minimize computational overhead. In the following, we first introduce the coding and decoding of SPC, and then discuss its deficiency in error correction.

Coding and decoding. CodeRepair employs SPC to encode and estimate coded bit values of 802.11 packet. Given n randomly selected coded bits $x_1, ..., x_n$, the SPC encoder computes a single parity p, such that $p \oplus x_1 ... \oplus x_n = 0$. Let $y_1, ..., y_n$ be the received coded bits, and q be the received parity. We use $\phi(y_i|s)$ to denote the LLR of y_i obtained through soft-demodulating the received signal s; $\phi(q|s)$ can be derived by soft-decision channel decoding. To decode SPC, we resort to the decoding techniques developed for LDPC. Specifically, for two independent binary variables a and b, $\phi(a \oplus b)$ can be derived using Jacobian function [3],

$$\phi(a \oplus b) = \text{sgn}\big(\phi(a)\big)\text{sgn}\big(\phi(b)\big)\big(\min(|\phi(a)|, |\phi(b)|) - c\big), \quad (2)$$

where $\text{sgn}(a)$ is the sign of a, and c can be approximated using a constant while incurring only a negligible loss in accuracy [4]. Following the Bayes' rule, we can use the received parity q to update the LLR of y_i as follows,

$$\phi(y_i|s,q) = \phi(y_i|s) + \phi(y_i|q) = \phi(y_i|s) + \phi(y_i \oplus u|s), \quad (3)$$

where $u = q \oplus y_1 \oplus ... \oplus y_n$.

Deficiency in error correction. Despite its computation efficiency, SPC is rarely used as an error correcting code. As an error detection code, SPC only detects odd number of errors, and cannot identify the positions of erroneous bits. When there is odd number of errors, $\text{sgn}(\phi(y_i|q)) = -\text{sgn}(\phi(y_i|s))$. In this case, the SPC decoder trades off the accuracy of correct coded bits to recover erroneous ones. When there is even number of errors, we have $\text{sgn}(\phi(y_i|q)) = \text{sgn}(\phi(y_i|s))$. In this case, the parity has no contribution to error correction. In Section VI-C, we will address this inherent deficiency of SPC, enabling it to recover coded bits while preserving its computation simplicity.

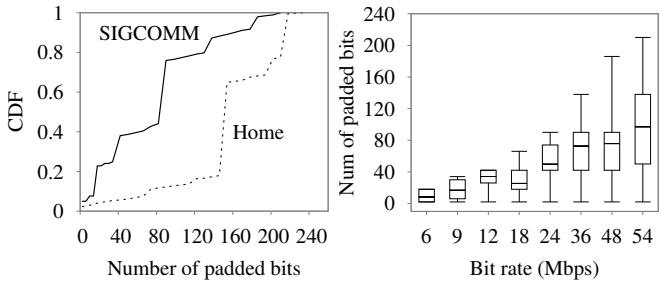
B. CodeRepair Sender

On the sender side, CodeRepair employs SPC to encode the coded bits of 802.11 packet into parities. Each parity randomly samples a group of coded bits, which are selected based on a pseudo-random number sequence generated offline. The sender and receiver share this sequence during SPC encoding and decoding. CodeRepair sender piggybacks parities on the padded bits of 802.11 (see Section III-B), allowing the receiver to correct corrupted packets without triggering retransmissions or consuming extra bandwidth.

Fig. 3(a) shows the number of padded bits measured in two data traces of real-life 802.11 networks. SIGCOMM [18] is a publicly available packet trace collected in WLANs of hundreds of users attending SIGCOMM 2008. The trace named Home is collected by ourselves in an apartment from an 802.11n WLAN. We observe plenty of padded bits in both traces. The average numbers of padded bits per packet are 83 and 157 SIGCOMM and Home, respectively. Fig. 3(b) further studies the impact of bit rate using the SIGCOMM trace. As shown in the figure, packets transmitted at high bit rates contain more padded bits. Denote n_{data} as the number of bits carried by one OFDM symbol. n_{data} increases with data rates. Assuming the packet size is randomly distributed, the average number of padded bits is $n_{data}/2$. In 802.11n and 802.11ac, padded bits can be even more substantial.

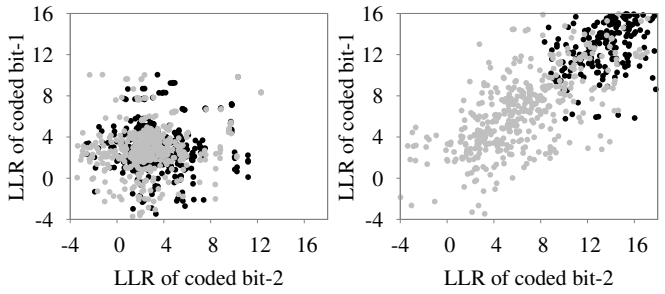
C. CodeRepair Receiver

At the receiver, CodeRepair utilizes the padded parities to recover coded bits, and then re-decodes to correct corrupted packets. To address the deficiency of SPC, CodeRepair optimizes the sampling rate of parities to maximize their utilities (see Section VI-D). Meanwhile, CodeRepair receiver uses two techniques called *code reversing* and *selective re-decoding* to augment SPC decoder. In the following, we explain how these



(a) Number of padded bits per packet. (b) Effect of bit rates measured in SIGCOMM trace.

Fig. 3. Padded data bits in real-life network traffic.



(a) Before code reversing. (b) After code reversing.

Fig. 4. Effect of code reversing. Each point denotes a codeword consisting of two coded bits. A negative LLR indicates an erroneous coded bit. Gray points denote the codewords containing at least one gray code.

techniques are integrated together, and then describe code reversing and selective re-decoding in detail.

Basic idea of augmenting SPC decoding. At the very high-level, CodeRepair employs SPC to correct coded bit errors. Due to the deficiency discussed in Section VI-A, SPC decoding may decrease the estimation accuracy of correct coded bits. CodeRepair suppresses this side effect through selective re-decoding. Specifically, by evaluating the decoding confidence, CodeRepair receiver first identifies a group of *gray blocks* that are likely to contain decoding errors. We denote the coded bits located in gray blocks as *gray codes*. Then the receiver performs code reversing, which exploits decoded bits of high confidence to recover non-gray codes. By optimizing the sampling rate of parities (as discussed in Section VI-D, we can maximize the chance at which each parity samples one gray code. As a result, the estimation error of the sampled gray code will be reduced after SPC decoding. However, as shown in Eq. (2) and Eq. (3), the accuracy of non-gray codes will be traded off. This side-effect is suppressed using selective re-decoding, where only the gray blocks are re-decoded to correct decoding errors.

Code reversing. The goal of code reversing is to recover non-gray codes through *soft* re-encoding data bits using their decoding confidence. The intuition behind is straightforward. Considering the case when a packet is decoded correctly, a simple method to correct all coded bit errors is to re-encode the packet. Specifically, to reverse a codeword $y_{2i}y_{2i+1}$, we take Eq. (1) into Eq. (2), and then follow the Bayes' rule to sum the results with $\phi(y_{2i}|s)$ and $\phi(y_{2i+1}|s)$ to update the estimation.

Fig. 4 illustrates the effect of code reversing. The transmitted packet consists of all zero bits. Based on Eq. (1), transmitted coded bits are all zeros too. Thus a negative LLR indicates a coded bit error. Each gray point in the figure denotes a codeword that contains at least one gray code. As shown Fig. 4, code reversing significantly reduces errors in non-gray codes. However, we note that simply re-decoding using the reversed codes will not correct most decoding errors because code reversing cannot recover errors in gray codes, which are the root causes of decoding errors.

Code coloring and selective re-decoding. In each corrupted packet, CodeRepair receiver identifies a group of gray blocks and then re-decodes them after coded bits recovery. Specifically, the receiver colors the i-th codeword as gray, if there exists a bit b_j at position j, such that $|j-i| \leq k$ and $|\phi(b_j)| \leq T$, where $k = 6$ is the constraint length of 802.11 convolutional code, and T is a pre-defined threshold on decoding confidence. This coloring scheme is motivated by the principle of 802.11 convolutional decoding, i.e., an erroneous codeword may affect the decoding of all the bits located within a distance of constraint length. Once the gray blocks are identified, the receiver calls the default channel decoder to re-decode using recovered gray codes for error correction. To choose T for CodeRepair, we conduct an empirical measurement on a testbed of 10 USRP links to find the minimum confidence that includes all bit errors for 95% packets.

D. Parity Sampling Rate Optimization

To assure that SPC decoding will improve the estimation accuracy of gray codes, CodeRepair tunes the sampling rate of parities to maximize the probability at which each parity samples one gray code. Therefore all reversed non-gray codes sampled by the same parity will contribute to recover the gray code during SPC decoding.

Let γ_g be the ratio of gray codes, and n be the number of coded bits sampled by each parity. Denote $\mathcal{P}(n, \gamma_g)$ as the probability where one gray code is sampled by a parity, we have,

$$\mathcal{P}(n, \gamma_g) = n \times \gamma_g \times (1-\gamma_g)^{n-1}, \tag{4}$$

Our goal is to find n, such that,

$$n_{opt} = \arg\max_n \{\mathcal{P}(n, \gamma_g)\} = -\frac{1}{1-\log \gamma_g}. \tag{5}$$

To optimize n, a rigorous approach would be to measure the probabilistic distribution of γ_g at runtime, and then maximize the expectation of Eq. (5) accordingly. Unfortunately, this may significantly complicate the design of PHY. We address this problem using an empirical approach to profile the distribution of γ_g offline, and then compute n_{opt} using Eq. 5. In our measurement, we observe that the distributions of gray code ratio varies for packet of different sizes. Corrupted packets of larger size have a higher gray code ratio. To address this issue, we profile different n_{opt} for different packet lengths.

VII. Performance Analysis

In this section, we present an analytical study to verify the efficiency of PHY packet recovery based on Shannon's random code [19]. We characterize the efficiency by metric γ, which is defined as the expected number of bit errors reduced by correcting one coded bit error.

We begin with a brief introduction on the background of random code. At the encoder, a message is first divided into n-bit blocks. Then each block is encoded into a m-bit codeword based on the random mapping defined in a codebook, which is shared by both sender and receiver. The above code is denoted as (n,m)-code, where n/m is the coding rate.

Assuming that a m-bit codeword is received with ε errors, the received codeword defines a set $\mathcal{A}^{\varepsilon}$, in which the Hamming distances from all codewords to the received one is less than ε. The size of $\mathcal{A}^{\varepsilon}$ is,

$$|\mathcal{A}^{\varepsilon}| = \binom{m}{0} + \binom{m}{1} + ... + \binom{m}{\varepsilon}$$

If at least one of the codewords in $\mathcal{A}^{\varepsilon}$ is selected to encode a block, the transmitted block will be eliminated during decoding, causing decoding errors[1]. Assuming that m is reasonably large so that $\varepsilon \ll 2^m$, the probability of correct decoding can be computed as,

$$\mathcal{P}(\varepsilon) = (1 - 2^{n-m})^{|\mathcal{A}^{\varepsilon}|}$$

Given that the block is decoded with errors, to correct the block, number of coded bits needed to correct is,

$$\sum_{0 \le k < \varepsilon} \frac{\Big(\varepsilon - k\Big)\Big(\mathcal{P}(k) - \mathcal{P}(k+1)\Big)}{1 - \mathcal{P}(\varepsilon)} = \frac{\varepsilon - \sum_{0 < k \le \varepsilon} \mathcal{P}(k)}{1 - \mathcal{P}(\varepsilon)}$$

Because of random coding, an erroneous n-bit block contains an average of $n/2$ bit errors. Thus γ can be derived as,

$$\gamma = \sum_{0 < \varepsilon < m} \rho(\varepsilon) \times \frac{n}{2} \times \frac{1 - \mathcal{P}(\varepsilon)}{\varepsilon - \sum_{0 \le k \le \varepsilon} \mathcal{P}(k)} \tag{6}$$

where $\rho(\cdot)$ is the probability of having ε coded bit errors in a m-bit codeword. For binary symmetric channel (BSC) and AWGN channel, coded bit errors occur independently of each other. Thus ε follows binomial distribution, i.e., $\varepsilon \sim \mathcal{B}(m, \beta)$, where β is the coded bit error rate.

Fig. 5 plots Eq. (6) for four random codes of different block lengths over binary symmetric channels. As shown in the figure, correcting one coded bits reduces a large number of bit errors. Moreover, we find that *PHY packet recovery is particularly efficient when the block length of code is large or the coded bit error rate is low.* This result suggests that real-life wireless networks will benefit from PHY packet recovery for two reasons. First, as a common practice, real-life wireless networks usually employ channel codes of large block length for improving the resistance to transmission errors. As an example, the block length of LDPC employed in 802.11n ranges from 648 to 1944. Second, coded bit error rate observed in real-life wireless networks is typically low due to the effect of bit rate adaptation, which optimizes the modulation to minimize transmission errors under time-varying channel conditions.

[1]We assume that the decoder will eliminate the true message if its distance is equal to another candidate. Thus the result derived in this section provides a lower bound of γ.

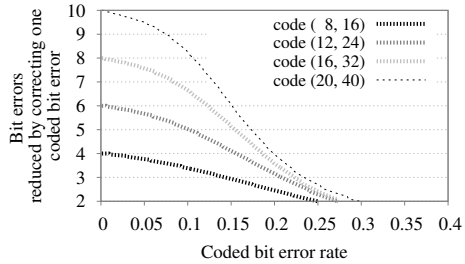


Fig. 5. Bit errors reduced by correcting one faulty coded bit for Shannon's random code.

VIII. Evaluation

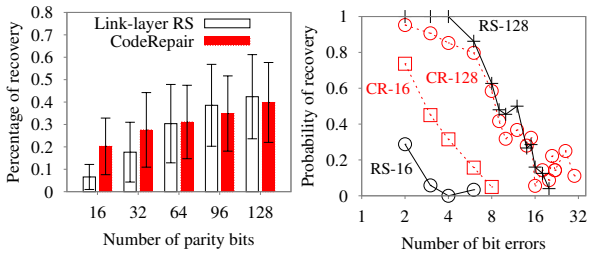
We prototype CodeRepair in an 802.11 like PHY atop the OFDM modules of GNURadio/USRP platform. The PHY implements 802.11 compliant operations, including scrambling, interleaving, convolutional coding and decoding. We employ BCJR [1], a widely adopted algorithm for maximum a posteriori decoding of 802.11 convolutional code. The decoder takes demodulated symbols as input, converts complex symbols on constellation to soft coded bits using a soft-output de-mapper [21], and then performs soft-in soft-out decoding. CodeRepair is integrated with BCJR following the structure shown in Fig. 2.

Our evaluation centers around three aspects. (1) How efficiently can CodeRepair recover corrupted packets? (2) What is the impact of CodeRepair on the link-layer of 802.11 with transmission rate adaptation? and (3) How much is the decoding overhead of CodeRepair?

A. Efficiency of Error Correction

Experiment settings. We evaluate the error correction performance of CodeRepair on a testbed of 10 USRP links deployed in an office building. Each link consists of two USRP nodes. Without loss of generality, we evaluate CodeRepair atop the rate-1/2 802.11 convolutional code, using BPSK modulation. Our experiments are conducted on a 4MHz channel centered at 5 GHz. We observe that the USRP links exhibit substantially different channel conditions. The average BER ranges from 0.06% to 5.37%. Our trace-driven emulation discussed in Section VIII-B will evaluate CodeRepair performance under other bit rates, on a 20MHz channel trace collected using production 802.11n NICs.

Our evaluation focuses on the efficiency of coded bits recovery and partial packet correction of CodeRepair. A key factor that affects error correction performance is the number of padded parities, which depends on the packet size and bit rate. In real networks, packet size may follow different



(a) Effect of parity number. (b) Effect of BER.

Fig. 6. Partial packet recovery of CodeRepair and link layer RS code. In RS-k and CR-k denote link layer RS code and CodeRepair when using k parities for error recovery.

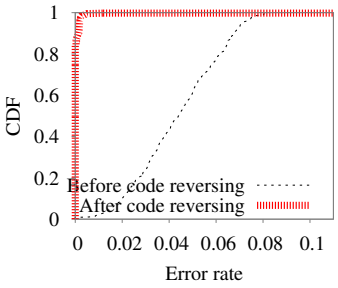


Fig. 7. Error rate of non-gray codes before and after code reversing.

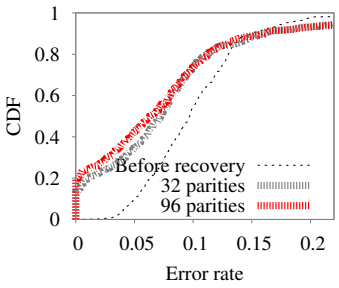


Fig. 8. Error rate of gray-codes before and after SPC decoding.

probabilistic distributions depending on the network traffic. In the following, we conduct controlled experiments to study the effect of the number of parity bits on CodeRepair performance. Specifically, we use a fixed packet size of 420 bytes, where the first 400 bytes are used for carrying data bits, and the last 20 bytes carrying parities. At the receiver, we tune the number of parities used for error correction to study the effects on error correction performance. In Section VIII-B, we will evaluate CodeRepair using real-life network traffic of different packet length distributions.

Coded bits recovery. We now evaluate the performance of coded bits recovery described in Section VI-C. In particular, CodeRepair first performs code reversing to recover non-gray codes, and then conducts SPC decoding described in Section VI-A to correct gray codes. We measure the performance by transmitting 1000 packets on each USRP link. Fig. 7 show the distribution of coded bit error rate for non-gray codes before and after code reversing. The results are measured using total 13872 corrupted packets logged in our testbed. We observe that code reversing recovers a significant amount of gray codes. Specifically, the average error rate is reduced from 4.4% to 0.2%.

Leveraging the low error rate of recovered non-gray codes, CodeRepair further conducts SPC decoding using parities to correct errors in gray codes. Fig. 8 shows the distribution of coded bit error rate for gray codes before and after SPC decoding using 32 and 96 parities. In particular, using 32 parities, CodeRepair reduces gray code error rates at the 25th and 75th percentile from 7.1% and 12.5% to 2.8% and 10.2%, respectively. Using 96 parities, the error rate at the 25th percentile is further reduced to 1.2%. Although the error reduction is not substantial, correcting a small number of gray code errors has significant impact on packet recovery because one coded bit affects the decoding of a group of data bits (see Section IV). In the following, we will evaluate the effect of coded bits recovery on correcting partial packets.

Partial packet recovery. To validate the packet recovery efficiency of CodeRepair, we compare its performance with link-RS, which uses the Reed-Solomon (RS) code of 8-bit symbols at the link layer to correct bit errors. RS code is efficient in correcting the burst bit errors which are common in 802.11 packet, and is widely adopted in previous work [12] [23].

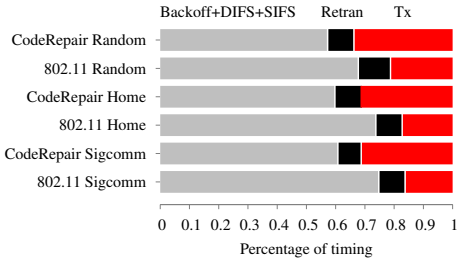


Fig. 12. Histogram of MAC timing measured on an 802.11 transmitter. The figure shows the percentage of time used in transmission (Tx), retransmission (Retran), and free time, which includes the backoff time incurred by channel access, DIFS, and SIFS.

Fig. 6(a) compares the packet recovery performance of CodeRepair and link-RS when using the same number of parities. We observe that CodeRepair performs similarly with link layer RS code when using 64, 96, and 128 parities. However, when the number of parities is limited, CodeRepair significantly outperforms link-RS. Specifically, using only 16 parities, CodeRepair recovers 20.2% partial packets, while link layer RS code only recovers 6.6%. This is because CodeRepair is able to leverage the efficiency of PHY packet recovery (demonstrated in Section IV), where correcting one coded bit improves the decoding success rate of a group of data bits.

Fig. 6(b) shows the packet recovery performance of CodeRepair under different numbers of bit errors. We observe that when using 128 parities, CodeRepair performs similarly with link layer RS code. However, decoding RS code incurs much higher computation overhead than CodeRepair. Moreover, when using 16 parities, CodeRepair significantly outperforms RS code in recovering packets of small number of bit errors. Link layer RS code can correct only one error byte using 16 parities. When bit errors are distributed in different bytes, RS code fails to recover the packet. In comparison, CodeRepair first corrects a small number with coded bit errors, and then leverages the efficiency of PHY packet recovery to correct more bit errors. Previous studies [2] [12] show that bit error rate observed in production 802.11 NIC is very low due to the effect of bit rate adaptation. The result shown in Fig. 6(b) suggests that CodeRepair will be more efficient than link layer FECs in real 802.11 networks.

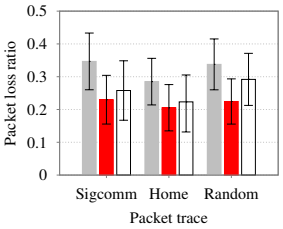


Fig. 9. Packet loss rate.

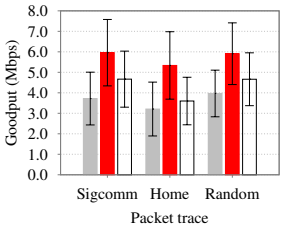


Fig. 10. Goodput.

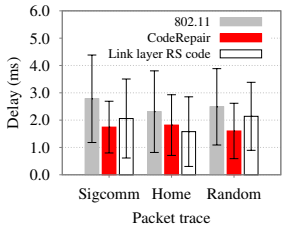


Fig. 11. Packet delay.

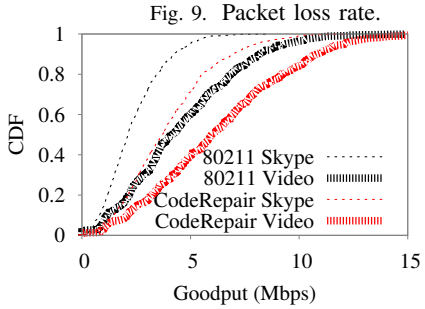


Fig. 13. Goodput gain of CodeRepair on different types of traffic.

B. Impact on Link Layer Performance

In this section, we evaluate the impact of CodeRepair on the link layer performance of 802.11 networks. Due to processing delay, USRP cannot support full-featured link-layer protocols (e.g., real-time rate adaptation). Therefore, we turn to trace-driven emulations to evaluate the end-to-end link performance gain of CodeRepair. To collect fine-grained channel trace, we employ the tool introduced in [9] to sample channel state information (CSI) on a 20 MHz channel using a link of two Intel 802.11n NICs. During trace collection, the sender transmits UDP packets at a speed of 2,000 packets/second. Meanwhile, the receiver moves around the sender at normal walking speed. Total 10-minute trace was collected. We then run our implementations of CodeRepair and the 802.11 like PHY over this channel trace to measure the end-to-end performance. We observe that the SNR during trace collection ranges from 3dB to 20dB. The signal experiences diverse multipath fadings. The maximum difference between subcarrier SNRs ranges from 5 dB to 22dB.

Evaluation settings. To adapt bit rate over time-varying channel, we employ the algorithm proposed in [8], where the transmitter tunes its bit rate based on the effective SNR measured using receiver ACKs. We implemented a binary exponential backoff algorithm where the transmitter doubles its backoff window whenever a packet loss is detected. According to 802.11, the maximum retry limit is set to 6, and the minimum contention window is set to 15.

We compare the performance of CodeRepair with two baselines, including 802.11 and a link layer FEC based scheme, which employs the RS code used in Fig. 6 and piggybacks parities in the padded bits of 802.11 packets. By comparing CodeRepair with link layer RS code, our goal is to demonstrate the advantages of recovering partial packets at the PHY layer. We note that realizing the link layer RS code is very challenging. Recent study [2] shows that a software RS decoder incurs a large processing delay, which cannot meet the timing constraint of 802.11. In addition, RS decoding incurs much higher computational overhead than CodeRepair. Implementing RS decoding in firmware will significantly increase the implementation cost.

A key factor that affects CodeRepair efficiency is the number of padded parities, which depends on the packet size and bit rate. In our evaluation, we study CodeRepair performance using three packet size distributions, including *Random*, *SIGCOMM* and *Home*. In Random, packet size is randomly choosen between 16 bytes and 1580 bytes, which is the maximum size of 802.11 packet. In SIGCOMM [18] and Home, we replay the packet trace collected in real-life 802.11 networks to improve the realism of our evaluation.

Evaluation results. Our evaluation focuses on three link layer metrics, including the packet loss rate, goodput, and packet delay. Higher layer factors, such as TCP reactions, affect how this link layer performance translates to real system throughput.

Fig. 9 compares the packet loss rates of 802.11, CodeRepair, and link layer RS code. We observe that CodeRepair yields the lowest packet loss rate in all packet traces. Specifically, the average packet loss rate of 802.11 in the three packet traces is 32%, which is similar to the result measured in mobile channels using production NICs [12]. CodeRepair reduces packet loss rate from 32% to 21%, recovering 34% partial packets. Moreover, CodeRepair corrects more packet errors than link layer RS code. The result demonstrates the efficiency of PHY packet recovery. Fig. 10 evaluates how this loss rate reduction translates to the goodput gain. As shown in the figure, CodeRepair significantly improves the end-to-end goodput when compared with 802.11. Specifically, the goodput gains of CodeRepair over 802.11 are 1.60x, 1.66x, and 1.49x in SIGCOMM, Home, and Random packet traces, respectively. The average goodput gain is 1.58x. Moreover, CodeRepair outperforms link layer RS code by about 25% in all packet traces. We further evaluate the packet delay in Fig. 11, where the packet delay is measured as the duration from the start of packet transmission to packet reception. CodeRepair reduces packet delay when compared with 802.11. In particular, the average packet delay is reduced by 37% in the SIGCOMM trace.

To explain why a 34% packet loss reduction translates to a 1.58x goodput gain, we analyze the 802.11 MAC timing statistics in Fig. 12. As 802.11 employs binary exponential backoff in MAC, it is very sensitive to packet losses. Whenever a packet loss is detected, it doubles its channel contention window, increasing the channel access overhead. By recovering 34% partial packets, CodeRepair reduces backoff cost and

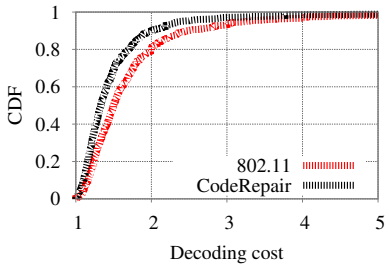


Fig. 14. Decoding cost.

improves the channel utilization of 802.11, yielding significant goodput improvement.

We further study the performance gain of CodeRepair for different traffic types. Fig. 13 plots the CDF of goodputs for Skype traffic and HD Video traffic. The average packet size for Skype and HD Videos are 271 bytes and 1157 bytes, respectively. We observe that the goodput gain of CodeRepair on Skype is 1.62x. In comparison, the gain on Video traffic is 1.36x. CodeRepair performs better on Skype traffic because the ratio of padded bits is higher in packets of smaller size.

C. Decoding Cost

CodeRepair relies on selective re-decoding to correct partial packets. When the packets cannot be recovered after re-decoding, CodeRepair pays extra decoding cost. We define the *decoding cost* as the ratio between the number of decoded bits to the number of successfully received bits. Fig. 14 evaluates the decoding costs of CodeRepair and 802.11 measured in SIGCOMM [18] trace. We observe that CodeRepair yields less decoding cost when compared with 802.11. Specifically, the average decoding costs of 802.11 and CodeRepair are 1.76 and 1.52, respectively. CodeRepair reduces decoding cost by 13.6%.

IX. Conclusion

We present CodeRepair, a practical system for recovering partially corrupted 802.11 packets. CodeRepair piggybacks parities in the padded bits of OFDM, obviating the need of transmitting extra information for error correction. CodeRepair leverages the padded parities to correct errors at the PHY layer, which is significantly more efficient than traditional link layer approaches. Moreover, CodeRepair minimizes the complexity of error recovery by augmenting the single parity code for correcting errors. The extra computation overhead incurred by CodeRepair is only 2% to 7% of the conventional channel decoders. Our results show that CodeRepair improves the system goodput by 59% on lossy 802.11 links.

To improve the interoperability of CodeRepair with production 802.11 devices, we have implemented CodeRepair sender, including the SPC encoding and parity padding, in *userspace* driver. This allows commodity 802.11 devices to communicate with CodeRepair receivers through a simple driver upgrade. Currently, CodeRepair receiver is prototyped on the GNURadio/USRP platform, which has a high processing delay. In the future, we plan to implement CodeRepair receiver in hardware. The extremely low complexity of SPC will lead to an efficient hardware implementation of CodeRepair.

X. Acknowledgments

This work was supported in part by U.S. National Science Foundation under grants CNS-1423221, CNS-0954039 (CAREER), CNS-1239108, CNS-1218718, IIS-1231680, and NSFC (61170296, 61190125).

References

[1] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. In *IEEE Transactions on Information Theory*, 1974.
[2] B. Chen, Z. Zhou, Y. Zhao, and H. Yu. Efficient error estimating coding: feasibility and applications. In *ACM SIGCOMM*, 2010.
[3] J. Erfanian, S. Pasupathy, and G. Gulak. Reduced complexity symbol detectors with parallel structure for isi channels. *Communications, IEEE Transactions on*, 1994.
[4] M. Fossorier, M. Mihaljevic, and H. Imai. Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *Communications, IEEE Transactions on*, 1999.
[5] M. Gowda, S. Sen, R. R. Choudhury, and S.-J. Lee. Cooperative packet recovery in enterprise wlans. In *IEEE INFOCOM*, 2013.
[6] A. Gudipati and S. Katti. Strider: automatic rate adaptation and collision handling. In *ACM SIGCOMM*, 2011.
[7] J. Hagenauer. Rate-compatible punctured convolutional codes (rcpc codes) and their applications. *Communications, IEEE Transactions on*, 1988.
[8] D. Halperin, W. Hu, A. Sheth, and D. Wetherall. Predictable 802.11 packet delivery from wireless channel measurements. In *ACM SIGCOMM*, 2010.
[9] D. Halperin, W. Hu, A. Sheth, and D. Wetherall. Tool release: Gathering 802.11n traces with channel state information. *ACM SIGCOMM CCR*, 2011.
[10] B. Han, A. Schulman, F. Gringoli, N. Spring, B. Bhattacharjee, L. Nava, L. Ji, S. Lee, and R. R. Miller. Maranello: Practical partial packet recovery for 802.11. In *NSDI*, 2010.
[11] K. Jamieson and H. Balakrishnan. Ppr: Partial packet recovery for wireless networks. In *ACM SIGCOMM*, 2007.
[12] K. C.-J. Lin, N. Kushman, and D. Katabi. Ziptx: Harnessing partial packets in 802.11 networks. In *ACM MobiCom*, 2008.
[13] M.-H. Lu, P. Steenkiste, and T. Chen. Design, implementation and evaluation of an efficient opportunistic retransmission protocol. In *ACM MobiCom*, 2009.
[14] E. Magistretti, K. K. Chintalapudi, B. Radunovic, and R. Ramjee. Wifi-nano: Reclaiming wifi efficiency through 800 ns slots. In *ACM MobiCom*, 2011.
[15] R. Mahajan, J. Padhye, S. Agarwal, and B. Zill. High performance vehicular connectivity with opportunistic erasure coding. In *USENIX ATC*, 2012.
[16] A. Miu, H. Balakrishnan, and C. E. Koksal. Improving loss resilience with multi-radio diversity in wireless networks. In *ACM MobiCom*, 2005.
[17] J. Perry, P. A. Iannucci, K. E. Fleming, H. Balakrishnan, and D. Shah. Spinal codes. In *ACM SIGCOMM*, 2012.
[18] A. Schulman, D. Levin, and N. Spring. CRAWDAD data set umd/sigcomm2008 (v. 2009-03-02).
[19] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 1948.
[20] E. Strinati, S. Simoens, and J. Boutros. Performance evaluation of some hybrid arq schemes in ieee 802.11a networks. In *IEEE VTC-Spring*, 2003.
[21] F. Tosato and P. Bisaglia. Simplified soft-output demapper for binary interleaved cofdm with application to hiperlan/2. In *Communications, 2002. ICC 2002. IEEE International Conference on*, 2002.
[22] G. R. Woo, P. Kheradpour, S. Dawei, and D. Katabi. Beyond the bits: cooperative packet recovery using physical layer information. In *ACM SIGCOMM*, 2007.
[23] J. Xie, W. Hu, and Z. Zhang. Revisiting partial packet recovery in 802.11 wireless lans. In *ACM MobiSys*, 2011.
[24] J. Zhang, H. Shen, K. Tan, R. Chandra, Y. Zhang, and Q. Zhang. Frame retransmissions considered harmful: improving spectrum efficiency using micro-acks. In *ACM MobiCom*, 2012.