LEAD: Leveraging Protocol Signatures for Improving Wireless Link Performance

Jun Huang, Yu Wang and Guoliang Xing
Michigan State University
{huangjun, wangyu3, glxing}@cse.msu.edu

ABSTRACT

Error correction is a fundamental problem in wireless system design as wireless links often suffer high bit error rate due to the effects of signal attenuation, multipath fading and interference. This paper presents a new cross-layer solution called LEAD to improve the performance of existing physical layer error correcting codes. While the traditional wisdom of cross-layer design is to exploit physical layer information at upper-layers, LEAD represents a paradigm shift in that it leverages upper-layer protocol signatures to improve the performance of physical layer decoding. The approach of LEAD is motivated by two key insights. First, error correcting code can correct more bit errors when the values of some bits, which we refer to as *pilots*, are known before decoding. Second, some header fields of upper-layer protocols are often fixed or highly biased toward certain values. These distinctive bit pattern signatures can thus be exploited as pilots to assist decoding. To realize this idea, we first characterize bit bias in real-life network traffic, and develop an efficient algorithm to extract pilot bits whose values have assured predictability. We then propose a decoding framework to allow existing error correcting codes to effectively exploit extracted pilots. We implement LEAD on USRP platform and evaluate its performance by replaying real-life packet traces on a testbed of 12 USRP links. Our results show that LEAD significantly improve wireless link performance, while incurring very low overhead. Specifically, LEAD reduces more than 90% error bits for 48.9% packets and corrects all errors for 29.4% packets. Moreover, LEAD improves the end-to-end link throughput by 1.43x to 1.93x over the existing error correction schemes.

1. INTRODUCTION

The last decade witnessed a phenomenal penetration rate of 802.11-based Wireless LANs (WLANs). Nevertheless, as an important part of communication infrastructure, today's WLANs struggle to keep up with the ever growing user demands for higher data rate and coverage. The fundamental challenge for significantly improving WLAN performance lies in the fact that wireless links suffer from high error rate due to the effects of signal attenuation, multipath fading, and interference. The problem is exacerbated by user mobility, which becomes the common case due to the proliferation of portable wireless devices such as tablets and smartphones.

In this paper, we propose LEAD, Leveraging protocol signaturEs for pilot-Assisted Decoding, to reduce BER of 802.11 links. LEAD is motivated by two key observations. First, 802.11 traffic carries numerous upper-layer protocol headers, and some particular fields in these headers are often fixed or biased toward certain values. For instance, the service type field of logical link control header is fixed to 0x0800 for packets that use IP at the network layer. Moreover, previous studies showed that network traffic is bursty in nature [16]. As a result, IP packets that target the same host will lead to temporal bias patterns in the destination field of IP header. Second, the physical layer decoder of wireless receiver can correct more errors when the values of some bits, which we refer to as *pilots*, are predictable at the receiver before decoding. This is due to the fact that the decoding success rates of adjacent bits are dependent on each other, and the predictable values of pilots may affect the decoding of adjacent bits, improving their decoding success rates.

The key idea of LEAD is to leverage protocol signatures as pilots to improve the performance of physical layer decoding. By capturing these protocol signatures, a LEAD receiver can 'guess' some values of received bits (before they are decoded), and then exposes them to the physical layer to guide the decoding of whole packet. We address several challenges in the design and implementation of LEAD. First, due to the inherent dynamics of network traffic, bit patterns of protocol signatures are time-varying, leading to possible pilot mispredictions and decoding errors. Second, protocol signatures are only present in packet headers and hence have limited impact on the decoding accuracy of packet payload. Moreover, bits received at the physical layer are scrambled and encrypted, making it challenging to recognize pilots before decoding. In summary, we make the following main contributions in this paper.

1. We present a novel cross-layer design called LEAD

to reduce bit errors in 802.11-based WLANs. While traditional wisdom of cross-layer design is to exploit physical layer information at MAC layer or above, LEAD represents a paradigm shift in that it leverages upper-layer protocol signatures to improve physical layer decoding. LEAD offers several key advantages. First, the design of LEAD is independent of the underlying error correcting code. Thus the pilot-assisted decoding mechanism of LEAD can be exploited by all existing error correcting codes to reduce error rates of wireless links. Second, LEAD works with existing 802.11 encoders and requires minimum modifications to the 802.11 physical layer. Moreover, it is largely orthogonal to existing error recovery mechanisms (*e.g.*, partial packet recovery [15], link layer forward error correction [17], and block retransmission-based recovery [12]) and can be integrated with them to further reduce packet error rate.

2. We conduct a measurement study to characterize bit bias in real-life network traffic. Our results show that, although bit bias is prominent in upper-layer protocol signatures, it yields distinct patterns that significantly affect the predictability of bit values. We then design an efficient algorithm that classifies bit bias patterns and extracts "good" pilot bits with assured prediction accuracy.

3. We develop a decoding framework that consists of several key components to enable pilot-assisted decoding for existing error correcting codes: *link addressing* allows a decoder to recognize the link identity of a received packet, so that it can leverage the protocol signatures extracted on this link to assist decoding; *pilot spreading* interleaves extracted pilots across the whole packet so that the decoding of payload can leverage adjacent pilots; *pilot reshaping* deals with the impacts of scrambling and encyption on pilot-assisted decoding; and *pilot misprediction handling* detects faulty pilots to avoid decoding errors induced by mispredictions.

4. We develop pilot-assisted decoders for convolutional code, LDPC, and rateless Spinal code, and integrate them with LEAD to leverage protocol signatures. We implement LEAD on GNURadio/USRP platform and evaluate its performance by replaying real-life packet traces on a testbed of 12 USRP links. Our results show that LEAD significantly improve wireless link performance, while incurring very low overhead. Specifically, LEAD reduces more than 90% error bits for 48.9% packets and corrects all errors for 29.4% packets. Moreover, LEAD improves the end-to-end link throughput by 1.43x to 1.93x over the existing error correction schemes.

The rest of this paper is organized as follows. Section 2 summarizes related work. Section 3 introduces the background on convolutional code. Section 4 motivates the design of LEAD. In Section 5 we characterize protocol signature and propose a simple method of pilots extraction. Section 6 introduce the details of LEAD design. Section 7 reports evaluation results and Section 8 concludes the paper.

2. RELATED WORK

Decoding error reduction. Several mechanisms have been proposed to improve the performance of physical layer decoding for 802.11-based WLANs. SoftCast [14], ParCast [18] and Joint source-channel decoding (JSCD) [8], exploit source redundancy to improve video qualities over wireless links. Different with these schemes that are designed for wireless video streaming, LEAD provides a general approach which exploits protocol signatures to improve physical layer decoding. This allows LEAD to work efficiently with diverse upper-layer protocols and applications.

Partial packet recovery. Partially correct packets can be exploited to improve the performance of wireless networks. Selective retransmission based solutions, such as Maranello [12] and PPR [15], reduce error recovery overhead by only retransmitting the corrupted parts of packets. ZipTx [17] exploits MAC layer error correcting codes to correct partial packet. Leveraging the broadcast nature of wireless channel, SOFT [29] allows multiple receivers that hear the same transmission to cooperatively correct faulty bits in a corrupted packet. Compared with these work, LEAD operates at the physical layer to reduce decoding errors. Therefore, LEAD is orthogonal to partial packet recovery schemes, and can be integrated with them to achieve better performance.

Cross-layer design. Previous cross-layer designs focus on exploiting physical layer information at the MAC layer or above to improve wireless performance. Eff-SNR [10], AccuRate [25] and SoftRate [28] exploit different physical layer hints to facilitate bit rate adaptation. Similarly, channel information is used in [22] [5] to provide unequal degrees of error protection for upper-layer traffic flows. Compared with existing work, LEAD represents a paradigm shift in that it leverages upper-layer protocol signatures to improve the performance of physical layer decoding.

Header compression protocols. Previous work [2] [1] exploits protocol signatures for compressing IP/TCP and IP/UDP headers. However, existing header compression protocols are exclusively tied with specific protocols, thus cannot be applied for signature extraction on other protocols. Moreover, the main objective of header compression is to save the transmission time of

state (m_1m_2)	Transition $m_0 = 0$	Transition $m_0 = 1$	Output $m_0 = 0$	Output $m_0 = 1$
00	00	10	00	11
01	00	10	11	00
10	01	11	10	01
11	01	11	01	10

Table 1: The transition and output table of a rate-1/2 convlutional encoder with a constraint length of 2. The two generator polynomials are $n_1 = m_0 \oplus m_1 \oplus m_2$, and $n_2 = m_0 \oplus m_2$, where m_0 is the input bit of the current round, m_1m_2 represents the encoder state.

Tx	Rx	PATH-1			PATH-2		
code	code	Code	Bit	Cost	Code	Bit	Cost
00	01	00	0	1	11	1	1
00	11	00	$\boxed{0}$	3	01	$\boxed{1}$	2
00	00	00	0	3	01	0	3
00	00	00	0	3	00	1	3
00	10	00	0	4	10	0	3

Table 2: An example of using known bits to reduce decoding errors. The true message is 00000, which is encoded using the code defined in Tab. 1. The transmitted and the received codeword sequences are 0000000000 and 0111000010, respectively. The table shows the expected codewords, decoded bits and accumulated costs of the correct path (PATH-1) and a faulty path (PATH-2). If the second bit of the original message is known to be 0, then PATH-2 can be eliminated because its output at the second round is (shown in a box) 1, even though it has a lower cost than the correct path.

headers to improve bandwidth usage. This may work on low bit rate networks, such as Deep Space Networks where the typical bit rate is below 100Kbps. However, today's 802.11n-based WLANs offer a maximum bit rate of 600Mbps, which reduces the transmission time of a 60-byte IP/TCP header to 800 nano-seconds. Recent study shows that most bandwidth resources of 802.11 are consumed in error packet recovery [30] and channel contentions [23] [24]. As a result, the benefits of header compression diminish substantially. Compared with header compression protocols, LEAD aims at exploiting protocol signatures to improve the performance of physical layer decoding on high bit rate 802.11 links, where the overhead of error recovery is significantly higher than that of header transmissions.

3. BACKGROUND ON 802.11 CODING

In this section, we introduce the background of 802.11 coding that is necessary for understanding our approach. Although we focus on convolutional code in this section, our idea of pilot-assisted decoding is also applicable to other error correcting codes (see Sec. 6).

Convolutional code is the default error correcting code used in 802.11. A rate-m/n convolutional code transforms each m-bit symbol of the original message into an

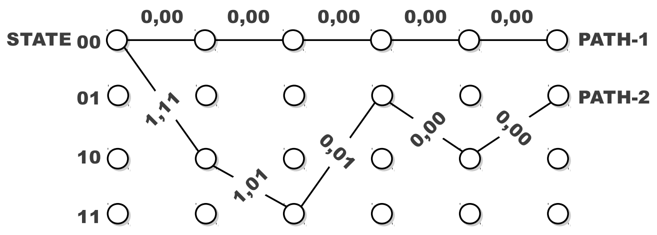


Figure 1: Illustration of convolutional encoding. Each branch is labeled by the input bit that triggers the transition, along with the output codeword.

n-bit codeword using n generator polynomials. Each polynomial defines how modulo 2 additions are performed on the current and the last k input symbols, which are buffered by the encoder using shift registers. The state of encoder is defined by the values of shift registers. Tab. 1 shows an example of a rate-1/2 code with $k = 2$. 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.

Fig. 1 shows the encoding process using the code defined in Tab. 1. PATH-1 and PATH-2 show the traces of state transitions when encoding the messages of 00000 and 11010. The objective of decoding is to find the most likely path that yields the codeword sequence of minimum *distance* to the received codeword sequence. A common metric that quantifies the distance between two paths is the number of different coded bits, namely Hamming distance. For example, given a received codeword 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.

4. MOTIVATION

The key observation that motivates the design of LEAD is that, *if some bits of a packet can be correctly predicted before decoding, then the receiver can exploit these predictions to identify the correct coding path, reducing decoding errors.* Tab. 2 shows an example. We compare the costs of the correct path (PATH-1) and a faulty path (PATH-2). Due to the large amount of errors in received codeword sequence, a conventional decoder will prune PATH-1 due to its higher cost, resulting in flipped bits in output. However, if the value of second bit (which equals to 0) can be correctly predicted, the receiver can eliminate PATH-2 where the second output is 1, therefore reviving PATH-1 in the subsequent steps of decoding. We refer to the bits whose values can be predicted by the receiver before decoding as *pilots*, and the process of using these pilots to identify correct coding path as *pilot-assisted decoding.* Although we use convolutional code in this example, we note that the idea of pilot-assisted decoding is also applicable to other error correcting codes. In Section 7, we will study the gains of pilot-assisted decoding for LDPC [9] and Spinal [19], which are typical examples of block code and rateless code, respectively.

To realize pilot-assisted decoding, a key question is how to obtain the pilots. We propose a novel algorithm that extracts pilots from real-life network traffic. This solution is motivated by the observation that, *upper-layer protocols of 802.11 link have unique signatures in their headers, which have distinct bit bias patterns.* The bit bias refers to the fact that a bit takes the value of 1 or 0 with high frequency, such that it can be predicted by the receiver with high accuracy. There are numerous examples of bit bias in network traffic. The link invariant bits, such as the bits located in MAC addresses, are fixed across all the packets transmitted on a link. In real-life traffic, the service type of logic link control header is biased to 0x0800 because most of packets use IP at the network layer. Moreover, bursty IP traffic [16] will cause a temporary bias in the address fields of IP header. Similarly, the retry flag of MAC header will be biased to 1 in the presence of interference caused bursty packet losses [26] [3].

The above observations motivate the design of LEAD, a cross-layer solution that leverages biased bits in upper-layer protocol headers as pilots to improve 802.11 decoding performance. In the following, we first discuss how to extract pilots from protocol signatures, and then introduce the design of LEAD in detail.

5. EXTRACTING PILOTS FROM PROTOCOL SIGNATURES

In this section, we formalize the definition of bit bias, characterize bit bias in real-life network traffic, discuss the causes of bit bias, and present the pilot extraction algorithm. Our study focuses on data packets. Control packets such as ACKs, RTSs and CTSs have heterogeneous protocol headers. Applying pilot-assisted decoding to these packets is left for the future work.

5.1 Definition of Bit Bias

We define bit bias as *the high probability that a bit takes value of 0 or 1 in a window of packets.* Given a link, we use $\mathbf{T}_{m,n}$ to represent a window of its packets,

$$\mathbf{T}_{n,m} = \begin{bmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \cdots & b_{m,n} \end{bmatrix} \quad (1)$$

where $b_{j,i}$=1 or 0 gives the value of the ith bit in the jth packet, and n defines the maximum packet size. If the size of jth packet is shorter than i, we define that $b_{j,i}$ takes 0 or 1 at equal probability. The bias of the bit i can be calculated as follows,

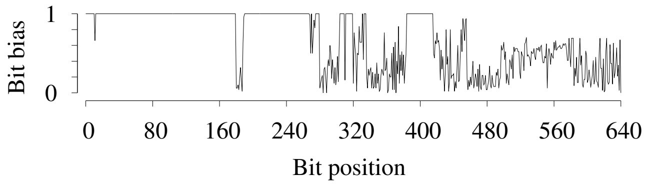


Figure 2: Bit bias measured in a window of 100 packets.

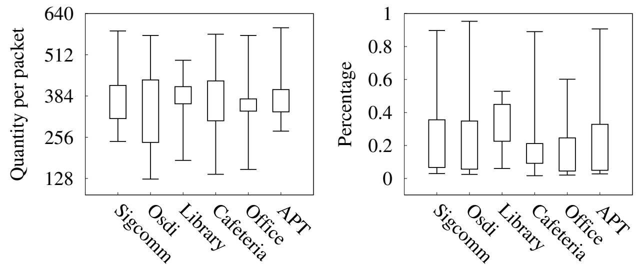


Figure 3: Quantities and percentages of fixed bits. Min, Max, 25th, and 75th percentiles are shown.

$$\beta_i(\mathbf{T}_{n,m}) = 2 \times \left| \frac{\sum_{k=1}^{m} b_{k,i}}{m} - \frac{1}{2} \right| \quad (2)$$

which gives the higher probability that 0 or 1 is observed on bit i. Specifically, in a window of m packets $\mathbf{T}_{n,m}$, if bit i is perfectly biased, i.e., fixed to 0 or 1, $\beta_i(\mathbf{T}_{n,m}) = 1$. If the bit is perfectly unbiased, i.e., takes 0 or 1 at equal probability, $\beta_i(\mathbf{T}_{n,m}) = 0$.

5.2 Bit Bias of Real-life Network Traffic

In the following, we present a measurement study to characterize bit bias in real-life network traffic. Our objectives are to validate the abundance of biased bits and to understand the temporal variations of bit bias.

Our measurement uses real-life traffic traces collected in WLANs that differ from each other in both scales and deployment scenarios. SIGCOMM'08 [21] and OSDI'06 [6] were collected in WLANs of hundreds of users attending the two conferences. PDX [20] includes traces collected in a library, an office building, and a cafeteria in a university. Our trace, named as APT, was collected in an apartment from an 802.11n WLAN that serves three laptops, one tablet, and two smartphones. For each packet in the above traces, application layer data was stripped out to preserve user privacy. Therefore, our measurements focus on the bit bias of the first 80 bytes of each packet, which include all protocol headers up to the transport layer.

We first validate the abundance of biased bits. For each link in the datasets described above, we measure bit bias using the first 100 packets. Fig. 2 plots the result for a randomly selected link. Among the 640 bits studied, 314 are fixed, *i.e.*, have bias of one. Fig. 3 shows the quantities and percentages of fixed bits, where the percentage is computed as the ratio of fixed

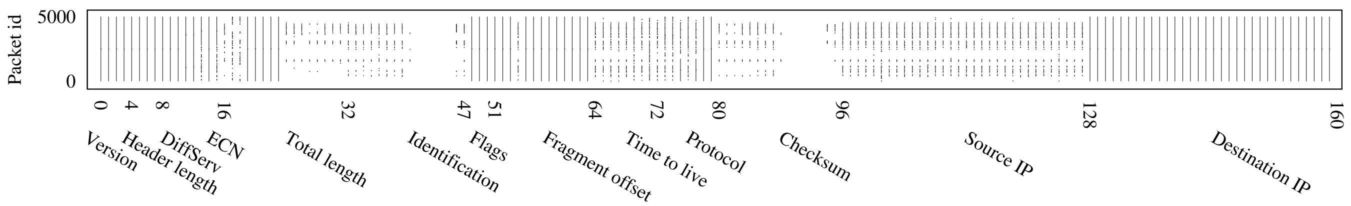


Figure 4: Temporal variations of bit bias in IPv4 header. A bar at a specific bit position indicates that the bit is fixed in the packets represented by the bar. For clear illustration, we only plot bars that contain more than 100 packets.

bits to the total traffic. Specifically, in the SIGCOMM'08 trace, each IP packet contains an average of 363 fixed bits, which account for 17.5% of the total traffic.

In Fig. 2 and Fig. 3, the bit bias is measured in a window of 100 packets. It is critical to understand how bit bias varies at different scales of windows. To this end, we collect bit traces using 5000 packets on an uplink (*e.g.*, packets are transmitted from a client to an access point) that uses IPv4 at the network layer. Fig. 4 plots the trace measured in the IPv4 header. We observe two distinct patterns of bit bias. First, some bits, such as the bits located in the destination and differentiated service (DiffServ) field, are *consistently* biased. While other bits, especially the bits in the source and time to live (TTL) field, experience *bursty* biases. Similar results were observed in other protocol headers on most links in the datasets.

5.3 Causes of Bit Bias

Bits in protocol headers have specific semantics. For example, the 11th bit of 802.11 MAC header is defined as the retry flag; the first four bits of IP header denote the version of IP protocol. In our measurements, we identified two classes of bit biases, namely *consistent bias* and *bursty bias*. In the following, we explain the causes of bit bias based on bit semantics. Tab. 3 summarizes our discussion.

- **Consistently bias**. Bits that describe link invariant protocol addresses, such as MAC addresses, the IP address of WLAN client, are fixed on each link. Bits that denote frame and service types are also consistently biased because of the uneven distribution of packet or service types in network traffic. For instance, most data packets use IP at the network layer, which is denoted by 0x0800 in the service type field of logic link control header.

- **Bursty bias**. Some bits experience bursty biases because of the burstiness of network traffic [16]. For example, on a link that uses IPv4 at the network layer, bursty TCP traffic causes the IP protocol number biased to 0x06, and leads to temporary biases on the bits located in the source and destination fields of the TCP header. Moreover, network dynamics, such as wireless channel variations and network congestions, may affect bit bias. For example, bursty packet losses [26] [3] may cause the retry flag of MAC header biased to 1. Similarly, the value of IP TTL is biased when many packets of an IP flow traverse paths of the same hop count through the Internet.

5.4 Pilot Extraction

In this section, we present an algorithm that leverages bit bias to extract pilots from the first 80 bytes of each packet, which include all protocol headers up to the transport layer. Since bit values of packet may appear random after encryption, the extractor works on decrypted bits to measure bit bias. The extractor design is driven by the following objectives.

Transparent to upper-layer protocols. The extractor should NOT be tied with specific upper-layer protocols, such that the extractor can be easily deployed on diverse wireless platforms where different protocols are operated at the upper-layers. Previous solutions [1, 2] predict bit values based on bit semantics of protocol headers. For example, as the first four bits in IP header denotes the version of IP protocol, their values can always be predicted by the receiver if the protocol version is known. However, existing solutions are exclusively rely on prior knowledge of protocol header structures, thus cannot be applied to other protocols. Moreover, as bits in protocol headers are coded during transmission, knowledge about upper-layer protocols is not available before the packet is decoded.

Maximizing pilot number while achieving assured prediction accuracy. Physical layer decoder can correct more bit errors with more predictable bits. However, due to the dynamics of network traffic, bit values in protocol headers are time-varying, making it challenging for accurate prediction. To maximize the performance gain of pilot-assisted decoding, the extractor should maximize the number of pilots while achieving assured prediction accuracy.

Extractor overview. Instead of relying on prior knowledge of bit semantics, we propose a simple extractor, which sniffs network traffic at run-time and selects bit i as a pilot if it is fixed in last k_i packets. Because different bits may have different bias levels and bias patterns, the extractor tunes k_i for each bit i to opti-

Pattern	Cause	Examples
Consistent bias	Link invariant data	MAC addresses, IP address of WLAN client.
	Uneven distribution of service types	Frame type in MAC header, service type in logic link control
Bursty bias	Network dynamics	MAC retry flag, IP TTL, IP explicit congestion control (ECN).
	Bursty traffic	IP destination of uplink, IP source of downlink, IP protocol number, source and destination port of TCP and UDP header.

Table 3: **Causes of bit bias in 802.11 traffic.**

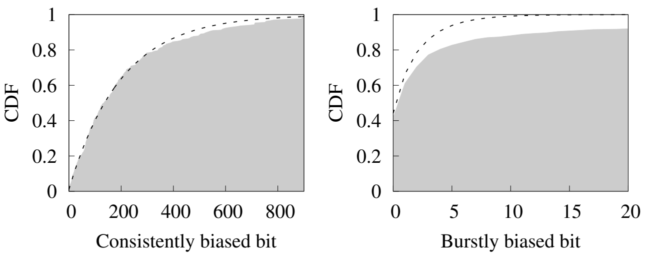


Figure 5: **The burst length CDFs measured on a consistently biased bit and a bursty biased bit. The former is from the service type field of logic link control header. The latter is located in the IP source of a downlink. The dashed curves are the theoretical burst length distributions of the memoryless bits with the same biases. Difference between the distributions quantifies the burstiness of bit bias.**

mize pilot extraction performance. Given a sequence of n packets, for the ith bit, we use $\mathcal{N}_i(k)$ to denote the times that it is selected as pilot using a history size k. We have $\mathcal{N}(k_0) \geq \mathcal{N}(k_1)$, if $k_0 \leq k_1$. This is because $\mathbf{h}_i(t, k_0) \subset \mathbf{h}_i(t, k_1)$, where $\mathbf{h}_i(t, k)$ is the set of last k observations on bit i when packet t is received. Thus if bit i is selected using $\mathbf{h}_i(t, k_1)$, it will also be selected using $\mathbf{h}_i(t, k_0)$. Let $\mathcal{E}_i(k)$ to be the pilot misprediction rate. Our goal is to tune k_i for each bit i, such that,

$$k_i = \underset{t}{\operatorname{argmin}}\{\mathcal{E}(t) \leq \Delta\} \tag{3}$$

where Δ is the pre-defined threshold on pilot misprediction rate. To reduce decoding errors, Δ should be lower than the BER of conventional decoder. Recent study [7] shows that the BER observed on commodity 802.11 radios typically ranges from 0% to 10%. In this work, we adopt a threshold of 0.5%. We will study the effect of pilot misprediction in Section 7.4.

A key factor that affects the predictability of protocol signature is the pattern of bit bias. Fig. 5 compares the burst length CDFs for a bursty biased bit and a consistently biased bit, where the burst length is the size of packet window in which a bit is fixed. The dashed curves in the figure show the theoretical burst length CDFs of the memoryless bits with the same bias, which follow the Bernoulli distribution. As shown in the figure, the two bits have distinct probabilistic distributions, calling for different strategies to control the history size of pilot extraction. Specifically, a small history size should be used for a bursty biased bit to capture its temporal predictability. In contrast, a large history size should be used on a consistently biased bit to assure that the measured predictability is statistically significant. In the following, we first present a bias pattern classifier, and then discuss how to optimize the history size for different bias patterns. Finally, we discuss how the pilot extractor works in practice.

Classifying bias patterns. We propose a metric that formally models the busrtiness of bit bias. Let $\mathcal{P}_i(t)$ to be the CDF of burst length for bit i, and $\mathcal{Q}_i(t)$ to be the theoretical burst length CDF for the memoryless bit, which follows Bernoulli distribution. To quantify bias burstiness, we define μ_i as the Kullback-Leibler divergence between $\mathcal{P}_i(t)$ and $\mathcal{Q}_i(t)$ measured on bit i, which can be expressed as,

$$\mu_i = \mathcal{D}_{KL}(\mathcal{P}_i||\mathcal{Q}_i) = \left|\sum_t \mathcal{P}_i(t) \ln \frac{\mathcal{P}_i(t)}{\mathcal{Q}_i(t)}\right| \tag{4}$$

where $\mathcal{P}_i(t)$ can be measured using empirical observations of burst lengths. Our empirical study shows that a threshold of 2 is sufficient to classify bursty bias from consistent bias, *i.e.*, bit i is classified as a bursty biased bit if its bias burstiness satisfies $\mu_i \geq 2$.

Optimization history size. The extractor selects bit i as a pilot if it is fixed in the history of last k_i packets. In the following, we discuss how to tune k_i to meet the objective of Eq. (3).

Consistently biased bits. When using a consistently biased bit as pilot, the misprediction rate can be computed as $0.5 - 0.5\beta$, where β is the bit bias. To satisfy Eq. (3), β must be larger than $1 - 2\Delta$. The extractor should use a large k such that if bit bias is lower than $1 - 2\Delta$, the probability that the bit is fixed in last k packets is small. This probability can be computed as $(0.5 + 0.5\beta)^k$. Using a confidence level of 95%, k should satisfy $(0.5 + 0.5\beta)^k < 0.05$. Therefore, we can calculate the minimum history size for a consistently biased bit as $k = \frac{\log 0.05}{\log(1-\Delta)}$.

Bursty biased bits. Given a bursty biased bit which has been fixed in last t packets, the probability that it will not change in the next packet can be computed as $\Pr(t+1|t) = \frac{1-\mathcal{P}(t+1)}{1-\mathcal{P}(t)}$, which is the conditional probability that the burst length is longer than $t+1$, given that the length of current burst is t. $\mathcal{P}(t)$ is the burst length CDF. As a result, the minimum history size can be derived as $k = \operatorname{argmin}_t \left\{ \frac{1-\mathcal{P}(t+1)}{1-\mathcal{P}(t)} > 1 - \Delta \right\}$. However, calculation of k requires a fine grained burst length CDF, which is difficult to obtain without a large number of samples. We address this problem by curve fitting. Observing that $\mathcal{P}_i(t)$ of bursty biased bit is featured by a long tail, we fit the measured burst length CDF to the power-law Pareto distribution, whose CDF is given by,

$$\mathcal{F}(t) = \begin{cases} 1 - \frac{1}{t^{\alpha}}, & t > 1 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where α can be derived from on-line measurements using maximum likelihood estimation. Given n observations of burst lengths, denoted by $\{x_1, ..., x_n\}$, we have $\hat{\alpha} = \frac{n}{\ln x_1 + ... + \ln x_n}$. Therefore, we can calculate the minimum history size for a bursty biased bit as,

$$k = \frac{1}{1 - \exp\left(\frac{\log(1-\Delta)}{\hat{\alpha}}\right)} - 1. \tag{6}$$

Putting it together. The extractor maintains a moving window of N correctly received packets, and updates the burst length CDF $\mathcal{P}(t)$ and the burstiness metric μ for each bit every T seconds. Our empirical study shows that $T = 10s$ and $N = 600$ work efficiently. For each sender, the receiver allocates 160 bytes in the physical layer RAM to store its pilots. Each bit in the first 80 bytes takes two bits: one is used to indicate whether it is a pilot; the other is the pilot value. At the MAC layer, the extractor maintains a counter for each bit to monitor the length of its ongoing burst. A bit is selected as pilot to decode the next packet, if its counter is longer than the computed minimum history size. The physical layer RAM is updated when new pilots are extracted, or any existing pilots are removed.

6. EXPLOITING PILOTS TO IMPROVE DECODING PERFORMANCE

In this section, we present a decoding framework that allows existing error correcting codes to effectively exploit extracted pilots. In the following, we first present the design of LEAD framework, and then discuss pilot-assisted decoding algorithms.

6.1 LEAD Framework

Several key challenges must be addressed to enable pilot-assisted decoding on 802.11 links. First, since extracted pilots differ on different links, LEAD needs to know the link address of a received packet for choosing pilots before decoding. Second, as bits are typically scrambled and encrypted during transmission, protocol signatures extracted at upper-layer cannot be directly used as pilots to direct physical layer decoding. Third, as pilots are extracted from packet headers, they have limited impact on the decoding accuracy of payloads. Fourth, due to the dynamics of network traffic, bit predictabilities are time-varying, which leads to possible mispredictions of pilots, causing decoding errors at the receiver.

We now introduce the framework of LEAD to address the above challenges. The framework is composed of four key components, including *link addressing*, *pilot reshaper*, *pilot spreader*, and *pilot misprediction handler*.

Link addressing. To identify the link address before decoding, LEAD adopts a link addressing scheme, where the sender and receiver MAC addresses are hashed into a one-byte identity, which is then appended after the 802.11 SIGNAL symbol. At the receiver, LEAD parses the identity before decoding, and then chooses the pilots that were extracted for this link to direct decoding. In practice, multiple links may be hashed to the same identity. Such identity collision can be easily detected by LEAD at the MAC layer. When collision occurs, LEAD falls back to the conventional decoding mode. Because the collision probability of two links is only 1/256 when a one-byte link identity is used, its effect on LEAD performance is negligible.

Pilot spreading. LEAD employs a MAC layer interleaver to uniformly spread pilots extracted from protocol headers over the whole packet, such that the decoding of payload bits can be benefited by exploiting adjacent pilots. At the receiver, the reverse operation is performed to recover bit orders after pilot-assisted decoding. Note that the default 802.11 interleaver cannot serve this purpose. The goal of 802.11 interleaving is to permutate coded bits located in the same OFDM symbol, to mitigate bursty transmission errors in frequency selective channels. Its interleaving depth is at most 216 bits (i.e., when QAM64 and rate-3/4 code, and 48 data sub-carriers are used), which is not enough to spread pilots over large size 802.11 packets.

Pilot reshaping. Bit values of a packet may appear random after encryption and scrambling. As discussed in Sec. 5, the pilot extractor of LEAD addresses this issue by extracting pilots after decryption and descrabmling. However, since the extracted pilot values may not match those received at the physical layer, the pilot cannot be directly applied in the physical layer decoding. LEAD addresses this issue in the pilot reshaping module. Specifically, before using pilots to decode a packet, the receiver first scrambles and encrypts the pilots, such that the values of reshaped pilots will match those received at the physical layer. To allow the pilot reshaper to use keystream for encryption before decoding, the LEAD sender appends the header of encryption protocol after the 802.11 SIGNAL symbol. When a packet is received, the receiver first decodes the encryption header to retrieve the keystream. A practical concern is the processing delay caused by reshaping. We

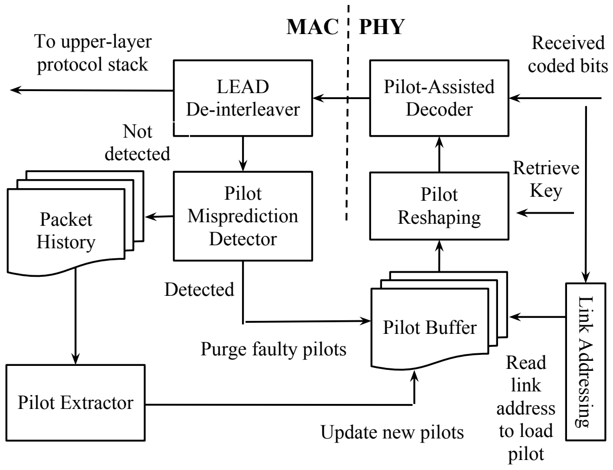


Figure 6: The architecture of LEAD receiver. Standard 802.11 operations such as OFDM demodulation, de-interleaving, and descrambling are not included.

will evaluate the delay in Section 7.4.

Handling pilot mispredictions. Pilot misprediction occurs when the pilot used in decoding does not match the value of a received bit, causing decoding errors. LEAD adopts a checksum-based method to detect pilot mispredictions. Ideally, the checksum should be calculated over only the pilot bits. However, this is difficult because pilots are extracted at the receiver at run-time, and notifying the sender of which bits are pilots may incur prohibitive messaging overhead. Another approach is to compute the checksum for the first 80-byte of the packet, from which the pilots are extracted. This approach is also inefficient because the checksum failure may be caused by the errors of non-pilot bits.

LEAD employs a block checksum mechanism to address this problem. Specifically, the sender divides the first 80 bytes into blocks and then calculate checksum for each block. If a checksum fails, the receiver stops using the pilots extracted from the corresponding block, until the pilots are updated using a correctly received block. This approach avoids using faulty pilots to decode a sequence of packets, therefore limiting the detrimental effect to a single transmission. In our implementation, we use two block checksums, and employ CRC-4 for checksum computation.

Putting it together. In the following, we describe the decoding process of LEAD by tracing the transmission and reception of an 802.11 packet.

At the transmitter, LEAD first uses a MAC layer interleaver to spread bits in protocol headers over the whole packet. After the packet is encrypted and scrambled, LEAD inserts its own header after the 802.11 SIGNAL symbol. The LEAD header contains the one-byte link identity, the two block checksums computed using CRC-4, and the encryption protocol header. To notify the receiver that the packet uses LEAD protocol for decoding, the sender flips the reserved bit in 802.11 SIGNAL symbol as the delimiter of LEAD packet.

Code	Decoder	No. of lines	
		Standard	Pilot-assisted
Convolutional	BCJR [4]	143	147
LDPC	BP [9]	87	97
Spinal	Bubble [19]	158	181

Table 4: Pilot-assisted decoders of LEAD.

Fig. 6 shows the architecture of a LEAD receiver. When a LEAD packet is received, the receiver first decodes the LEAD header, parses the link address to choose pilots, and retrieves keystreams to reshapes pilots through scrambling and encryption. The reshaped pilots are then fed into the pilot-assisted decoder. If decoding is successful, LEAD logs the first 80 bytes to the MAC layer packet history, and updates the physical layer buffer if new pilots are extracted. Otherwise, LEAD uses the two block checksums embedded in LEAD header to locate bad blocks that contain mispredicted pilots. If detected, all pilots in the bad blocks will be purged from the physical layer buffer to prevent further decoding errors.

6.2 Pilot-Assisted Decoders

The design of LEAD is independent with underlying error correcting code. Thus LEAD can be exploited by all existing error correcting codes using their pilot-assisted decoders. In this work, we augment three decoding algorithms, including BCJR [4] of convolutional code, the belief propagation (BP) decoder [9] of LDPC code, and the bubble decoder of rateless Spinal code [19], to perform pilot assisted decoding. Tab. 4 summarizes our implementations of these pilot-assisted decoders. Due to space limitation, we only introduce the design of pilot-assisted BCJR. Details of other decoding algorithms are available in a technical report [13].

Pilot-assisted BCJR decoder. BCJR is a popular soft-output decoding algorithm for maximum *a posteriori* decoding. Given a sequence of k received codewords $o_{1:k}$, the BCJR decoder infers the coding input with the maximum *a posteriori* probability estimation. Specifically, the decoder performs forward recursion and backward recursion to compute the posterior marginals of all states for inference. During the forward recursion, the decoder computes the probability that the coder ends up at a particular state s at time t given the first t observations in received codeword sequence.

$$\alpha_t(s) = \Pr(s|o_{1:t}) = \sum_{s'} \alpha_{t-1}(s')\gamma_t(s,s') \qquad (7)$$

where $\gamma_t(s,s')$ computes the probability of state transition from s to s' at time t, which depends on the

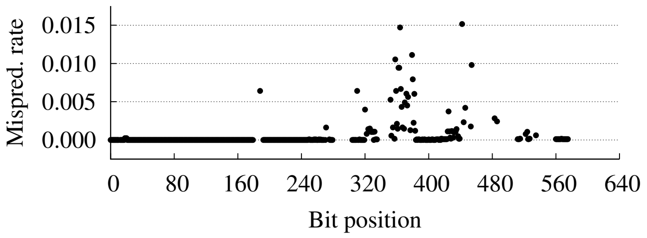


Figure 7: **Misprediction rate of extracted pilots when using a threshold of 0.005.**

distance between received codeword at time t and the expected codeword given the transition. During the backward recursion, the decoder computes the probability of observing the remaining codewords given a start point t.

$$\beta_{t-1}(s) = \Pr(o_{t:k}|s) = \sum_{s'} \beta_t(s')\gamma_t(s, s') \tag{8}$$

Based on the Bayes rule, the probability that the coder reaches state s at time t given the received codeword sequence $o_{1:k}$ can be calculated by

$$\lambda_t(s) \propto \Pr(s|o_{1:t}) \Pr(o_{t+1:k}|s) \tag{9}$$

Since the encoder starts and ends up at state 0, the BCJR decoder can be initialized as follows

$$\begin{aligned} &\alpha_0(0) = \beta_k(0) = 1 \\ &\alpha_0(s) = \beta_k(s) = 0 \quad \text{for all } s \neq 0 \end{aligned} \tag{10}$$

Let $\mathbf{s^0}$ and $\mathbf{s^1}$ denote the state where the starting bit is 0 and 1 respectively. The soft-output of the decoder is the log-likelihood for bit y_t, which is calculated as

$$\text{LLR}(y_t) = \log \frac{\sum_{s \in \mathbf{s^1}} \lambda_{t+1}(s)}{\sum_{s \in \mathbf{s^0}} \lambda_{t+1}(s)} \tag{11}$$

Pilot-assisted BCJR augments the standard BCJR by improving the estimation of $\lambda_t(s)$ in Eq. (9), where the $\gamma_t(s, s')$ in Eq. (7) and Eq. (8) equals to 0 if the bit that corresponds to the transition $s \to s'$ does not match the pilot bit. Therefore the pilot bits reduce the space of possible transitions and improves the estimation quality of adjacent data bits.

7. EVALUATION

LEAD is implemented on top of the OFDM modules of GNURadio/USRP platform. LEAD takes demodulated symbols as input, convert complex symbols on constellation to soft coded bits using a soft-output demapper [27], and then performs pilot-assisted decoding through the procedures illustrated in Fig. 6.

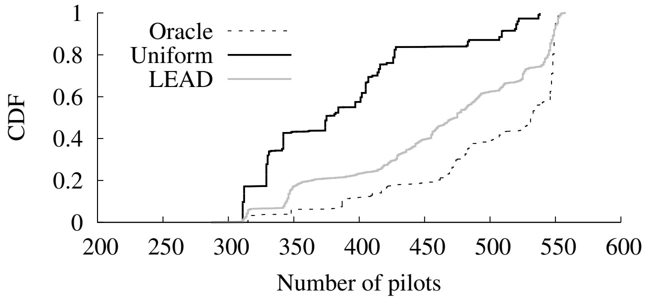


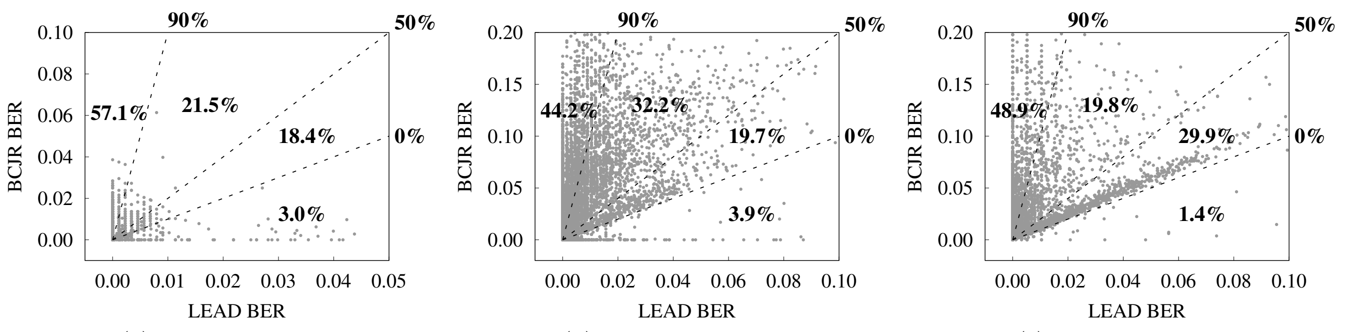
Figure 8: **CDF of the number of extracted pilots.**

In this section, we evaluate the performance of LEAD. Our evaluation centers around four aspects. (1) How efficiently can LEAD extract decoding pilots from upper-layer protocols? (2) Compared with standard decoder, how much performance gain does LEAD yield on decoding 802.11 packets in real-life traffic? (3) How much is the overhead of LEAD? and (4) How does LEAD perform on different error correcting codes?

7.1 Pilot Extraction

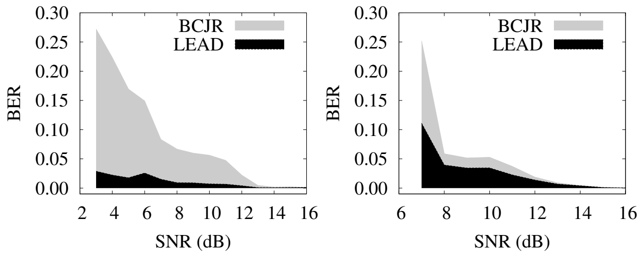
We first evaluate the performance of pilot extractor proposed in Section 5.4. LEAD extracts a bit as pilot if it is fixed in last k packets. Because bit predictability depends on the bias level and bias pattern, LEAD classifies bias patterns, and tunes the history size k for each bit, to maximize the number of extracted pilots while limiting the misprediction rate under a predefined threshold. In our implementation, we adopt a threshold of 0.005, and update the history size every 10 seconds. In the following, we first study how efficiently LEAD controls the pilot misprediction rate, and then evaluate the quantity of extracted pilots. Our evaluation is performed using five real-life traffic traces introduced in Section 5. Due to space limitation, we only present the results measured using the SIGCOMM'08 trace [21]. We observe similar results on other traces.

Fig. 7 shows the average misprediction rates for the bits located in the first 80 byte of packets, when using a misprediction threshold of 0.005. The results are measured using 14968 packets collected on a WLAN downlink. Among the first 80 bytes of packet, total 575 bits were used as pilots. Only 14 bits have an average misprediction rate higher than 0.005. This result shows that our history size estimation (described in Section 5.4) can effectively bound the misprediction rate. We further evaluate the quantity of pilots extracted by LEAD. Two baseline algorithms are employed for performance comparison. The first baseline, named Uniform, uses the same history size for all the bits, which is set as the minimum value that bounds the average misprediction rate below 0.005 on all the bits. The second baseline, named Oracle, chooses the optimal history size for each bit. For both baselines, the history sizes



(a) QPSK, rate-2/3. (b) QAM16, rate-1/2. (c) QAM64, rate-2/3

Figure 9: Bit error rates of standard BCJR and LEAD. Each point represents a pair of BERs of a packet decoded by the standard BCJR and LEAD. We divide the points into 4 regions based on the percentage of reduced error bits when using LEAD in decoding, including increasing the BER, reducing 0% 50% error bits, reducing 50% 90% error bits, and reducing more than 90% error bits. The numbers in the figure give the percentages of error packets in the corresponding region.



(a) Smaller than 256B. (b) Larger than 256B.

Figure 10: BER for packets that are smaller and larger than 256 bytes.

are updated every 10 seconds. To implement Uniform and Oracle, we first run the extraction algorithm using all history sizes in each interval of 10 seconds, and then look back to choose the history size that meets the misprediction bound. Fig. 8 shows the distributions of pilot quantity measured on 24 links. The average numbers of pilots extracted by Uniform, LEAD, and Oracle are 375, 477 and 522, respectively. LEAD outperforms Uniform by 27.2%, which validates the effectiveness of our per-bit history size optimization mechanism based on the level and pattern of bit bias.

7.2 Bit Error Reduction

Experiment setting. We now evaluate the performance of LEAD on convolutional code, which is the default error correcting code of 802.11. To quantify the decoding efficiency, we compare the BERs of packets decoded by the standard BCJR algorithm and LEAD, which uses the pilot-assisted BCJR algorithm described in Section 6.2. We deployed a testbed of 12 USRP links in an office building. Each link consists of two USRP nodes. Our experiments are conducted on a 4MHz channel centered at 5 GHz. We observe that the USRP links in our testbed differ substantially in their channel conditions. The average signal-to-noise ratios (SNR) measured on the channels of USRP links range from 7 dB to 20 dB. Our trace-driven simulations discussed in Sec. 7.3 will evaluate LEAD performance on 20MHz channel trace collected using production 802.11 NICs.

To improve the realism of our evaluation, we replay the traffic trace collected in SIGCOMM'08 [21] on our USRP testbed. For each USRP link, we randomly select a link from the trace, and replay 10 minutes of traffic at the USRP transmitter. The experiment is repeated multiple times, each time using a different combination of modulation and coding schemes. For a fair comparison over time-varying wireless channels, we log the demodulated symbols at the receiver, and decode them using LEAD and the standard BCJR algorithm to compare their BERs.

BER reduction. Fig. 9 compares the BER of standard BCJR and LEAD under three bit rates. We observe that LEAD outperforms the standard BCJR in all settings. Specifically, when QAM64 and rate-2/3 coding are used, LEAD reduces more than 90% error bits for 48.9% packets. Moreover, for 29.4% packets, LEAD mitigates all bit errors. Only in 1.4% packets, LEAD induces more bit errors due to pilot mispredictions. We observed similar results on other physical layer settings. Unless stated otherwise, we will use QAM16 and rate-1/2 coding in the following experiments.

Effect of packet size. Fig. 10 compares the performance of LEAD for packets of different sizes. We observe that LEAD achieves significant performance gain for packets that are shorter than 256 bytes. Specifically, when the SNR of channel is 6 dB, the BERs of LEAD and standard BCJR are 0.025 and 0.149, respectively. LEAD reduces bit errors by 83%. This is because small packets have a higher percentage of pilots. We note that small packets, which are used extensively by Internet telephony or gaming applications, account for a major fraction of real-life network traffic. For example, in the SIGCOMM'08 and our APT trace, 57.3% and 49.5% packets are smaller than 256 bytes.

		BER ($\times 10^{-3}$)	
Transport	Network	BCJR	LEAD
TCP	IPv4	25.4	4.9
UDP	IPv4	18.5	2.9
IGMP	IPv4	62.9	1.7
	IPv6	48.6	3.1
	ARP	10.2	0.7
	LLDP	31.2	2.6

Table 5: Decoding BERs of upper-layer protocols.

Different upper-layer protocols. We further evaluate the gain of LEAD for different upper-layer protocols. Tab. 5 compares the BERs of packets decoded by LEAD and the standard BCJR. We observe that LEAD consistently outperforms BCJR for all protocols. LEAD is especially efficient for those protocols that use small packet sizes. For example, for IGMP, LEAD reduces 97.3% bit errors compared to the standard BCJR decoder.

7.3 Impact on Link Layer Performance

In this section, we present trace-driven simulations to evaluate the impact of LEAD on link layer performance. Due to the processing delay, USRP cannot support full-featured link-layer protocols (*e.g.*, real-time rate adaptation). Therefore, we turn to trace-driven simulations in this section to evaluate the end-to-end link performance gain of LEAD. To collect fine-grained channel trace, we employ the tool introduced in [11] 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 LEAD and standard 802.11 over this channel trace to compare their performance.

Link layer settings. To adapt bit rate over time-varying channel, we employ the algorithm proposed in [10], 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 drop is detected. According to 802.11, the maximum retry limit is set to 6. To evaluate the impact of LEAD on link layer error recovery mechanisms, we implemented three protocols, including *Maranello* [12], *link layer forward error correction (FEC)* [17], and *partial packet retransmission (PPR)* [15].

- *Maranello.* Maranello [12] is a block retransmission based partial packet recovery protocol. Specifically, a corrupted packet is divided into blocks at the receiver that computes a 32-bit checksum for each block and returns the checksums to the sender to identify corrupted block(s). To reduce recovery overhead, Maranello only retransmits corrupted block(s). In our implementation, we set the block size to 64 bytes, which is compliant to the setting introduced in [12].

- *Link layer FEC.* To evaluate the impact of LEAD on link layer error correcting schemes, we implemented the protocol introduced in [17]. To correct bit errors, Reed-Solomon code is adopted at the link layer. RS code breaks each packet into n-bit symbols, and then groups each r symbols to a codeword, where $r \leq 2^n$. Two redundant symbols of RS code can correct up to one bad symbol in a codeword. In our implementation, we set n=8, which is the popular choice on byte-oriented systems. When an ACK timeout is detected, the transmitter sends the *optimal* amount of parity symbols, which is the minimum number of symbols that assures error correction.

- *PPR.* The basic idea of PPR [15] is to exploit physical layer decoding confidence to estimate bit error probabilities. The sender only retransmits the bits of low decoding confidence to avoid the costs of retransmitting correct bits.

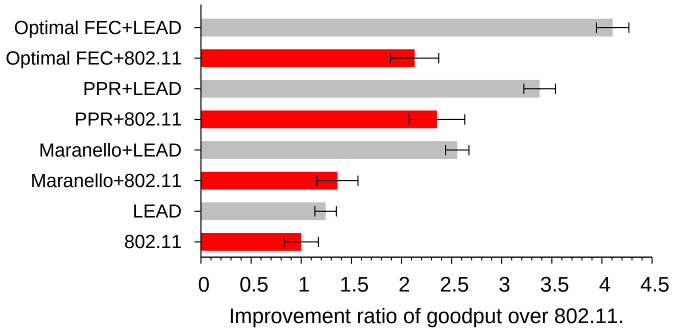


Figure 11: Link layer goodput gains of LEAD over 802.11 when different error recovery protocols are used at the link layer.

Evaluation results. We evaluate the impact of LEAD on link layer performance by replaying real-life traffic traces over our channel trace. Total 100 links are randomly selected from the SIGCOMM'08 trace. For each link, we replay a 10-minute traffic trace and measure the link layer goodput at the receiver. Higher layer factors, such as TCP reactions, affect how this link layer goodput translates to real system throughput. We will integrate TCP implementation to study the performance of LEAD in our future work.

We integrate LEAD with the above three error recovery protocols. The performance of total 8 link layer settings is studied. The results are shown in Fig. 11.

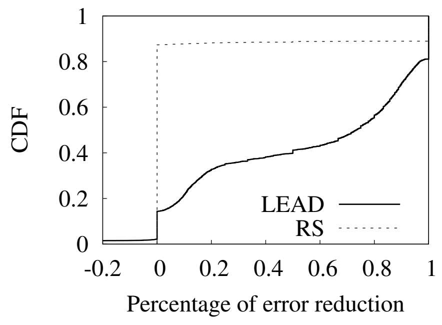


Figure 12: Percentage of error reduction of LEAD and RS code.

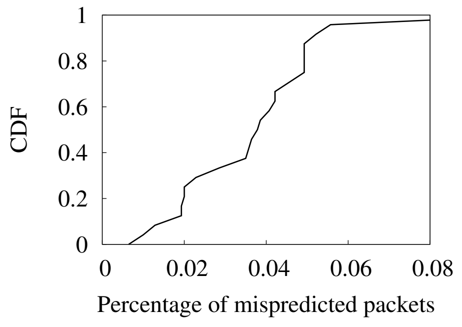


Figure 13: Pilot-misprediction caused packet losses.

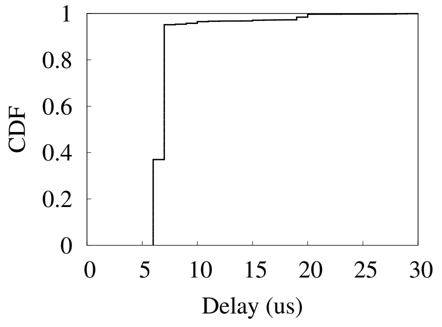


Figure 14: Processing delay of software pilot reshaping.

The x-axis represents the performance gain over the default 802.11. Our result shows that LEAD outperforms 802.11 by 24% in terms of goodput. Moreover, the performance gain is significantly boosted with the support of link layer error recovery protocols. Specifically, when both 802.11 and LEAD are integrated with Maranello, PPR, and link layer FEC, the performance gains of LEAD over 802.11 are 1.72x, 1.43x, and 1.93x, respectively. This is because LEAD enables error recovery protocols to better exploit partially correct packets to improve network performance. Since LEAD is able to significantly reduce bit errors, the error recovery protocols need to retransmit fewer packets, leading to higher system throughput. In summary, the results in this section show that the bit-level error reduction due to LEAD corresponds to significant end-to-end performance improvement at the link layer.

7.4 Overhead

In the following, we evaluate the overhead of LEAD, including the protocol overhead, the cost of pilot misprediction, and the processing delay of pilot reshaping.

Protocol overhead. For each data packet, LEAD requires two bytes for inserting its own protocol header. A question we aim to answer is if it is better off to use these two bytes to carry redundant information for error correction. To justify the protocol overhead of LEAD, we compare its performance to an error correction scheme, which adopts the Reed-Solomon (RS) code at the link layer to correct errors. RS code breaks each packet into n-bit symbols, and then groups each r symbols to a codeword. Two redundant symbols of RS code can correct up to one erroneous symbol in a codeword. In our implementation, we adopt the setting used in [17], where n=8 and r=255. For each packet, we randomly select a codeword to add two redundant symbols, which is equivalent to the protocol overhead of LEAD.

Fig. 12 compares the percentage of error reduction of LEAD and RS code. We observe that, although RS code performs similarly with LEAD in terms of recovering corrupted packets, RS code cannot partially reduces bit errors when there are more flipped bits than its correcting capability. In comparison, LEAD is able to significantly reduce error bits, which can be transformed to a significant performance gain when error recovery protocols are available at the link layer, as discussed in Sec. 7.3.

Cost of pilot misprediction. During pilot-assisted decoding, LEAD eliminates the outputs that do not match the values of pilots. As a result, pilot misprediction may cause decoding errors. We evaluate this cost by studying the packet delivery performance of LEAD when replaying real-life traffic on a wireless channel of good quality (average SNR=20dB) using the most robust bit rate (*i.e.*, BPSK with rate-1/2 coding). Fig. 13 shows the percentage of packets where mispredictions are observed. The results are measured using 24 links randomly selected from SIGCOMM'08 trace. We observe that for 90% links, less than 5% packets have mispredictions. We observe that when link layer retransmission is enabled, the impacts of misprediction on packet loss rate and link goodput are negligible. Moreover, pilot mispredictions in channels of strong qualities can be mitigated by adopting a strict misprediction rate bound when deriving the history size of pilot extraction.

Pilot reshaping delay. As bit values observed in upper-layer protocols usually differ from those received at the physical layer, protocol signatures extracted from packet headers cannot be directly used in decoding. Therefore LEAD receiver reshapes pilots through encryption and scrambling before using them in pilot-assisted decoding, which incurs a processing delay.

To evaluate the processing delay, we implemented a software pilot reshaper for the WiFi Protected Access (WPA) protocol and the standard scrambler of 802.11. The reshaper follows the key mixing function defined in Temporal Key Integrity Protocol (TKIP) to combine the root key and the initialization vector into a 128-bit RC4 key, and takes the result to generate the keystream. Then the reshaper performs encryption and scrambling on 80 bytes, which is the upper bound on the length of bits used by LEAD to find bit bias. To measure the distribution of reshaping delay, we repeat this experiment 10000 times on a laptop equipped with a 2.9GHz CPU. Fig. 14 shows the distribution of reshaping delay. We observe that for more than 95% cases, the reshaping delay is below 7 *us*, which is shorter than the 10 *us* SIFS defined in 802.11g. As a result, the reshaping delay does not cause an 802.11 link to drop packets. We also note that such delay can be significantly reduced when the pilot reshaper is implemented in the firmware of WLAN NICs.

7.5 Pilot Availability and Other Decoders

Our measurements presented in Section 5 on the five real-life traffic datasets show that the ratio of pilot bits typically ranges from 10% to 30%. However, the percentage of pilot bits in a particular network also varies substantially depending on the size of packet payload. Moreover, the efficiency of pilot-assisted decoding also depends on the specific error correcting codes.

We now evaluate the effect of pilot ratio on the decoding performance. We demonstrate the efficiency of pilot-assisted decoding for three error correcting codes, including the BCJR decoder of convolutional code, the belief propagation decoder of LDPC, and the bubble decoder of Spinal code. We augment these standard decoders to exploit pilots, and measure the percentage of error reduction. The spinal code is configured using the parameters introduced in [19]. For LDPC and convolutional code, we use a coding rate of 1/2 with QPSK for modulation and demodulation. We also conducted the same experiments under other physical layer settings. Similar results are observed.

We deployed a link of two USRP nodes in an office environment. The transmitter sends back-to-back packets using different transmission power levels. The receiver logs the trace of demodulated symbols, and decodes them using standard and pilot-assisted decoders, respectively. For each packet, a group of bits are randomly selected as pilots[1], whose values are made known to the pilot-assisted decoder. We then vary pilot ratios to study the impact on decoding performance. For each error correcting code, the performance gain of pilot-assisted decoding is measured using 10000 packets that are erroneously decoded by the standard decoder.

The results are shown in Fig. 15. We observe that pilot-assisted decoding significantly reduces BERs for all codes. Specifically, when the pilot ratio is 20%, bit error is reduced by 71% and 74% for convolutional code and LDPC, respectively. Using Spinal code, the transmitter keeps sending coded symbols round by round, until the receiver successfully decodes the packet. We hence compare the BERs of the standard and pilot-assisted decoders in different rounds. Using 30% pilots, the BER is reduced by 44%, 80%, and 94% in the second, fifth and tenth round. The results demonstrate the efficiency of exploiting pilots to reduce decoding errors across different error correcting codes.

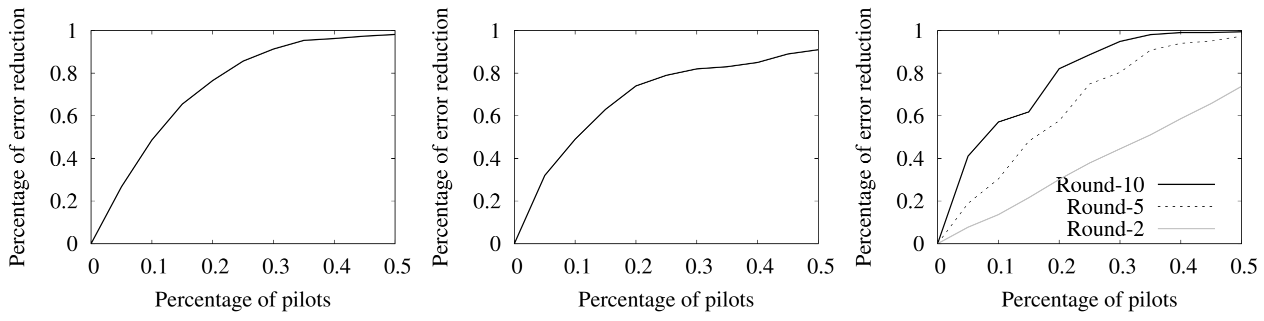
[1]The random pilot selection strategy is consistent with our implementation where the pilots are uniformly spread over the entire packet by the interleaver.

8. CONCLUSION AND FUTURE WORK

We present a new cross-layer wireless error correction approach called LEAD. The key idea of LEAD is to predict the values of some received bits based on the signatures of upper-layer protocols, and then use them as pilots to improve the PHY decoding performance. We first characterize the bit bias of protocol signatures in real 802.11 traffic, and develop an efficient algorithm to extract pilot bits whose values have high predictability. We then develop several mechanisms to integrate LEAD with 802.11 protocol stack, including interleaving protocol signatures across a whole packet so that the decoding of payload can leverage adjacent pilots, handling scrambled and encrypted pilots, and detecting and handling mispredicted pilots. We augment three standard decoders, including the BCJR decoder of convolutional code, the belief propagation decoder of LDPC code, and the bubble decoder of the Spinal code to effectively exploit pilots. We implement LEAD on USRP platform and evaluate its performance by replaying real-life traffic traces on a testbed of 12 USRP links. Our results show that LEAD can reduce BER by more than 90% for half of the error packets while incurring very low overhead. Moreover, LEAD improves the end-to-end link throughput by 1.43x to 1.93x over the existing error correction schemes.

The performance of LEAD depends on the availability of "good" pilots whose values can be correctly predicted by the receiver. The pilot extraction algorithm of LEAD maximizes the number of pilots while limiting the misprediction threshold under a given threshold. When the wireless channel condition is good, it is desirable to set a lower misprediction rate to minimize the additional bit errors caused by pilot misprediction. In contrast, when the channel condition is bad, the threshold should be set higher to produce more pilots. In the future, we will develop an adaptive scheme which adjusts the bound dynamically according to the measurement of channel quality.

LEAD has important implications for the efficiency of several types of wireless protocols. The pilots extracted by LEAD can be utilized to estimate accurate channel BER at the receiver without resorting to probing packets. This can significant reduce the overhead of rate adaptation. Several cross-layer protocols such as PPR [15] utilize PHY information (e.g., decoding confidence) to estimate packet errors. LEAD can improve the decoding performance and hence helps error esti-



(a) Convolutional code. (b) LDPC. (c) Rateless spinal code.

Figure 15: Ratios of reduced error bits when using pilot-assisted decoding for convolutional code, LDPC, and rateless Spinal code.

mation accuracy. We will integrate LEAD with these protocols and study the benefits of pilot-assisted decoding on partial packet recovery and rate adaption.

9. REFERENCES

[1] RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed. RFC 3095, July 2001.

[2] The RObust Header Compression (ROHC) Framework. RFC 4995, July 2007.

[3] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *ACM SIGCOMM*, 2004.

[4] 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.

[5] A. Bhartia, Y.-C. Chen, S. Rallapalli, and L. Qiu. Harnessing frequency diversity in wi-fi networks. In *ACM MobiCom*, 2011.

[6] R. Chandra, R. Mahajan, V. Padmanabhan, and M. Zhang. CRAWDAD data set microsoft/osdi2006 (v. 2007-05-23).

[7] B. Chen, Z. Zhou, Y. Zhao, and H. Yu. Efficient error estimating coding: feasibility and applications. In *ACM SIGCOMM*, 2010.

[8] P. Duhamel and M. Kieffer. *Joint Source-Channel Decoding: A Cross-Layer Perspective with Applications in Video Broadcasting*. Academic Press, 2010.

[9] R. G. Gallager. Low-density parity-check codes, 1963.

[10] D. Halperin, W. Hu, A. Sheth, and D. Wetherall. Predictable 802.11 packet delivery from wireless channel measurements. In *ACM SIGCOMM*, 2010.

[11] D. Halperin, W. Hu, A. Sheth, and D. Wetherall. Tool release: Gathering 802.11n traces with channel state information. *ACM SIGCOMM CCR*, 2011.

[12] 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.

[13] J. Huang, Y. Wang, and G. Xing. Lead: leveraging protocol signatures for improving 802.11 decoding performance.

[14] S. Jakubczak and D. Katabi. A cross-layer design for scalable mobile video. In *ACM Mobicom*, 2011.

[15] K. Jamieson and H. Balakrishnan. Ppr: Partial packet recovery for wireless networks. In *ACM SIGCOMM*, 2007.

[16] W. E. Leland, W. Willinger, M. S. Taqqu, and D. V. Wilson. On the selfsimilar nature of ethernet traffic. In *ACM SIGCOMM*, 1993.

[17] K. C.-J. Lin, N. Kushman, and D. Katabi. Ziptx: Harnessing partial packets in 802.11 networks. In *ACM MobiCom*, 2008.

[18] X. L. Liu, W. Hu, Q. Pu, F. Wu, and Y. Zhang. Parcast: soft video delivery in mimo-ofdm wlans. In *ACM Mobicom*, 2012.

[19] J. Perry, P. Iannucci, K. E. Fleming, H. Balakrishnan, and D. Shah. Spinal Codes. In *ACM SIGCOMM*, 2012.

[20] C. Phillips and S. Singh. CRAWDAD data set pdx/vwave (v. 2009-07-04).

[21] A. Schulman, D. Levin, and N. Spring. CRAWDAD data set umd/sigcomm2008 (v. 2009-03-02).

[22] S. Sen, S. Gilani, S. Srinath, S. Schmitt, and S. Banerjee. Design and implementation of an ”approximate” communication system for wireless media applications. *ACM SIGCOMM*, 2010.

[23] S. Sen, R. Roy Choudhury, and S. Nelakuditi. Csma/cn: carrier sense multiple access with collision notification. In *ACM MobiCom*, 2010.

[24] S. Sen, R. Roy Choudhury, and S. Nelakuditi. No time to countdown: migrating backoff to the frequency domain. In *ACM MobiCom*, 2011.

[25] S. Sen, N. Santhapuri, R. R. Choudhury, and S. Nelakuditi. Accurate: constellation based rate estimation in wireless networks. In *NSDI*, 2010.

[26] K. Srinivasan, M. A. Kazandjieva, S. Agarwal, and P. Levis. The &#946;-factor: measuring wireless link burstiness. In *ACM SenSys*, 2008.

[27] 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.

[28] M. Vutukuru, H. Balakrishnan, and K. Jamieson. Cross-layer bit rate adaptation. In *ACM SIGCOMM*, 2009.

[29] G. R. Woo, P. Kheradpour, S. Dawei, and D. Katabi. Beyond the bits: cooperative packet recovery using physical layer information. In *ACM SIGCOMM*, 2007.

[30] 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.