LEAD: Leveraging Protocol Signatures for Improving Wireless Link Performance

Jun Huang, Yu Wang and Guoliang Xing
Department of Computer Science and Engineering, Michigan State University, USA
{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 channel decoders. 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 channel decoding. The approach of LEAD is motivated by two key insights. First, channel codes can correct more 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 channel decoding. To realize this idea, we first characterize bit bias in real-life network traffic, and develop an efficient algorithm to extract pilot bits with assured prediction accuracy. We then propose a decoding framework to allow existing channel decoders to effectively exploit extracted pilots. We implement LEAD on GNU Radio/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% bit errors for 48.9% packets, and improves the end-to-end link throughput by 1.43x to 1.93x over existing error correction schemes.

Categories and Subject Descriptors
C.2.1 [Computer Communication Networks]: Network Architecture and Design—wireless communication

Keywords
Cross-layer design, bit predictability, channel code, decoder

1. INTRODUCTION
The last decade has 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 bit error rate due to the effects of signal attenuation, multipath fading, and interference. To improve the resilience to transmission errors, wireless communication systems employ channel coding at the physical layer (PHY). However, when the number of errors exceeds the correcting capability of channel code, the original data cannot be fully recovered through channel decoding, leading to corrupted packets. 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 bit error rate (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 [17]. 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 channel 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 channel 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 LEAD, a novel cross-layer design to reduce bit errors in 802.11-based WLANs. While traditional wisdom of cross-layer design is to exploit physical layer information at the MAC layer or above, LEAD represents a paradigm shift in that it leverages upper-layer protocol signatures to improve physical layer channel decoding. LEAD offers several key advantages. First, the design of LEAD is independent of the underlying channel code. Thus the pilot-assisted decoding mechanism of LEAD can be exploited by all existing channel 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 [16], link layer forward error correction [18], and block retransmission-based recovery [13]) 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 channel 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 encryption 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 GNU Radio/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 of 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

Reducing decoding errors. Several mechanisms have been proposed to improve the performance of channel decoding for 802.11-based WLANs. For example, SoftCast [15], ParCast [19], 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 the performance of channel decoding. This allows LEAD to work efficiently with diverse upper-layer protocols and applications.

Modern network standards, such as 802.11 and LTE, use pilot symbols explicitly inserted into the physical layer to improve the performance of demodulation. For example, current 802.11 OFDM PHY deploys four pilot subcarriers in each OFDM symbol to help the receiver track channel conditions. In comparison, LEAD uses pilots to help channel decoding, thus is complimentary to pilot-assisted demodulation. Moreover, LEAD extracts decoding pilots from protocol headers, thus avoids the extra bandwidth consumption for embedding pilot symbols.

Partial packet recovery. Partially corrupted packets can be exploited to improve the performance of wireless networks. Selective retransmission based protocols [16] [13] reduce packet recovery overhead by only retransmitting the bits that are likely to be wrong. Similarly, Zhang et al. [34] propose μACK, which sends ACKs for every few symbols on a separate feedback channel to enable in-frame recovery. Other protocols, such as ZipTx [18], use link layer FEC to correct bit errors. Unite [33] combines selective retransmission and link layer FECs to minimize error recovery overhead. Leveraging the broadcast nature of wireless channel, cooperative protocols [10] [20] [21] [32] coordinate multiple receivers that hear the same transmission to correct packet errors. Compared with these work, LEAD operates at the physical layer to improve the performance of channel decoding. Therefore, LEAD is orthogonal to existing partial packet recovery schemes, and can be integrated with them to achieve better performance.

Cross-layer design. Cross-layer design has been extensively studied in existing literature. Most prior work focuses on leveraging physical layer information at the upper-layer, e.g., using decoding confidence to reduce link layer retransmission overhead [16], or exploiting the error structure at the PHY to provide unequal protection to upper-layer flows [25] [5]. Similarly, PHY-aware rate adaptation algorithms, such as EffSNR [11], AccuRate [28] and Soft-Rate [31], exploit different physical layer hints to facilitate bit rate adaptation. Compared with these work, LEAD features a different design which leverages upper-layer protocol signatures in channel decoding. LEAD can be integrated with previous cross-layer design to further improve wireless link performance.

Exploiting bit predictability. Prior work [1] [2] exploits bit predictability for compressing IP/TCP and IP/UDP headers. However, existing header compression protocols are exclusively tied with specific protocols, thus cannot be applied
Table 1: The transition and output table of a rate-1/2 convolutional 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, \(m_1 m_2\) represents the encoder state.

<table>
<thead>
<tr>
<th>State ((m_1 m_2))</th>
<th>Transition (m_0 = 0)</th>
<th>Transition (m_0 = 1)</th>
<th>Output (m_0 = 0)</th>
<th>Output (m_0 = 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00 10</td>
<td>00 11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>00 10</td>
<td></td>
<td>11 00</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>01 11</td>
<td>10 01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>01 11</td>
<td>01 10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 0110000010, respectively. The table shows the expected codewords, decoded bits and accumulated costs of the correct path (PATH-1) and a faulty path (PATH-2).

<table>
<thead>
<tr>
<th>Tx code</th>
<th>Rx code</th>
<th>PATH-1 Code</th>
<th>PATH-1 Bit</th>
<th>PATH-1 Cost</th>
<th>PATH-2 Code</th>
<th>PATH-2 Bit</th>
<th>PATH-2 Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>00 0 3</td>
<td>11 1 1</td>
<td>1</td>
<td>01 3</td>
<td>01 0 3</td>
<td>3</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>00 0 3</td>
<td>01 1 2</td>
<td>2</td>
<td>00 3</td>
<td>00 1 3</td>
<td>3</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00 0 3</td>
<td>01 0 3</td>
<td>3</td>
<td>00 4</td>
<td>10 0 3</td>
<td>3</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>00 0 4</td>
<td>10 0 3</td>
<td>3</td>
<td>11 1 1</td>
<td>11 0 1</td>
<td>1</td>
</tr>
</tbody>
</table>

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 convolutional decoder will prune PATH-1 due to its higher cost, resulting in flapped 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 channel codes. In Section 7, we will study the gains of pilot-assisted decoding for LDPC [9] and Spinal [22], 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 [17] 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 [29] [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 the performance of channel decoding. 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 $T_{m,n}$ to represent a window of its packets,

$$
T_{n,m} = \begin{bmatrix}
    b_{1,1} & \cdots & b_{1,n} \\
    \vdots & \ddots & \vdots \\
    b_{m,1} & \cdots & b_{m,n}
\end{bmatrix}
$$

where $b_{i,j} = 1$ or 0 gives the value of the $i$th bit in the $j$th packet, and $n$ defines the maximum packet size. If the size of $j$th 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,

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

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

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 [24] and OSDI'06 [6] were collected in WLANs of hundreds of users attending the two conferences. PDX [23] 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 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.

![Figure 2: Bit bias measured in a window of 100 packets.](image1)

![Figure 3: Quantities and percentages of fixed bits. Min, Max, 25th, and 75th percentiles are shown.](image2)
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 [17]. 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 [29] [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 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 optimize pilot extraction performance.

Given a sequence of $n$ packets, for the $t$th bit, we use $N_i(k)$ to denote the times that it is selected as pilot using a history size $k$. We have $N_i(k_0) \geq N_i(k_1)$, if $k_0 \leq k_1$. This is because $h_i(t, k_0) \subset h_i(t, k_1)$, where $h_i(t, k)$ is the set of last $k$ observations on bit $i$ when packet $t$ is received. Thus if bit $i$ is selected using $h_i(t, k_1)$, it will also be selected using $h_i(t, k_0)$. Let $E_i(k)$ to be the pilot misprediction rate. Our goal is to tune $k_i$ for each bit $i$, such that

$$k_i = \arg\min \{E_i(t) \leq \Delta\}$$  \hspace{1cm} (3)

where $\Delta$ is the pre-defined threshold on pilot misprediction rate. To reduce decoding errors, $\Delta$ should be lower than the BER of conventional decoder. Recent study [7] shows that the BER observed on commodity 802.11 radios typically

---

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

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

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

\[
\mu_i = D_{KL}(P_i(t) || Q_i(t)) = \sum_t P_i(t) \ln \frac{P_i(t)}{Q_i(t)}
\]

where \( P_i(t) \) 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 \( \mu_i \geq 2 \).

Optimizing 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 \( \beta \) is the bit bias. To satisfy Eq. (3), \( \beta \) 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 \( \Pr(t+1|t) = 1 - P(t+1) \), which is the conditional probability that the burst length is longer than \( t + 1 \), given that the length of current burst is \( t \). \( P(t) \) is the burst length CDF. As a result, the minimum history size can be derived as,

\[
k = \arg\min_t \left\{ \frac{1 - P(t+1)}{1 - P(t)} > 1 - \Delta \right\}
\]

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 \( P_i(t) \) 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,

\[
F(t) = \begin{cases} 1 - \frac{1}{\alpha t}, & t > 1 \\ 0, & \text{otherwise} \end{cases}
\]

where \( \alpha \) 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,

\[
\hat{\alpha} = \frac{n}{\sum_{i=1}^{n} \ln x_i}
\]

Therefore, we can calculate the minimum history size for a bursty biased bit as,

\[
k = \frac{1}{1 - \exp \left( \frac{\log (1 - \Delta)}{\hat{\alpha}} \right)} - 1
\]

Putting them together. The extractor maintains a moving window of \( N \) correctly received packets, and updates the burst length CDF \( P(t) \) and the burstiness metric \( \mu \) for each bit every \( T \) seconds. We observed that LEAD performance is not sensitive to the update period \( T \), i.e., the measured burst length distribution is accurate enough to assure low misprediction rate even using long update intervals. This is because the burst patterns of header bits mainly depend on the bit semantics defined by upper-layer protocols, which do not vary over time. Motivated by this observation, we conduct offline measurements using real-life packet traces to find a long update interval, which minimizes the update overhead while assuring that the misprediction rates of extracted pilots are below the given threshold. Our empirical study shows that \( T = 10s \) and \( N = 600 \) work efficiently in practice.

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 channel decoders to effectively exploit the extracted pilots. In the following, we first present the design of LEAD framework, and then describe 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. Since pilots are extracted from protocol headers, LEAD needs an interleaver to spread pilots over the whole packet, such that the decoding of payload bits can be benefited from adjacent pilots. Unfortunately, the default 802.11 interleaver cannot serve this purpose because of its limited interleaving depth. Specifically, it only permutes coded bits within each OFDM symbol to mitigate the bursty errors caused by frequency selective fading. In 802.11g, the maximum number of bits carried by an OFDM symbol is only 216 bits, while a packet can be as large as thousands of bits. As a result, using the default 802.11 interleaver, pilots extracted from protocol headers cannot be spread over the entire packet. To address this problem, LEAD employs an additional MAC layer interleaver to uniformly spread pilots to packet payload. At the receiver, the reverse operation is performed to recover bit orders after pilot-assisted decoding.

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 descrambling. 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 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 calculates 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 them 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.

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 the underlying channel code. Thus LEAD can be exploited by all existing channel 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 [22], 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 [14].

**Table 4: Pilot-assisted decoders of LEAD.**

<table>
<thead>
<tr>
<th>Channel code</th>
<th>Decoder</th>
<th>Number of lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolutional</td>
<td>BCJR [4]</td>
<td>143 147</td>
</tr>
<tr>
<td>LDPC</td>
<td>BP [9]</td>
<td>87 97</td>
</tr>
<tr>
<td>Spinal code</td>
<td>Bubble [22]</td>
<td>158 181</td>
</tr>
</tbody>
</table>

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.

\[
\alpha_t(s) = \Pr(s | o_{1:t}) = \sum_{s'} \alpha_{t-1}(s') \gamma_t(s, s') \tag{10}
\]

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

\[
\beta_t(s) = \Pr(o_{t:k} | s) = \sum_{s'} \beta_{t+1}(s') \gamma_t(s, s') \tag{11}
\]

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

\[
\lambda_t(s) \propto \Pr(s | o_{1:t}) \times \Pr(o_{t+1:k} | s) \tag{12}
\]

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

\[
a_0(0) = \beta_k(0) = 1 \\
\alpha_0(s) = \beta_k(s) = 0 \quad \text{for all } s \neq 0 \tag{13}
\]

Let \(s^0\) and \(s^1\) 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

\[
\text{LLR}(y_t) = \log \frac{\sum_{s \in s^0} \lambda_t(s)}{\sum_{s \in s^1} \lambda_t(s)} \tag{14}
\]

Pilot-assisted BCJR augments the standard BCJR by improving the estimation of \(\lambda_t(s)\) in Eq. (12), where the \(\gamma_t(s, s')\) in Eq. (10) and Eq. (11) equals to 0 if the bit that corresponds to the transition \(s \rightarrow s'\) 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 atop the OFDM implementation of GNURadio/USRP platform. LEAD takes demodulated symbols as input, converts complex symbols on constellation to soft coded bits using a soft-output demapper [30], and then performs pilot-assisted decoding through the procedures illustrated in Fig. 6.

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

Figure 8: CDF of the number of extracted pilots.

maximize the number of extracted pilots while limiting the misprediction rate under a pre-defined 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 [24]. The results observed on other traces are similar.

Fig. 7 shows the average misprediction rates for the bits located in the first 80 bytes 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. The history size 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 are updated every 10 seconds. To implement Oracle, we first run the extraction algorithm using all history sizes in each interval of 10 seconds, and then look back to choose the optimal 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 a convolutional code, which is the default channel 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. Our evaluation is conducted on a testbed of 12 USRP links deployed in an office building, on a 4MHz channel centered at 5 GHz. We observe that channel conditions differ substantially in our testbed, where the average signal-to-noise ratios (SNR) measured on USRP links range from 7 dB to 20 dB. To improve the realism of our evaluation, we replay the traffic trace collected in SIGCOMM'08 [24] 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 rate. 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 to compare their BERs.

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

**Effect of packet size.** Fig. 10 evaluates the effect of packet size on LEAD performance. We observe that LEAD achieves significant performance gain for packets that are shorter than 256 bytes. Specifically, when the packet size is shorter than 128 bytes, LEAD mitigates 89% bit errors. When the packet size is between 128 and 256 bytes, LEAD reduces the bit error rate from 0.077 to 0.027. This is because the ratio of pilots is higher for short packets. We observed similar results unless stated otherwise, we will use LEAD consistently outperforms BCJR for all protocols. LEAD is especially efficient for those protocols that extensively use short packets. For example, for IGMP, LEAD reduces 97.3% bit errors compared to the standard BCJR decoder.

### 7.3 Impact on Link Layer Performance

In the following, we 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 performance gain of LEAD. To collect fine-grained channel trace, we employ the tool introduced in [12] 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. 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 [11], where the transmitter tunes its bit rate based on the effective SNR measured using receiver ACKs. We implemented the binary exponential backoff at the MAC layer, 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 [13], link layer forward error correction (FEC) [18], and partial packet retransmission (PPR) [16].

- **Maranello.** Maranello [13] is a block retransmission based partial packet recovery protocol. Specifically, a partially corrupted packet is divided into blocks. For each block, a 32-bit checksum is computed and returned to the sender for identifying block(s) that contain bit errors. To reduce recovery overhead, Maranello only retransmits corrupted block(s). In our implementation, we use a block size of 64 bytes, which is compliant to the setting introduced in [13].

- **Link layer FEC.** To evaluate the impact of LEAD on link layer error correcting schemes, we implemented the protocol introduced in [18]. To correct bit errors, Reed-Solomon (RS) 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 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.
Figure 12: Link layer goodput gains of LEAD over 802.11 when different error recovery protocols are used at the link layer.

- PPR. PPR [16] exploits 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.

**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 integrate LEAD with the above error recovery protocols, and evaluate their effects on link layer goodput. The results are shown in Fig. 12. The goodputs shown in the figure is normalized to 802.11 goodput. Our result shows that the performance gain of LEAD 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 optimal 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 corrupted packets to improve network performance. Since LEAD is able to significantly reduce bit errors, the error recovery protocols need to retransmit fewer bits, leading to higher system throughput. The results demonstrate that the bit-level error reduction achieved by LEAD results in a significant end-to-end performance gain 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 bit errors. Specifically, when both 802.11 and LEAD are integrated with Maranello, PPR, and optimal 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 corrupted packets to improve network performance. Since LEAD is able to significantly reduce bit errors, the error recovery protocols need to retransmit fewer bits, leading to higher system throughput. The results demonstrate that the bit-level error reduction achieved by LEAD results in a significant end-to-end performance gain 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 bit errors. Specifically, when both 802.11 and LEAD are integrated with Maranello, PPR, and optimal 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 corrupted packets to improve network performance. Since LEAD is able to significantly reduce bit errors, the error recovery protocols need to retransmit fewer bits, leading to higher system throughput. The results demonstrate that the bit-level error reduction achieved by LEAD results in a significant end-to-end performance gain 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 integrate LEAD with the above error recovery protocols, and evaluate their effects on link layer goodput. The results are shown in Fig. 12. The goodputs shown in the figure is normalized to 802.11 goodput. Our result shows that the performance gain of LEAD 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 optimal 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 corrupted packets to improve network performance. Since LEAD is able to significantly reduce bit errors, the error recovery protocols need to retransmit fewer bits, leading to higher system throughput. The results demonstrate that the bit-level error reduction achieved by LEAD results in a significant end-to-end performance gain 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 bit errors. Specifically, when both 802.11 and LEAD are integrated with Maranello, PPR, and optimal 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 corrupted packets to improve network performance. Since LEAD is able to significantly reduce bit errors, the error recovery protocols need to retransmit fewer bits, leading to higher system throughput. The results demonstrate that the bit-level error reduction achieved by LEAD results in a significant end-to-end performance gain 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 bit errors. Specifically, when both 802.11 and LEAD are integrated with Maranello, PPR, and optimal 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 corrupted packets to improve network performance. Since LEAD is able to significantly reduce bit errors, the error recovery protocols need to retransmit fewer bits, leading to higher system throughput. The results demonstrate that the bit-level error reduction achieved by LEAD results in a significant end-to-end performance gain 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 bit errors. Specifically, when both 802.11 and LEAD are integrated with Maranello, PPR, and optimal 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 corrupted packets to improve network performance. Since LEAD is able to significantly reduce bit errors, the error recovery protocols need to retransmit fewer bits, leading to higher system throughput. The results demonstrate that the bit-level error reduction achieved by LEAD results in a significant end-to-end performance gain 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 bit errors. Specifically, when both 802.11 and LEAD are integrated with Maranello, PPR, and optimal 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 corrupted packets to improve network performance. Since LEAD is able to significantly reduce bit errors, the error recovery protocols need to retransmit fewer bits, leading to higher system throughput. The results demonstrate that the bit-level error reduction achieved by LEAD results in a significant end-to-end performance gain 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 bit errors. Specifically, when both 802.11 and LEAD are integrated with Maranello, PPR, and optimal 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 corrupted packets to improve network performance. Since LEAD is able to significantly reduce bit errors, the error recovery protocols need to retransmit fewer bits, leading to higher system throughput. The results demonstrate that the bit-level error reduction achieved by LEAD results in a significant end-to-end performance gain at the link layer.
each packet, a group of bits are randomly selected as pilots.

The trace of demodulated symbols, and decodes them using different transmission power levels. The receiver logs similar results are observed.

for modulation and demodulation. We also conducted the convolutional code, we use a coding rate of 1/2 with QPSK used using the parameters introduced in [22]. For LDPC and the bubble decoder of Spinal code. We augmented these standard decoders to exploit pilots, and measure the percentage of error reduction. The Spinal code is configured using the parameters introduced in [22]. 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, whose values are made known to the pilot-assisted decoder. We then vary pilot ratios to study the impact on decoding performance. For each channel 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. 16. 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 channel codes.

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 augmented three standard decoders, including the BCJR decoder of convolutional code, the belief propagation decoder of LDPC code, and the bubble decoder of 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 significantly reduce the overhead of rate adaptation. Several cross-layer protocols such as PPR [16] utilize PHY information (e.g., decoding confidence) to estimate packet errors. LEAD can improve the decoding performance and hence helps error estimation accuracy. We will integrate LEAD with these protocols and study the benefits of pilot-assisted decoding on partial packet recovery and rate adaption.

Figure 15: Delay caused by software pilot reshaping.

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

This work was supported in part by U.S. National Science Foundation under grants CNS-0916576 and CNS-0954039 (CAREER). We thank the anonymous reviewers and the shepherd Dr. Suman Banerjee for providing valuable feedbacks for this work.

REFERENCES