Towards Accurate Corruption Estimation in ZigBee Under Cross-Technology Interference

Gonglong Chen[1], Wei Dong[1*], Zhiwei Zhao[2], and Tao Gu[3]
[1]College of Computer Science, Zhejiang University.
[2]Computer Science, University of Electronic Science and Technology of China.
[3]Computer Science and Information Technology, RMIT University.
Email: {*chengl, dongw*}*@emnets.org*, *zzw@uestc.edu.cn*, *tao.gu@rmit.edu.au*

***Abstract*—Cross-Technology Interference affects the operation of low-power ZigBee networks, especially under severe WiFi interference. Accurate corruption estimation is very important to improve the resilience of ZigBee transmissions. However, there are many limitations in existing approaches such as low accuracy, high overhead, and requiring hardware modification. In this paper, we propose an accurate corruption estimation approach, AccuEst, which utilizes per-byte SINR (Signal-to-Interference-and-Noise Ratio) to detect corruption. We combine the use of pilot symbols with per-byte SINR to improve corruption detection accuracy, especially in highly noisy environments (i.e., noise and interference are at the same level). In addition, we design an adaptive pilot instrumentation scheme to strike a good balance between accuracy and overhead. We implement AccuEst on the TinyOS 2.1.1/TelosB platform and evaluate its performance through extensive experiments. Results show that AccuEst improves corruption detection accuracy by 78.6% on average compared with state-of-the-art approach (i.e., CARE) in highly noisy environments. In addition, AccuEst reduces pilot overhead by 53.7% on average compared to the traditional pilot-based approach. We implement AccuEst in a coding-based transmission protocol, and results show that with AccuEst, the packet delivery ratio is improved by 20.3% on average.**

I. Introduction

The recent years have witnessed the unprecedented proliferation of smart wireless devices. A number of radio technologies exist such as WiFi, Bluetooth and ZigBee for applications with different requirements in throughput, timeliness and energy-efficiency. Since these radio technologies operate on the same 2.4GHz ISM band, it inevitably causes Cross-Technology Interference (CTI). In particular, low-power ZigBee is more sensitive to CTI since it has to combat with severe interference from ubiquitous WiFi Access Point (AP) deployment [1].

Prior studies have shown that ZigBee packets may suffer from severe corruption with WiFi interference [1, 2]. Many solutions have been proposed to improve the resilience of ZigBee transmission by first estimating corruptions and then recovering corrupted packets. On successfully identifying corruptions in a packet received, the receiver may be able to recover partial packet by requesting the retransmission of corruptions [3], enable whole packet recovery by combining the correct parts of multiple partial packets [4, 5], or recover more potential corrupted packets with the estimated corruptions (e.g., Reed Solomon code) when a packet is encoded with forward error correction (FEC) [6, 7]. All these methods rely heavily on the accuracy of corruption estimation.

Several corruption estimation methods exist in the literature [3, 8–12]. The PHY-based approach (e.g., PPR [3], AccuRate [8]) relies on the *detailed* PHY-layer information, e.g., Hamming distance between chip sequences or dispersion in the constellation space. Such an approach, albeit accurate, cannot work on COTS ZigBee devices since the detailed PHY-layer information is simply inaccessible. The pilot-based approach (e.g., ZipTx [9], LEAD [10]) relies on the *known* pilot symbols for coarse-grained BER (Bit Error Rate) estimation. Such an approach suffers from an inherent tradeoff between accuracy and overhead: if we instrument a small number of pilots, the accuracy will be low; otherwise, the packet overhead will be high. In general, this approach, when used alone, is *insufficient* to identify corruption in a packet. All the existing methods rely on hardware modification or incur large packet overhead.

Recently, in-packet RSSI sampling has accelerated much research interest [13–19]. In-packet RSSI sampling, working at the maximum sampling rate, provides fine-grained per-byte RSSI values for a packet. A key benefit of this technique is that it can be directly supported by COTS ZigBee devices without hardware modification. The fine-grained RSSI time series, provided by in-packet RSSI sampling, have been used in several works to classify interference or corruption estimation in the presence of WiFi interference. The *in-packet RSSI-based approach* such as REPE [13] and CARE [14] can work on COTS ZigBee devices with no packet overhead. However, they still suffer from *low accuracy*, especially in a common industry environment [20, 21] with high noise (i.e., noise and interference are at the same level as we analyzed in Section III). Achieving accurate corruption estimation in the in-packet RSSI-based approach is challenging. A straightforward idea is to sample noise level and explicitly mitigate the impact of noise power. However, WiFi interference is time-varying, hence it is not enough to distinguish noise from interference by sampling noise power. Our experimental study shows that existing in-packet RSSI-based approaches do not perform well in a highly noisy environment.

To address the above challenge, in this paper we propose a novel approach named AccuEst, to Accurately Estimate corruptions of ZigBee packets with WiFi interference in a highly noisy environment. We discover an interesting noise-resistance characteristic of link-layer pilot symbols which are

*This work was supported in part by the National Science Foundation of China under Grant No. 61472360, National Key Basic Research Program of China under 973 Grant No. 2015CB352400, CCF-Tencent Open Research Fund, Zhejiang Provincial Key Research and Development Program No. 2017C02044, and the Fundamental Research Funds for the Central Universities No. 2017FZA5013. Wei Dong is the corresponding author.

known to both senders and receivers. This information can be used to indicate whether symbol in a received packet is corrupted or not, and hence it can guide us to train a model to detect corruption. The PHY-layer information (i.e., per-packet RSSI, per-packet LQI, and per-byte RSSI values) offered by COTS ZigBee devices can be regarded as the input feature of the model. In particular, the per-byte RSSI values can capture the impact of interference (e.g., burstiness, fluctuations) in a *fine-grained* manner. However, a direct combination of PHY-layer information and link-layer information may not work effectively. The reason is that the PHY-layer information only reflects the absolute value of interference strength, and it is not enough to be used directly for corruption detection. We thus propose a new PHY-layer feature, per-byte SINR (Signal to Interference and Noise Ratio) deriving from per-byte RSSI values, to better indicate byte-level corruptions. In this way, we can achieve better corruption estimation by combining *cross-layer* information.

While combining cross-layer information looks promising, network throughput may degrade since the link-layer information (i.e., pilot symbols) increases packet redundancy. Directly reducing the amount of the link-layer information, however, may decrease corruption estimation accuracy. It is thus challenging to ensure high corruption detection accuracy while minimizing overhead. To further investigate this issue, we first conduct experimental studies as shown in Section IV-C. We observe from our experimental results that when the interference pattern (i.e., burstiness, fluctuation) is not obvious, the low interference power is highly possible to be recorded as noise power, and this is essential to the calculation of per-byte SINR. As a consequence, accuracy is decreased due to the interfered per-byte SINR and more pilot symbols are needed to quickly update the model to compensate for the interfered PHY-layer feature. On the other hand, when the interference pattern is obvious, few pilot symbols are needed since the PHY-layer feature can be inferred accurately. Motivated by the above observations, we design an *interference pattern-aware* approach to strike a good balance between accuracy and overhead.

We implement AccuEst on TinyOS 2.1.1 with TelosB nodes and evaluate its performance in different environments. Our results show that AccuEst consistently achieves better performance than the state-of-the-art approach, i.e., CARE. Specifically, the improvement of corruption detection accuracy is obvious when interference level and noise level are close (i.e., 78.6% in average). AccuEst reduces pilot overhead by 53.7% on average compared to the traditional pilot-based approach while achieving the equivalent relative error of SER (Symbol Error Rate) estimation. To demonstrate the effectiveness of AccuEst in real scenarios, we implement AccuEst in a coding-based transmission protocol. The testbed results show that with AccuEst, the packet delivery ratio is improved by 20.3% on average.

The contributions of this paper are summarized as follows.

- We theoretically analyze and identify the limitations of existing in-packet RSSI-based corruption estimation approaches in highly noisy environments (i.e., noise and interference are at the same level).
- We propose a novel corruption estimation approach, which exploits cross-layer information and a learning-based model to combine different information sources to achieve high accurate corruption prediction.
- We design an interference pattern-aware approach to minimize overhead while ensuring high corruption detection accuracy.
- We implement AccuEst on the TelosB platform with TinyOS 2.1.1 and evaluate its performance extensively. The results show that AccuEst significantly improves the corruption detection accuracy compared with the state-of-the-art approach in in highly noisy environments.

The rest of this paper is organized as follows. Section II introduces the related work. Section III presents the limitation of prior work. Section IV shows the design of AccuEst. Section V presents the evaluation results. Section VI discusses the generality of AccuEst, and finally, Section VII concludes this paper and discusses future research directions.

II. Related Work

In this section, we discuss the related work. We first introduce existing corruption estimation (i.e., PHY-based approach which requires hardware modification, and pilot-based approach which requires known pilot symbols). We then present the in-packet RSSI sampling technique and discuss how the per-byte RSSI time series can benefit the upper-layer protocol design.

A. PHY-based Approach

The detailed PHY-layer information provides accurate hints and it has been utilized by much prior works to identify erroneous bits [3], estimate BER [8, 12] or classify interference [5, 22].

PPR [3] implements an expanded PHY-layer interface called SoftPHY that provides a detailed PHY-layer hint about the PHY's confidence in each bit it decodes. Essentially, this confidence is derived from the Hamming distance between the actual received chip sequence and decoded chip sequence corresponding to a valid symbol. The higher layer can then use these hints to identify incorrect bits and request the retransmission of only the selected portions of a packet where there are bits likely to go wrong. AccuRate [8] exploits the dispersion of the symbol constellation space to compute the optimal bit rate. Smaller dispersion means better link quality which is capable of supporting higher bit rates. By comparing these dispersions to the permissible dispersions at different bit rates, AccuRate derives the maximum rate to be used for packet transmission.

Other than the above work which directly estimates corruptions, there is also work which accurately detects interference source. DoF [22] utilizes Fast Fourier Transform (FFT) to extract the repeating hidden patterns of different wireless protocols and then classifies interference sources according to the extracted patterns. CrossZig [5] exploits the variations in demodulated results of signals (i.e., soft values) for interference type detection (i.e., Intra- and Cross-Technology Interference) and then enables an appropriate mechanism to recover the corrupted packet.

Different from the above approaches which require modification in hardware, our approach works directly on COTS ZigBee devices with no hardware modification.

B. Pilot-based Approach

Pilot symbols/bits are the symbols/bits which are known before decoding and they can be used to estimate BER [9, 11,

23] (Bit Error Rate) or assist channel decoding [10].

LEAD [10] is a cross-layer solution that improves the performance of existing channel decoders. It extracts the fixed or highly biased header fields as the pilot bits and spreads the pilot bits over the whole packet to guide packet decoding. SmartPilot [11] further extracts more pilots from both detailed PHY-layer information and upper layer protocol headers to estimate BER and then picks a good data rate. However, they both require hardware modification, limiting their use on existing ZigBee devices.

ZipTx [9] is a software-only approach for recovering partial packets. It evenly instruments the known pilot bits into a packet transmitted from sender. The receiver then uses these known pilot bits to estimate the overall BER of the packet. High BER packets will be requested for retransmission while low BER packets will be decoded by error correcting codes.

The pilot-based approach does not require hardware modification. However, they only provide coarse-grained information which is insufficient to localize corruptions precisely. In addition, they incur relatively large packet overhead due to pilot symbols. To address this issue, we design an interference pattern-aware pilot instrumentation method to strike a good balance between accuracy and overhead.

C. In-packet RSSI Approach

In-packet RSSI sampling has been recently proposed to provide fine-grained per-byte RSSI values for a packet, and it has accelerated a lot of interesting work [13–19, 24].

SoNIC [18] utilizes the key insight that different interferers disrupt individual 802.15.4 packets in different ways that can be detected by sensor nodes. Then the distinct patterns, e.g., variances of in-packet RSSI series, link quality indication and etc., can be used to classify different interference sources (i.e., Bluetooth, WiFi, microwave oven). Different from SoNIC, TIIM [17] skips the classification step and directly builds a decision tree classifier to learn under which interference patterns a particular mitigation scheme empirically achieves the best performance. The key idea of Smoggy-Link [19] is based on the observation that link quality is highly related to interference. It therefore maintains a link model to trace the relationship between interference and link quality of sender's outbound links. With such a link model, Smoggy-Link can obtain fine-grained spatio-temporal link information to perform adaptive link selection and transmission scheduling.

REPE [13] utilizes the observation that RF interference typically manifests as an additive increase in RSSI [25]. It samples the RSSI values per symbol with a high-resolution hardware timer (i.e., 62.5kHz). By calculating the difference between RSSI[i] (i.e., the combination of ZigBee signal and WiFi interference) and $RSSI_{base}$ (i.e., ZigBee signal strength without interference), REPE utilizes a single threshold-based approach to detect incorrect symbols. CARE [14] also exploits the in-packet RSSI time series to compute the corruption level of a packet. The corruption estimation component in CARE achieves significantly improvement compared with REPE due to careful selection of two thresholds. Then an adaptive coding scheme which is based on the corruption level is designed to retransmit the redundancy information.

However, as we shown in Section III, the difference between RSSI[i] and $RSSI_{base}$ suffers inevitable errors in a highly noisy environment (i.e., noise and interference are at the same level), and hence it degrades corruption detection accuracy. Different from REPE and CARE, our approach introduces a more accurate indicator per-byte SINR from the per-byte RSSI time series to detect corruption. In addition, we use the link-layer information (pilot symbols) to further improve corruption detection accuracy.

III. Limitation of In-packet RSSI Approach

In-packet RSSI values have been used in prior work such as CARE and REPE for corruption estimation. Assume we have obtained an array of RSSI values, RSSI[n], which corresponds to a received packet of n bytes. Each element in the array RSSI[i] ($1 \leq i \leq n$) indicates the Received Signal Strength in dBm for the i-th byte in the received packet.

Both CARE and REPE detect corrupted bytes by using the difference between RSSI[i] and $\text{RSSI}_{base} = \min_i$ (RSSI[i]) as an indicator. RSSI_{base} roughly represents the ZigBee signal strength without interference, while RSSI[i] represents the combination of ZigBee signal and WiFi interference. If the RSSI difference, ΔRSSI[i], exceeds a threshold, it is highly probable that there exists a high interference and the corresponding byte is corrupted. We argue that ΔRSSI[i] may not be a robust indicator in some circumstances.

Before we perform a detailed analysis, we first introduce the following notations and formulas:

- The received power for the i-th byte is denoted as $P_{mW}[i]$ which can be split into three components: $P^S_{mW}[i]$, $P^N_{mW}[i]$, $P^I_{mW}[i]$, representing the powers of signal, noise, and interference, respectively. We assume that the powers are additive [25], i.e., $P_{mW}[i] = P^S_{mW}[i] + P^N_{mW}[i] + P^I_{mW}[i]$.
- The power in mW (P_{mW}) can also be expressed in dBm (P_{dBm}) (and vice versa) by the following formula:

$$P_{dBm} = 10\log_{10} P_{mW} \tag{1}$$

ΔRSSI[i] as an indicator. Based on the above notations, we can compute ΔRSSI[i] as follows:

$$\begin{aligned}\Delta RSSI[i] =& RSSI[i] - RSSI_{base}\\ =& 10\log_{10}(P^S_{mW}[i] + P^N_{mW}[i] + P^I_{mW}[i])\\ & - 10\log_{10}(P^S_{mW}[i] + P^N_{mW}[i])\\ =& 10\log_{10}\left(\frac{P^S_{mW}[i] + P^N_{mW}[i] + P^I_{mW}[i]}{P^S_{mW}[i] + P^N_{mW}[i]}\right)\\ =& 10\log_{10}\left(1 + \frac{P^I_{mW}[i]}{P^S_{mW}[i] + P^N_{mW}[i]}\right)\end{aligned} \tag{2}$$

In both CARE and REPE, the byte corruption probability Pr_e has positive correlation with ΔRSSI[i], i.e., Pr_e increases when ΔRSSI[i] increases.

Standard indicator. In reality, a standard indicator for *correct* transmission is the signal to interference and noise ratio, denoted as SINR, which can be computed as follows:

$$SINR_{dB}[i] = 10\log_{10}\left(\frac{P^S_{mW}[i]}{P^I_{mW}[i] + P^N_{mW}[i]}\right) \tag{3}$$

The byte corruption probability Pr_e has negative correlation with $SINR_{dB}[i]$.

Analysis. When $P^N_{mW}[i] = 0$, the first indicator correctly identifies corruptions since it is true that Pr_e is large when ΔRSSI[i] is large (high interference). However, when $P^N_{mW}[i]$ increases to a large value comparable to $P^S_{mW}[i]$ (or $P^S_{mW}[i]$ decreases to a small value comparable to $P^N_{mW}[i]$), there will be inconsistency between the two indicators. Suppose $P^N_{mW}[i]$ increases to a large value, the ΔRSSI[i] will regard the byte

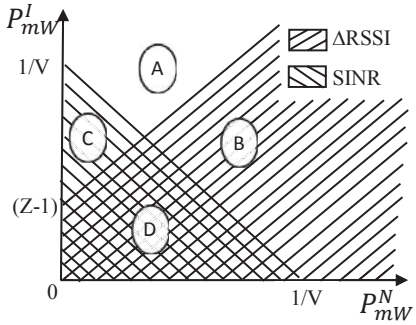


Fig. 1: Corruption detection using different indicators.

as correct since large noise makes this indicator small. On the other hand, the standard SINR indicator will regard the byte as erroneous since large noise makes SINR small.

To better understand the above description, without loss of generality, we assume $P^S_{mW}[i]$ as 1 mw and simplify the standard indicator and the ΔRSSI[i] indicator to their antilogarithm part. Then, according to the standard indicator, we regard the byte as correct when the simplified standard indicator is larger than the threshold V:

$$\frac{1}{P^I_{mW}[i]+P^N_{mW}[i]} > V \tag{4}$$

According to the ΔRSSI[i] indicator, we regard the byte as correct when:

$$1+\frac{P^I_{mW}[i]}{1+P^N_{mW}[i]} < Z \tag{5}$$

Where Z is the threshold for determining the correct byte. We plot Fig. 1 to clearly see the inconsistency. The quadrant is split into four regions by above two conditions. Regions C and D satisfy the condition (4), while Regions D and B satisfy the condition (5).

When $P^N_{mW}[i]$ increases to a large value comparable to $P^I_{mW}[i]$ (e.g., in Region B), the ΔRSSI[i] indicator classifies the byte as correct since large noise makes this indicator small, however, the standard indicator SINR will classify the byte as erroneous since large noise makes SINR small. Similarly, for Region C, since the ratio of $P^I_{mW}[i]$ and $P^N_{mW}[i]$ is large but the sum of them is small, we would make two opposite detection results using the ΔRSSI[i] indicator and the standard indicator SINR.

It is worth noting that the inconsistency in Region C can be eliminated by selecting two appropriate thresholds Z and V. However, since $Z-1$ and $1/V$ are both larger than 0, the inconsistency between the ΔRSSI[i] indicator and the standard indicator SINR always exists in Region B.

As a conclusion, the ΔRSSI[i] indicator cannot identify corruptions when noise and interference are at the same level.

IV. Design

We describe the key idea in our approach as follows. We combine the use of pilot symbols with the accurate SINR indicator aiming to significantly improve the corruption detection accuracy, especially when noise and interference are at the same level. To achieve this, we first extract useful features from cross-layer information in a packet, i.e., in-packet RSSI time series from the PHY-layer and pilot symbols from the link-layer, to build a regression-based model. We then automatically train this model using known pilot symbols.

Fig. 2 shows the overall architecture of AccuEst. It sits

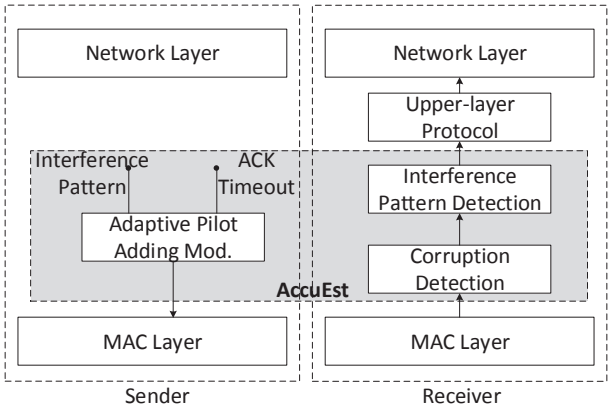


Fig. 2: The AccuEst architecture

between the MAC layer and the network layer. At the sender side, AccuEst instruments the pilot symbols evenly into the packets. The number of pilot symbols is computed according to the incoming events. (1) When the sender receives ACK/NACK packets carrying the information of interference pattern, AccuEst infers the the number of pilot symbols according to interference pattern. (2) When the ACK timer times out, AccuEst delivers the event to the upper-layer protocol. At the receiver side, when a packet is received, the receiver needs to complete two tasks: (1) if this packet is corrupted, the receiver detects corruptions and deliveries the results to the upper layer protocol. (2) No matter this packet is corrupted or not, the receiver calculates the interference pattern of this packet and transmits ACK/NACK packets that carries the interference pattern to the sender.

Implementing the above idea has the following three challenges:

- Which features are essential to corruption detection?
- How to build the model and automatically train the model?
- How to design an adaptive approach to instrument pilots?

In the rest of this section, we detail the design of AccuEst to address above challenges.

A. Feature Extraction

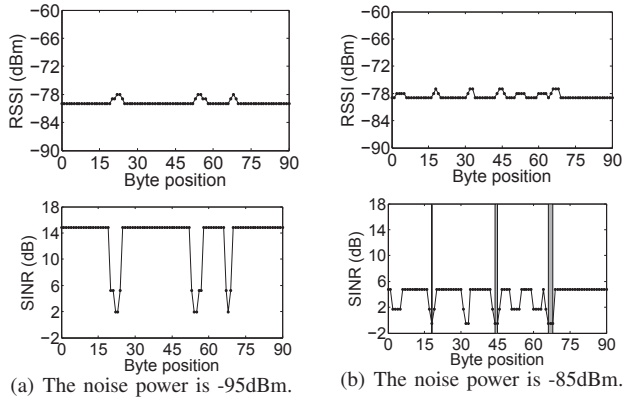
We first introduce the information that can be obtained directly from the PHY-layer and the link-layer, and then show how to extract features that are most relevant to corruption detection.

1) PHY-layer feature

Per-packet RSSI. It denotes the average RSSI value for the 8 first symbols in a received packet. It only reflects the signal strength of the packet header, therefore it has limited capacity to detect corruptions in the whole packet.

Per-packet LQI. The link quality indication (LQI) is a characterisation of the quality of a received packet. CC2420 provides an average correlation value based on the 8 first symbols to denote LQI. LQI has been used to detect the sudden changes in the packet header caused by interferers [18]. However, LQI alone provides limited help for determining erroneous bytes in the rest of the packet.

In-packet RSSI time series. We modify the radio driver in TinyOS 2.1.1 to sample RSSI at a rate of about one sample/byte. Our modified driver starts sampling RSSI whenever a SFD (Start Frame Delimiter) interrupt signals an incoming



(a) The noise power is -95dBm. (b) The noise power is -85dBm.

Fig. 3: illustration examples of corruption detection under different noise power using the standard indicator and the ΔRSSI indicator, respectively.

packet, and it keeps sampling until the last byte of the packet is received. Prior works [13, 14] simply utilize the difference between RSSI[i] of the i-th byte and $RSSI_{base}$ (i.e., ZigBee signal strength without interference) as an indicator to detect corruptions. However, as we have shown in Section III, this indicator is not robust especially when noise and interference are the same level. Fig. 3 presents the examples of detecting corruptions using the standard indicator and the ΔRSSI indicator. For the standard indicator, we regard the i-th byte as correct when SINR[i] of the i-th byte is larger than 1. For the ΔRSSI indicator, we use the threshold in CARE that when ΔRSSI[i] of the i-th byte is lower than 2, we determine the i-th byte as correct.

When interference and noise levels are relative low (i.e., as shown in Fig. 3-(a)), there are no corrupted bytes and both the standard indicator and the ΔRSSI indicator can accurately identify the correct bytes. When the noise power increases from -95dBm to -85dBm, the packet is corrupted as shown in Fig. 3-(b). The grey region denotes the erroneous bytes that are detected correctly using the corresponding indicator. We cannot identify any erroneous bytes using the ΔRSSI indicator while most of the erroneous bytes are correctly detected using the standard indicator.

Therefore, we carefully select the per-byte SINR as our indicator. To infer SINR[i] of the i-th byte, besides the in-packet RSSI time series, AccuEst samples the noise power as soon as the link turns idle after packet reception (i.e., $RSSI_n$). Let P^N_{mW} and P^S_{mW} denote the noise and the 802.15.4 signal power, respectively. The interference power of the i-th byte is $P^I_{mW}[i]$, then SINR[i] can be computed as follows.

$$\begin{aligned} P^N_{mW} &= 10^{RSSI_n/10} \\ P^S_{mW} &= 10^{RSSI_{base}/10} - P^N_{mW} \\ P^I_{mW}[i] &= 10^{RSSI[i]/10} - P^S_{mW} - P^N_{mW} \\ SINR[i] &= 10log_{10}\frac{P^S_{mW}}{P^N_{mW} + P^I_{mW}[i]} \end{aligned} \tag{6}$$

Where $RSSI_{base}$ is the minimum RSSI value during packet reception.

In summary, the PHY-layer feature vector X_i of the i-th byte is expressed as follows.

$$X_i = [SINR_i, LQI, RSSI_{pkt}] \tag{7}$$

2) Link-layer feature

Pilot symbols. Pilot symbols are the symbols which are known before decoding. They provide a noise-resistance information about whether the symbol is correct or not in a received packet. In the rest of this paper, we will use **pilot symbols** and **pilots** interchangeably. Combining the link-layer information and the PHY-layer feature makes it possible for training our model (as we shown in Section IV-B).

B. Combination of Cross-layer Information

Suppose we have obtained the L pilot symbols from the received packet. Then our goal is: given the link-layer pilot symbols and the PHY-layer feature in a sliding window with size W, train and automatically update a model to determine the corruption probability of the bytes. Formally, the j-th training set can be expressed as follows.

$$TrainS_j = [PKT_j, PKT_{j-1}, ..., PKT_{j-W+1}] \tag{8}$$

Where PKT_j is comprised of the PHY-layer feature as follows.

$$PKT_j = [X^j_1, X^j_2, ..., X^j_L] \tag{9}$$

Where X^j_i denotes the i-th pilot PHY-layer feature of the j-th packet (i.e., $SINR^j_i, LQI^j, RSSI^j_{pkt}$). In order to train the corruption detection model, we have the following label for the i-th pilot symbol in j-th packet.

$$P(Y = 1|X^j_i) = \begin{cases} 0, correct \\ 1, erroneous \end{cases} \tag{10}$$

Where $Y = 1$ denotes the byte is erroneous. If the pilot symbol is erroneous then we set $P(*) = 1$, meaning that the error probability of the byte is 1 and vice visa.

The corruption detection model should be lightweight to run on resource-constrainted sensor node. Therefore, we utilize the Logistic regression (LR) classification model to detect corrupted bytes. Logistic regression has been utilized by much prior works [26, 27] and it is easy to implement on sensor node.

The logistic regression classifier can be expressed as:

$$\begin{aligned} P(Y = 1|X^j_i) &= \frac{1}{1 + exp(-f(X^j_i))} = h_\beta(X^j_i) \\ P(Y = 0|X^j_i) &= \frac{exp(-f(X^j_i))}{1 + exp(-f(X^j_i))} = 1 - h_\beta(X^j_i) \end{aligned} \tag{11}$$

Where $P(Y = 1|X^j_i)$ and $P(Y = 0|X^j_i)$ denote the erroneous and correct probability of the byte, respectively. $f(X^j_i) = \beta_0 + \sum_{k=1}^{M} \beta_k X_k$ and M is the number of features. In order to train the parameters $\beta(i.e., [\beta_0, ..., \beta_M])$, we maximize the log likelihood below:

$$J(\beta) = \sum_{i=1}^{L} Y^j_i log(h_\beta(X^j_i)) + (1 - Y^j_i) log(1 - h_\beta(X^j_i)) \tag{12}$$

We then apply stochastic gradient descent (SGD) to update parameters β. SGD is an online algorithm that operates by repetitively drawing a fresh random sample and adjusting the weights on the basis of this single sample only [27]. Then the gradient of the i-th pilots of the j-th packet for the feature $SINR^j_i$ (i.e., the k-th feature) can be expressed as:

$$\nabla J_k(\beta) = (Y^j_i - h_\beta(SINR^j_i))SINR^j_i \tag{13}$$

The update procedure of β based on gradient is shown as follows:

$$\beta_k \leftarrow \beta_k + \lambda \nabla J_k(\beta) \tag{14}$$

Where λ is learning rate and we set λ to 0.01 in our evaluation.

C. Adaptive Pilot Instrumentation

We now further improve corruption detection accuracy by combining the pilot symbols with the PHY-layer feature. However, instrumenting pilots would also degrade network throughput. Therefore, how to minimize the number of instrumented pilot symbols while keeping high accuracy of corruption detection is a challenging issue.

To better understand the benefits of combining the pilot symbols with the PHY-layer features, we first conduct an experimental study to show the impact of different interference patterns on the accuracy of inferred PHY-layer features. The details of experimental settings are similar to that in Section V, except that an extra node is bounded with the receiver node and is enabled always-on RSSI sampling to obtain the ground truth of PHY-layer features. Our approach can detect the potential corrupted symbols, which CARE cannot identify, despite of any noise power as shown in Section IV-A. We thus only focus on four different scenarios (as shown in Table 1) that have different interference power and traffic pattern. The noise power is generated (i.e., -90dBm) using USRP [28]. Before analyzing experimental results, we introduce the following features to quantify interference patterns:

PAPR. It is a common measurement for the fluctuation of signal power and can be used to distinguish different PHY modulation techniques. We apply PAPR to analyze the WiFi interference power level. As previous studies have shown [15], 802.11g/n have a large PAPR ($\geq$ 1.9) while ZigBee has a relatively small PAPR ($\leq$ 1.3) because they employ the single-carrier modulation techniques. Suppose the normalized RSSI sequence of a N-byte packet is $nRSSI$, the PAPR of this packet can be calculated as:

$$PAPR = \frac{max\{nRSSI[i]^2 | 0 \leq i \leq N\}}{\overline{nRSSI^2}} \tag{15}$$

where $\overline{nRSSI^2}$ denotes the average of the squared values of the elements in the normalized RSSI sequence $nRSSI$.

Bursty level. Error burst means a sequence of corrupted symbols that may contain subsequences of at most four consecutive correct symbols in a packet. Prior work has observed that 802.15.4 corruptions under WiFi interference are highly bursty and the bursty density is utilized to classify different interference type [1, 18, 29]. We use a simple threshold-based approach to identify the start and the end points of each burst segment. The threshold thd is set to 2dB according to [18]. Given the RSSI series $RSSI[i]$, the sets of start (S) and end (E) position can be expressed as follows.

$$\begin{aligned} S = \{s | & RSSI[s-1] - RSSI_{base} < thd, \\ & RSSI[s] - RSSI_{base} \geq thd\} \\ E = \{e | & RSSI[e] - RSSI_{base} \geq thd, \\ & RSSI[e+1] - RSSI_{base} < thd\} \end{aligned} \tag{16}$$

The bursty level can be computed as the average bursty length.

Fig. 4 shows relative errors of estimating PHY-layer features under four different interference scenarios (as shown in Table 1). The label *SINR-xburst* means the feature per-byte SINR is evaluated under bursty level x. Only the results of bursty levels 1, 10, 11, 16 are shown due to space limitation.

Empirical analysis of the impact of interference patterns on PHY-layer features. The accuracy of per-packet RSSI and per-packet LQI has been validated to maintain relatively high in most interference patterns [18].Therefore, four interference patterns have little impact on above two PHY-features.

Table 1: Four scenarios considering interference power (PAPR) and average number of consecutive corrupted symbols (bursty level). H(high), L(low), I(interference), B(bursty).

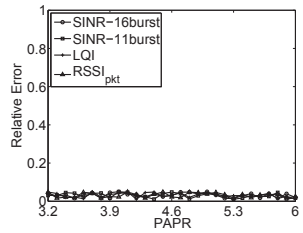
Scenarios	PAPR	Bursty Level
HIHB (Nearby video streaming in office)	$\geq$ 3.2	10$\sim$16
HILB (Nearby browser in office)	$\geq$3.2	$\leq$10
LIHB (Faraway video streaming in office)	1$\sim$3.2	10$\sim$16
LILB (Faraway browser in office)	1$\sim$3.2	$\leq$10

The accuracy of per-byte SINR is highly affected by the measured noise power that could be interfered. The noise power is recorded when the link turns to idle. When the interference strength is large (i.e., $PAPR > 3.2$), the received power is highly possible to be larger than the CCA threshold, thus we can filter the interfered noise and sample the correct noise power with high probability. Therefore, the relative error of inferred per-byte SINR is small (as shown in Fig. 4-(a) and (b)). When interference strength is small ($1 < PAPR < 3.2$), the interfered noise power could be smaller than the CCA threshold and then is recorded. Fig. 4-(c) and (d) present the relative error of inferring PHY-layer features under LIHB and LILB. First, we see the relative error is increasing with the increased PAPR (i.e., from 1.3 to 2.4), because the WiFi interference starts to appear and leads to interfered PHY-layer features. Then, the relative error is decreasing with the increased PAPR (i.e., from 2.4 to 3.2), because the interference power starts to become larger and the interfered PHY-layer features can be filtered by the CCA threshold. Besides, the relative error is increased when bursty level is high, the reason is that the more the consecutive corrupted bytes are, the higher the probability of recording interfered noise power is.

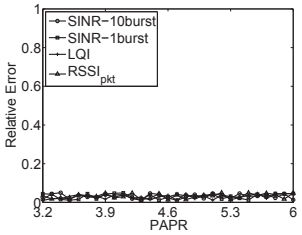
Calculating interference pattern. Based on the above observations, we design a simple but effective interference pattern calculation algorithm to guide the instrumentation of pilots (outlined in Algorithm 1). The input of the algorithm is the in-packet RSSI time series sampled during packet reception. The output of the algorithm is the interference pattern which determines the number of instrumented pilots. We first detect different interference power level according to HighInterf, then the interference pattern is computed by the multiplication of PAPR_d (i.e., the relative ratio to the PeakPapr) and burst_p (i.e., the ratio to the range of bursty level) as shown in line 14 of Algorithm 1.

Fig. 5 shows the results of corruption estimation accuracy when instrumenting different number of pilots. We omit the results of high interference power scenarios due to its minor impact on accuracy (e.g., only <2 pilots are enough to achieve >90% accuracy when the Logistic model is converged). The results from Fig. 4 guide us to set the upper and lower bound of pilots number (i.e., MaxBurst and MinBurst).

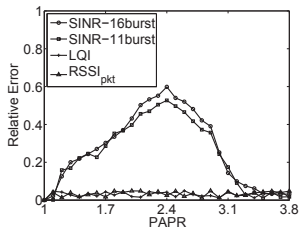
Adaptive pilot instrumentation at sender. There are two steps to adaptively instrument pilots at the sender side. (1) Obtaining interference pattern (IPT) corresponding to the packet for transmission. The IPT is obtained from IPT_buffer according to pkt_id. IPT_buffer stores the key-value pair (IPT, pkt_id) that is extracted from the feedback information of the receiver. It means that the receiver determines IPT for the packet with pkt_id. The IPT is set to 0 when there is no corresponding pkt_id. (2) Obtaining the number of



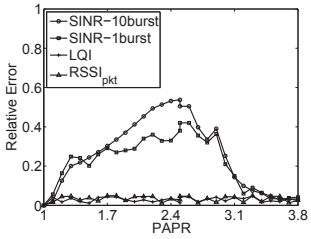
(a) Impact of high interference power ($PAPR > 3.2$) and high bursty level ($11 \sim 16$).



(b) Impact of high interference power ($PAPR > 3.2$) and low bursty level ($1 \sim 10$).

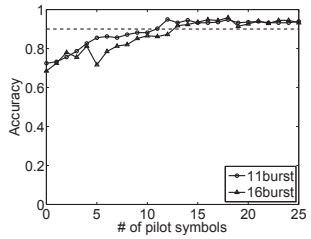


(c) Impact of low interference power ($1 < PAPR < 3.2$) and high bursty level ($11 \sim 16$).

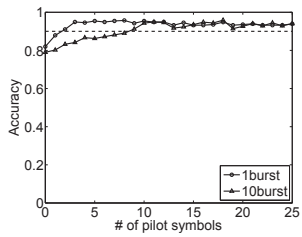


(d) Impact of low interference power ($1 < PAPR < 3.2$) and low bursty level ($1 \sim 10$).

Fig. 4: Impact of interference power (PAPR) and average number of consecutive corrupted symbols (bursty level) on the relative error of inferred PHY-layer features.



(a) Low interference power and high bursty level scenario.



(b) Low interference power and low bursty level scenario.

Fig. 5: Impact of the number of instrumented pilot symbols on corruption estimation accuracy.

instrumented pilots according to interference pattern. The number of pilots is determined by the multiplication of IPT and the range of pilots. When the sender is under model convergence state (i.e., IniPktC$\geq$0), the number of pilots is set to MaxPilot to let the model converge quickly.

Adaptive pilot instrumentation at receiver. When a packet arrives, the receiver needs to complete four tasks. (1) Obtaining pilots. The receiver first obtains IPT according to this packet id, pkt_id, from IPT_buffer. Note that both receiver and sender maintain the same IPT_buffer. It is generated by the receiver and is sent back to the sender. The receiver does not extract pilots when there is no corresponding pkt_id. Then the number of pilots is calculated following the same way as the sender (as shown in Algorithm 2). (2) Discarding abnormal pilots. When the link is unreliable, the feedback information could be lost at the sender, resulting in that the receiver extracts wrong pilots. When the difference of corruption estimation results inferred by pilots and our model exceeds a threshold AbPilots, the receiver regards the pilots as abnormal and discards them. (3) Updating the model and detecting corruptions. The model is updated when there are available pilots. Besides detecting corruptions, the receiver can

Algorithm 1: Interference pattern calculation at receiver

Input : RSSI sequence RSSI[L]
Output: Interference pattern IPT

```
PAPR = get_papr(RSSI[L]);
bursty = get_bursty_level(RSSI[L]);
burst_p = 0; PAPR_d = 0;
if PAPR > HighInterf then
    return 0;
if bursty ≤ MaxBurst then
    burst_p = (bursty - MinBurst)/(MaxBurst - MinBurst);
else
    burst_p = 1;
if PAPR ≤ PeakPapr then
    PAPR_d = (PAPR - MinPapr)/(PeakPapr - MinPapr);
else
    PAPR_d = 1 - (PAPR - PeakPapr)/(MaxPapr - PeakPapr);
IPT = PAPR_d*burst_p;
return IPT;
```

Algorithm 2: Adaptive pilot instrumentation at sender

```
convergence packet count IniPktC; interference pattern IPT;
pilot_size = MaxPilot - MinPilot;
case pkt received from network layer do
    if IniPktC ≥ 0 then
        evenly instrument MaxPilot symbols to pkt;
        decrease IniPktC;
    else
        find IPT in IPT_buffer corresponding to this pkt_id;
        p_size = MinPilot + pilot_size*IPT;
        evenly instrument pilots to pkt according to p_size;
    send pkt and start ACK timer;
case ACK timer times out do
    notify upper-layer protocol ACK timer times out;
case ACK/NACK received do
    extract (IPT, pkt_id) from ACK/NACK;
    store (IPT, pkt_id) into IPT_buffer;
    notify upper-layer protocol ACK/NACK;
```

calculate SER (symbol error rate) from estimated corruptions. (4) Calculating interference pattern. The interference pattern, IPT, is calculated as Algorithm 1 and sent back to the sender using ACK/NACK. The key-value pair (IPT, next_packet_id) are stored into IPT_buffer.

V. Evaluation

In this section, we evaluate the performance of AccuEst, and compare AccuEst with the state-of-the-art, i.e. CARE [14]. We also deploy AccuEst on an indoor testbed running CTP protocol [30] to further evaluate the system and discover its benefits to assist a coding-based transmission protocol (i.e., ACR [31]). The indoor testbed consists of 8x10 TelosB nodes and 30 of them are used in our experiments (see Fig. 6). AccuEst is implemented on the TelosB platform running TinyOS 2.1.1. The code size is $\sim$8.9 KB in ROM, and $\sim$3.6 KB in RAM. Considering a TelosB node has a total of 48 KB ROM and 10 KB RAM, this overhead is acceptable.

A. Experimental Methodology

We select overlapped channels for WiFi and CC2420 radio (i.e., channel 11 for WiFi and channel 21 for CC2420 radio).

Algorithm 3: Adaptive pilot instrumentation at receiver

```
sliding window W[]; convergence packet count IniPktC;
pilot_size = MaxPilot - MinPilot; interference pattern IPT;
case SFD rising do
    start RSSI sampling and store value in RSSI[L];
case SFD falling do
    stop RSSI sampling;
case pkt received do
    if IniPktC ≥ 0 then
        p_size = MaxPilot;
        decrease IniPktC;
    else
        find IPT in IPT_buffer corresponding to pkt_id;
        p_size = MinPilot + pilot_size*IPT;
    get feature phy[L], pilots link[] according to p_size;
    if (phy[L], link[]) is abnormal then
        discard link[];
    else
        put phy[L], link[] into W[] and update model;
    calculate IPT from RSSI[L];
    store IPT, next_pkt_id into IPT_buffer;
    if pkt is correct then
        reply ACK (IPT, next_pkt_id);
    else if pkt is error then
        estimate corruption cor_pos[] from phy[L];
        calculate SER according to cor_pos[];
        deliver_to_upperprotocol(cor_pos[], SER, pkt);
        reply NACK (IPT, next_pkt_id);
```

Fig. 6: 8x10 indoor testbed.

To present different WiFi traffic patterns [7, 18] (i.e., web browsing and video streaming), we use iperf [32] to generate different bursty levels (i.e., 3∼6M, 9∼12M TCP traffic, and 15∼18M UDP traffic, respectively). To present different noise environments (i.e., power control room and transformer vault [33]), we use USRP [28] to generate background noise levels (i.e., -85∼-82dBm, -90dBm and -98∼-95dBm). To present different SINR, we vary the distance between the transceiver pair and the interferer for different noise environments to generate SINR ranges (e.g., -10∼-6dB, -8∼-3dB, 0∼4dB). With the above tools, we can simulate practical interference scenarios like office or noisy industry environments [1, 7, 33].

Single-hop experiment settings. We use two TelosB nodes running TinyOS 2.1.1 as a transceiver pair. They communicate with each other using the CC2420 radio with the power level of 6 at a distance of 1.5m apart. The sender sends data packets with a payload of 97 bytes to the receiver with an interval of 512ms.

Multi-hop testbed experiment settings. As shown in Fig. 6, the distance between any two nodes is 0.5m. We set the radio power of each node to -32.5 dBm, resulting in a multi-hop wireless sensor network. We apply AccuEst to a coding-based transmission protocol ACR [31] to further evaluate the system. ACR relies on detected corruptions to determine the retransmitted partial packet when the decoding procedure fails. We replace the corruption detection component in ACR with our approach and CARE, respectively. We then compare the end-to-end performance metrics, i.e., packet delivery rate and data latency. The detailed experimental settings are shown in Table 1.

Table 2: Experimental settings. H(high), L(low), I(interference), B(bursty), N(noise).

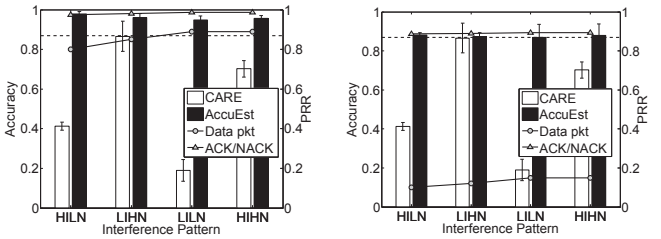
Scenarios	SINR	Traffic	Noise power
HIHB (Nearby video streaming in office)	-5∼-1dB	UDP 15∼18M	-90dBm
HILB (Nearby browser in office)	-5∼-1dB	TCP 3∼6M	-90dBm
LIHB (Faraway video streaming in office)	0∼4dB	UDP 15∼18M	-90dBm
LILB (Faraway browser in office)	0∼4dB	TCP 3∼6M	-90dBm
HIHN (Nearby interf. in power control room)	-10∼-6dB	TCP 9∼12M	-85∼ -82dBm
HILN (Nearby interf. in transformer vault)	-8∼-3dB	TCP 9∼12M	-98∼ -95dBm
LIHN (Faraway interf. in power control room)	-3∼2dB	TCP 9∼12M	-85∼ -82dBm
LILN (Faraway interf. in transformer vault)	1∼6dB	TCP 9∼12M	-98∼ -95dBm

B. Corruption Detection Accuracy

As analyzed in Section III, the ΔRSSI indicator suffers from low accuracy in a highly noisy environment (i.e., power control room and transformer vault [33]). We compare AccuEst with CARE in terms of corruption detection accuracy under different noisy environments and link reliability. Since CARE is a retransmission protocol, and we only compare AccuEst with the corruption estimation component of CARE.

As shown in Fig. 7-(a), AccuEst consistently achieves higher accuracy than CARE under all scenarios. Specifically, AccuEst improves accuracy significantly by 78.6% on average compared to CARE. The reason is that when SINR is extremely low (e.g., -10∼-6dB) or higher than -3dB, and noise level is comparable to or higher than interference strength (i.e., HIHN, LIHN and LILN), CARE would misjudge the byte as correct since the RSSI difference is negligible in such cases. Differently, AccuEst will classify the byte as erroneous since large noise makes SINR small. Moreover, AccuEst can adapt to various RSSI sensitivity of nodes using Logistic model, resulting in a higher accuracy than CARE when the RSSI difference is large (i.e., HILN).

To evaluate the impact of unreliable link on AccuEst, we reduce the power level of two TelosB nodes to 2, at a distance of 1.5m. With this configuration, the received signal strength is low, resulting in an unreliable link (<15% PRR for data packet). Fig. 7-(b) presents the corruption detection accuracy for the unreliable link. Comparing Fig. 7-(a) and Fig. 7-(b), we see that the accuracy reduction of AccuEst is small. The reason is two-fold. First, the lost of feedback information (i.e., interference pattern and packet id) may reduce the accuracy of AccuEst. However, the size of ACK/NACK packet including the feed information is much smaller than typical data packet. Therefore, as shown in Fig. 7-(b), the loss rate of ACK/NACK packet (about 10.3%) is much smaller than that of the data



(a) Corruption estimation under reliable link.

(b) Corruption estimation under unreliable link.

Fig. 7: Corruption estimation accuracy.

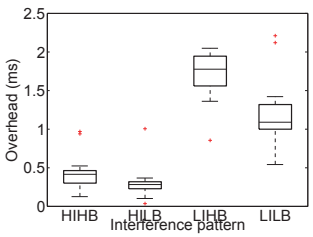


Fig. 8: Computation overhead.

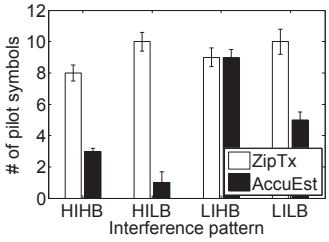


Fig. 9: Pilot overhead.

packet (about 76.4%). This mitigates the impact of unreliable link. Second, when the feedback information is lost, AccuEst incorporates an effective method to discard abnormal pilots. Therefore, the pilots instrumented at unsynchronized positions between the sender and the receiver are abandoned. This further improves the robustness of AccuEst against unreliable link.

C. Computation Overhead

The major computation overhead of AccuEst falls into the number of pilots that are used to update the parameters. The number of pilots is adjusted along with the change of interference pattern. Therefore, we evaluate computation overhead under different interference patterns. The results are obtained from 1000 packets under four interference patterns, as shown in Fig. 8. We can see from Fig. 8 that even in the worst case (i.e., LIHB) the median computation overhead is only about 1.803ms, and it is acceptable considering the level of the MAC backoff time at 5ms in expectation, and the packet transmission time at 3.5 ms for 110 bytes packet given a data rate of 250kbps. In the worst case (i.e., LIHB), the computation overhead of the classification is about 9.5% of whole computation overhead (i.e., feature calculation, parameter update and corruption classification). It is comparable to feature calculation (e.g., about 10.3%), but smaller than parameter update (e.g., about 90.2%). The reason is that parameter update needs more iterations for multiple packets in a sliding window, while both classification and feature calculation only deal with the received packet.

D. Pilot Overhead

As described in Section IV, we utilize the number of detected corruptions to estimate the SER in a packet. We thus compare the pilot overhead between AccuEst and the pilot-based approach, i.e., ZipTx [9], with respect to the relative error of estimating SER. The pilot overhead is obtained by averaging the number of instrumented pilots when achieving the same relative error under different interference patterns.

We can see from Fig. 9 that AccuEst reduces the pilot

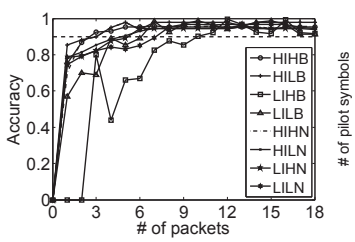


Fig. 10: Convergence speed.

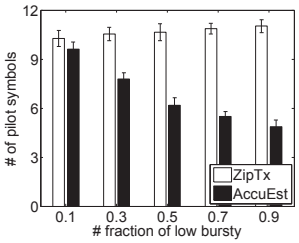


Fig. 11: Impact of dynamic interference patterns.

overhead significantly by 53.7% on average compared to ZipTx. The reason is that AccuEst can infer SER using the trained Logistic model, and only instruments small number of pilots to update the model when needed. While ZipTx always needs to instrument a number of pilots to achieve the same accuracy of SER estimation.

E. Convergence Speed

We now investigate the convergence speed of our Logistic regression model under different interference scenarios. To evaluate how the corruption detection accuracy evolves, we set the number of instrumented pilots to MaxPilot (as shown in Algorithm 3) for each packet. The experiment is repeated 10 times for each scenarios and the results are averaged. As shown in Fig. 10, we observe that the worst case is LIHB. The reason is that the LIHB scenario would change the SINR model with high probability. Then we need more pilots to converge the dynamic changing model. Under the LIHB scenario, only 11 packets (i.e., about 1.035s for 110 bytes packet, given 250kbps data rate and 0.1s inter-packet interval) are needed to make AccuEst achieve higher than 90% accuracy. This number is quite consistent for all 10 packet traces on average. We thus set the convergence packet number IniPktC to 11 to ensure our model quickly converge under any scenarios.

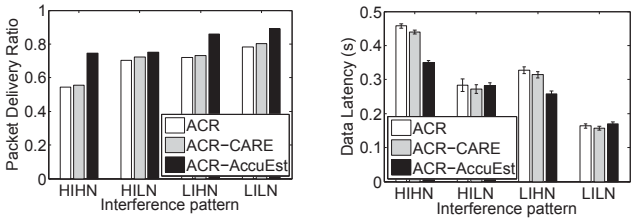
F. Impact of Dynamic Interference Pattern

In this experiment, we evaluate our adaptive pilot instrumentation component under a dynamic interference experiment. The number of pilots is small under high interference scenarios, because the SINR model is relative stable and AccuEst does not need to frequently update the model. Therefore, we only evaluate the impact of high SINR scenarios (e.g., 0∼4dB) on the performance of adaptive pilot instrumentation component. We set SINR to range [0, 4]dB while varying bursty levels. The low-bursty level fraction is computed as the ratio of the low bursty time duration and the total duration (i.e., 100 minutes in our evaluation). We compare the pilot overhead between AccuEst and ZipTx when achieving lower than 10% relative error of estimated SER.

We conduct each experiment 10 times and show the average results in Fig. 11. The result shows that our approach significantly reduces the pilot overhead by 36.1% on average, while achieving the approximately equivalent relative error compared to ZipTx. The reason is that to achieve a small relative error of estimated SER under any scenarios, ZipTx has to set the number of pilots according to the worst case. AccuEst can adapt to the interference patterns to reduce the expectation number of pilots.

G. Testbed Experiments

ACR actively converts most potential collisions into the long and short packet collision patterns to enable a lightweight FEC



(a) Packet delivery ratio. (b) Data latency.

Fig. 12: End-to-end performance comparison

scheme to recover collided packets. By default, a short ACR packet has 25 bytes data with a 16-bit CRC, and a long ACR packet consists of 11 blocks where the first 8 blocks are data and the rest are redundancy. The block size is 10 bytes. When the number of erroneous blocks is larger than 3, ACR relies on the detected corruptions to determine the retransmitted partial packets for FEC decoding. If the decoding procedure fails, ACR switches to the ARQ scheme. A block is regarded as corrupted when the difference between $RSSI[i]$ and $RSSI_{base}$ is larger than 5. We replace the corruption detection component with AccuEst and CARE, respectively. A block is regarded as corrupted when there exists at least one erroneous byte.

Packet delivery ratio. Figure 12-(a) shows the packet delivery ratio (PDR) with different corruption detection approaches. The result shows that ACR-AccuEst improves PDR by 20.3% and 18.3% on average compared to ACR and ACR-CARE, respectively. We observe that the PDR of ACR and ACR-CARE is low under HIHN, LIHN and LILN scenarios. The reason is that both ACR and ACR-CARE suffer from low accuracy of corruption estimation under these scenarios. In HIHN and LIHN, false negatives are more likely to happen (erroneous blocks are identified as correct). The FEC decoding would fail and multiple retransmissions will be incurred. In LILN, false positives are more likely to happen (correct blocks are identified as erroneous), leading to multiple retransmissions of the correct parts of the packets.

Data latency. The overhead of successfully transmitting a packet at each hop includes three parts: (1) ACR encoding and decoding. (2) MAC backoff and packet transmission. (3) Corruption estimation. The overhead of ACR encoding and decoding is 0.1ms and 0.5ms, respectively, according to our experimental results. The expected MAC backoff time is 5ms because the maximum initial backoff time in TinyOS is about 10ms. The packet transmission time is about 3.5ms for a 110-byte packet with a data rate of 250kbps on the CC2420 radio [13]. According to Fig. 8, corruption estimation typically costs 1.412ms and 0.376ms on average, under low interference and high interference scenarios, respectively. Therefore, it is acceptable in terms of the MAC backoff time and packet transmission time.

Fig. 12-(b) shows the data latency in a 5-hop network. The results show that compared to ACR-CARE, ACR-AccuEst reduces data latency by 10.2% on average under HIHN and LIHN scenarios, The reason is that under the HIHN and LIHN scenarios, the symbol error rate is relatively high, the corrupted blocks that are missed by ACR and ACR-CARE would lead to multiple FEC decoding failures and retransmissions of whole packet, resulting in large latency. Differently, ACR-AccuEst can detect more corrupted blocks and reduce extra FEC decoding overhead, only requesting the retransmissions of corrupted part of the packets.

VI. Discussion

Energy efficiency when enabling high-resolution RSSI sampling procedure. The high-resolution RSSI sampling will not incur considerable energy consumption. The reason is two-fold: 1) The RSSI value in the register is averaged over 8 symbol periods (128 μs) according to the CC2420 datasheet [34]. Hence, our implementation does not change hardware sampling and only increases the register reading rate, which incurs small energy consumption compared to radio operations. For example, TelosB [35] nodes consume 21.8mA when receiving a packet. The additional RSSI reading only adds 0.5mA [36]. Note that our approach does not turn CPU into active during duty cycling. It turns the CPU into active when receiving a packet, which lasts for at most 4ms (considering a max packet length of 128 bytes). 2) Although the energy consumption is slightly increased by RSSI readings, the energy efficiency can be improved since retransmissions can be reduced with our accurate corruption estimation.

Generality on other platforms. The core component of our approach is high-resolution RSSI sampling. We believe that for any platform that can enable high-resolution RSSI sampling (e.g. Micaz[37]) or provide fine-grained state information (CSI) of the channel where the bits are transmitted on (e.g., 5300 NIC[38]), our approach can be directly applied. For other platforms, our approach can still work for coarse-grained corruption estimation scenarios. For example, detecting corrupted blocks that typically contain multiple bytes.

Other interference types. We have built the model according to the sampled RSSI, which is adaptive to different types of interference. Our approach will perform well when there are clear patterns in the interference and may not perform well otherwise. We also note that when the interference is random, our approach can still perform better than ZipTx according to our evaluation results shown in Fig. 11.

Limitation. Considering fast transmission rate of 802.11n (e.g., typical data rate of 200Mbit/s [39]), the typical packet on-air time is 5.45 or 57.3μs according to Esense[40]. Our approach with high-resolution RSSI sampling (e.g., one sample per 32μs) may not be able to capture the corruption pattern caused by short WiFi packets.

VII. Conclusion

Accurate corruption estimation is one of the key factors in improving the resilience of ZigBee transmissions. This paper reveals the limitations of existing in-packet RSSI approaches, and uses per-byte SINR to detect corruption. We combine the link-layer information (pilot symbols) and the PHY-layer features (i.e., per-packet LQI, per-packet RSSI, and per-byte SINR) to further improve corruption detection accuracy. Besides, we design an interference pattern-aware pilot instrumentation scheme to strike a good balance between accuracy and overhead. We conduct extensive experiments including single-link and multi-hop to evaluate our approach. Our testbed results show that our approach significantly improves packet delivery ratio.

For our future work, there are multiple directions to explore. First, we will generalize our approach to other interference types. Second, we will develop a better way of instrumenting pilot symbols to further reduce the pilot overhead.

REFERENCES

[1] C. J. M. Liang, N. B. Priyantha, J. Liu, and A. Terzis, "Surviving Wi-Fi Interference in Low Power ZigBee Networks," in *Proc. of ACM SenSys*, 2010.

[2] X. G. Mobashir Mohammad and M. C. Chan, "Oppcast: Exploiting spatial and channel diversity for robust data collection in urban environments," in *Proc. of ACM/IEEE IPSN*, 2016.

[3] K. Jamieson and H. Balakrishnan, "Ppr: Partial packet recovery for wireless networks," in *Proc. of ACM SIGCOMM*, 2007.

[4] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard, "Symbol-level network coding for wireless mesh networks," in *Proc. of ACM SIGCOMM*, 2008.

[5] H. S. J. G. S. D. Anwar Hithnawi, Su Li, "CrossZig: Combating Cross-Technology Interference in Low-power Wireless Networks," in *Proc. of ACM/IEEE IPSN*, 2016.

[6] A. Koetter, Ralf; Vardy, "Algebraic soft-decision decoding of reedsolomon codes," vol. 49, no. 11, pp. 2809–2825, 2003.

[7] Z. Zhao, W. Dong, G. Chen, G. Min, T. Gu, and J. Bu, "Embracing corruption burstiness: Fast error recovery for zigbee under wi-fi interference," *IEEE Transactions on Mobile Computing*, 2016,To appear.

[8] S. Sen, N. Santhapuri, R. R. Choudhury, and S. Nelakuditi, "Accurate: Constellation based rate estimation in wireless networks," in *Proc. of USENIX NSDI*, 2010.

[9] C. J. Lin, N. Kushman, and D. Katabi, "Ziptx: Harnessing partial packets in 802.11 networks," in *Proc. of ACM MobiCom*, 2008.

[10] J. Huang, Y. Wang, and G. Xing, "LEAD: Leveraging Protocol Signatures for Improving Wireless Link Performance," in *Proc. of ACM MobiSys*, 2013.

[11] L. Wang, X. Qi, J. Xiao, K. Wu, M. Hamdi, and Q. Zhang, "Wireless rate adaptation via smart pilot," in *Proc.of IEEE ICNP*, 2014.

[12] M. Vutukuru, H. Balakrishnan, and K. Jamieson, "Cross-layer wireless bit rate adaptation," in *in Proc. of ACM SIGCOMM*, 2009.

[13] J.-H. Hauer, A. Willig, and A. Wolisz, "Mitigating the effects of rf interference through rssi-based error recovery," in *Proc. of EWSN*, 2010.

[14] W. Dong, J. Yu, and X. Liu, "CARE: Corruption-Aware Retransmission with Adaptive Coding for the Low-Power Wireless," in *Proc. of IEEE ICNP*, 2015.

[15] X. Zheng, Z. Cao, J. Wang, Y. He, and Y. Liu, "ZiSense: towards interference resilient duty cycling in wireless sensor networks," in *Proc. of SenSys*, 2014.

[16] D. Liu, Z. Cao, M. Hou, and Y. Zhang, "Frame counter: Achieving accurate and real-time link estimation in low power wireless sensor networks," in *Proc. of ACM/IEEE IPSN*, 2016.

[17] A. Hithnawi, H. Shafagh, and S. Duquennoy, "Tiim: Technology-independent interference mitigation for low-power wireless networks," in *Proc. of ACM/IEEE IPSN*, 2015.

[18] F. Hermans, O. Rensfelt, T. Voigt, and E. Ngai, "SoNIC: Classifying interference in 802.15.4 sensor networks," in *Proc. of ACM/IEEE IPSN*, 2013.

[19] X. Z. D. F. D. X. T. X. Meng Jin, Yuan He, "Smogy-link:fingerprinting interference for predictable wireless concurrency," in *Proc. of IEEE ICNP*, 2016.

[20] F. M. Sallabi, A. M. Gaouda, A. H. El-Hag, and M. M. A. Salama, "Evaluation of zigbee wireless sensor networks under high power disturbances," *Proc. of IEEE Transactions on Power Delivery*, vol. 29, no. 1, pp. 13–20, 2014.

[21] J. M. Cano-Garca and E. Casilari, "An empirical evaluation of the consumption of 802.15.4/zigbee sensor motes in noisy environments," in *Proc. of International Conference on Networking, Sensing and Control*, 2011.

[22] S. S. Hong and S. R. Katti, "DOF: A Local Wireless Information Plane," in *Proc. of SIGCOMM*, 2011.

[23] K. J. P. L. Rodrigo Fonseca, Omprakash Gnawali, "Four-bit wireless link estimation," in *Proc.of HotNets*, 2007.

[24] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *In Proc. of ACM Mobicom*, 2003.

[25] S. R. D. Ritesh Maheshwari, Shweta Jain, "On estimating joint interference for concurrent packet transmissions in low power wireless networks," in *Proc. of ACM WiNTECH*, 2008.

[26] T. Liu and A. E. Cerpa, "Foresee (4c): Wireless link prediction using link features," in *Proc. of ACM/IEEE IPSN*, 2011.

[27] T. Liu and A. E. C., "TALENT: temporal adaptive link estimator with no training," in *Proc. of ACM SenSys*, 2012.

[28] E. R. LLC, "https://www.ettus.com/," 2007.

[29] M. Spuhler, V. Lenders, and D. Giustiniano, "Blitz: Wireless link quality estimation in the dark," in *In Proc. of EWSN' 13*, 2013.

[30] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *In Proc. of ACM SenSys*, 2009.

[31] Y. Wu, G. Zhou, and J. A. Stankovic, "ACR: active collision recovery in dense wireless sensor networks," in *in Proc. of IEEE INFOCOM*, 2010.

[32] J. D. J. F. A. Tirumala, F. Qin and K. Gibbs, "Iperf: The tcp/udp bandwidth measurement tool," http://dast.nlanr.net/Projects, 2005.

[33] V. C. Gungor, B. Lu, and G. P. Hancke, "Opportunities and challenges of wireless sensor networks in smart grid," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 10, pp. 3557–3564, 2010.

[34] C. Datasheet, "http://focus.ti.com/lit/ds/symlink/cc2420.pdf," Texas Instruments, 2007.

[35] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *In Proc. of ACM/IEEE IPSN*, 2005.

[36] R. Fonseca, P. Dutta, P. Levis, and I. Stoica, "Quanto: Tracking energy in networked embedded systems," in *in Proc. of USENIX OSDI*, 2008.

[37] J. Hill and D. Culler, "Mica: a wireless platform for deeply embedded networks," *IEEE MICRO*, vol. 22, no. 6, pp. 12–24, 2002.

[38] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, "Tool release: Gathering 802.11n traces with channel state information," *Proc. of ACM SIGCOMM CCR*, vol. 41, no. 1, p. 53, 2011.

[39] I. standard 802.11n, "http://luci.subsignal.org/ jow/802.11n-2009.pdf," 2009.

[40] K. Chebrolu and A. Dhekne, "Esense: Communication through energy sensing," in *Proc. of ACM MobiCom*, 2009.