Towards Accurate Corruption Estimation in ZigBee Under Cross-Technology Interference

#1570301171

***Abstract*—Cross-Technology Interference affects the operation of low-power ZigBee networks, especially under severe WiFi interference. To improve the resilience of ZigBee transmission in wireless sensor networks, it is important to estimate corruption accurately with low overhead and no additional hardware. In this paper, we propose an accurate corruption estimation approach, AccuEst, which utilizes SINR (Signal-to-Interference-and-Noise Ratio) to detect corruption. We combine the use of pilot symbols with SINR to improve the corruption detection accuracy, especially under low interference. In addition, we design an adaptive pilots 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. Our results show that AccuEst consistently achieves better performance than two recent approaches (i.e., REPE and CARE) under low interference (i.e., 72% and 63% on average compared to REPE and CARE, respectively). We also implement AccuEst in a coding-based transmission protocol, and the result shows that with AccuEst, the packet delivery ratio is improved by 18.1% on average with a latency overhead of 4.4%.**

I. Introduction

The recent years have witnessed the unprecedented proliferation of smart wireless devices. Due to different application requirements (e.g., throughput, timeliness, and energy-efficiency), many different radio technologies are developed, such as WiFi, Bluetooth, ZigBee (802.15.4) and etc. These radio technologies usually operate on the 2.4GHz ISM band, inevitably causing Cross-Technology Interference (CTI). CTI is especially *critical* for the low-power ZigBee network that has to combat with the severe interference from ubiquitous WiFi Access Point (AP) deployment [1].

Prior studies have shown that ZigBee packets may suffer from severe corruption due to WiFi interference. Accurate corruption estimation is very important for improving the resilience of ZigBee transmission, especially when ZigBee and WiFi operate in the overlapping frequency channels. With the availability of corruption patterns, it is possible to devise better retransmission mechanisms [2, 3] or better coding schemes [4], resulting in better transmission efficiency.

To achieve an efficient and practical solution, corruption estimation in ZigBee transmission should have the following requirements:

- No hardware modification. It should work on existing COTS ZigBee devices without hardware modification. This is to ensure the solution can be directly deployed in legacy ZigBee devices.
- High accuracy. It should achieve a high level of accuracy since an accurate estimation of corruption patterns provides important guidelines to the upper-layer protocol design.
- Low overhead. It should not introduce much overhead. In addition, it should be computationally lightweight to meet the resource constraint of ZigBee devices.

Several corruption estimation approaches exist in the literature [2, 5–9], however none of them satisfies all the above requirements. The PHY-based approach, e.g., PPR [2], AccuRate [5], 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 [6], LEAD [7], rely on the *known* pilot symbols for coarse-grained BER (Bit Error Rate) estimation. Such approach suffers from an inherent tradeoff issue 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, such an approach, when used alone, is *insufficient* to identify corruption in a packet.

Recently, in-packet RSSI sampling technique has accelerated many interesting research works [3, 10–14]. In-packet RSSI sampling, working at the maximum sampling rate, can provide *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, has been used in several works, to classify interference sources, or corruption estimation in the presence of WiFi interference. The in-packet RSSI-based approach, e.g., REPE [10], and CARE [3], can work on COTS ZigBee devices with no packet overhead. However, they suffer from *low accuracy*, especially in low interference environments (as shown in Section III).

Aiming to satisfy all the above requirements, in this paper we propose a novel corruption estimation approach, named AccuEst, with the following key techniques.

- We explore *cross-layer* information to achieve better corruption estimation accuracy. (1) AccuEst uses PHY-layer information offered by COTS ZigBee devices, including per-packet RSSI, per-packet LQI, and per-byte RSSI values. In particular, the per-byte RSSI values can capture the impact of interference (e.g., burtiness, fluctuations) in a *fine-grained* manner. Using this information, we further derive the per-byte SINR (signal to interference and noise ratio) which can better indicate whether the byte is corrupted or not. (2) It also uses pilot information from the link-layer to further improve the accuracy, especially in low-interference scenarios. AccuEst uses *cross-layer* information to automatically train a regression-based model for corruption predication. This model is trained on-line, and yields high predication performance when a sufficient number of packets are transmitted and received. The exploitation of cross-layer information enables AccuEst to achieve higher accuracy than existing approaches.

- We explore *Adaptive* instrumentation of pilot symbols to further improve the corruption estimation accuracy. We investigate in this paper that the use of PHY-layer information alone cannot achieve sufficiently high accuracy. For example, the accuracy of sampled noise would be degraded due to the superposition of interference, thus reducing the corruption detection accuracy. Especially under low interference scenarios, the reduction is obvious. AccuEst instruments pilot symbols to enhance corruption estimation. The instrumentation of pilot symbols is *adaptive* to strike a good balance between accuracy and overhead: the key idea of adaptively instrumenting the pilot symbols is that we instrument fewer pilots under the high interference levels and vice visa.

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 two recent approaches (i.e., REPE and CARE). Specifically, the improvement of corruption detection accuracy is obvious under the low interference scenario (i.e., 72% and 63% on average compared to REPE and CARE, respectively). AccuEst significantly reduces pilot overhead by 16.1% compared to the traditional pilot based approach while achieving the equivalent relative error of SER (symbol error rate) estimation. To demonstrate its application in real secnarios, we implement AccuEst in a coding-based transmission protocol. The testbed results show that with AccuEst, the packet delivery ratio is improved by 18.1% with incurring only 4.4% latency overhead at most.

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 under low interference.
- We propose a novel corruption estimation approach to satisfy all the three requirements through . Our approoach exploits cross-layer information, and employs a novel machine-learning based model to combine different information sources to achieve high accurate corruption prediction.
- We implement AccuEst on the TelosB platformm with TinyOS 2.1.1, and evaluate its performance extensively. The results show that AccuEst significantly improves the corruption detection accuracy compared to REPE and CARE under the low interference scenario.

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, and finally, Section VI concludes this paper and discusses future research directions.

II. Related Work

In this section, we discuss the related work most relevant to ours. Section II-A and Section II-B introduce traditional corruption estimation approaches, i.e., PHY-based approaches which require hardware modifications, and pilot-based approaches which insert known pilot symbols to packets. In Section II-C, we introduce the recent in-packet RSSI sampling technique and how the per-byte RSSI time series can benefit upper-layer protocol design.

A. PHY-based Approach

The detailed PHY-layer information provides accurate hints and it has been utilized by many prior works to identify erroneous bits [2], estimate BER [5, 9] or classify the interference type [15, 16].

PPR [2] implements an expanded PHY-layer interface called SoftPHY that provides the detailed PHY-layer hints 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 the incorrect bits and request the retransmission of only the selected portions of a packet where there are bits likely to be wrong. AccuRate [5] exploits the dispersions of the symbol constellation space to compute the optimal bit rate. The smaller dispersion means the 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 the packet could transmit at.

Besides the above related works which directly estimate corruptions, there are also related works which accurately detect the interference sources. DoF [15] utilizes FFT operation of the signal to extract the repeating hidden patterns of different wireless protocols and then classifies the interference sources according to the extracted patterns. CrossZig [16] 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.

All above approaches require modifications in the hardware. Different from them, our approach is directly supported by COTS ZigBee devices.

B. Pilot-based Approach

Pilot symbols/bits are the symbols/bits which are known before decoding and they are useful information to estimate BER [6, 8] (Bit Error Rate) or assist the channel decoding [7].

LEAD [7] 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 the decoding of whole packet. SmartPilot [8] further extracts more pilots from both detailed PHY-layer information and upper layer protocol headers to estimate the BER and then picks a good data rate. However, they both rely on the hardware modifications and it limits their usage on current ZigBee devices.

ZipTx [6] is a software-only approach for recovering the partial packets. It evenly instruments the known pilot bits into the transmitted packet at the sender side. The receiver uses these know pilot bits to estimate the overall BER of the packet. Then the high BER packets are requested for retransmission while the low BER packets are decoded by error correcting codes.

Pilot-based approaches do not require hardware modifications. However, they provide coarse-grained information which is insufficient to precisely localize corruptions. Besides, they incur relatively large packet overhead due to pilot symbols. To address this issue, we design an adaptive pilot instrumentation method to strike a good balance between accuracy and overhead.

C. In-packet RSSI Approach

In-packet RSSI sampling can provide fine-grained per-byte RSSI values for a packet and it has accelerated a lot of interesting works [3, 10–14].

SoNIC [14] 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 for classifying the different interference sources (i.e., Bluetooth, WiFi, microwave ovens). Different from SoNIC, TIIM [13] 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.

REPE [10] utilizes the observation that RF interference typically manifests as an additive increase in RSSI [17]. It samples the RSSI values per symbol with a high hardware resolution timer (i.e., 62.5kHz). By calculating the difference between the RSSI[i] (the combination of ZigBee signal and WiFi interference) and the $RSSI_{base}$ (the ZigBee signal strength without interference), REPE utilizes a single threshold-based approach to detect the incorrect symbols and requests retransmission for them. CARE [3] also exploits the in-packet RSSI time series to compute the corruption level of a packet. The corruption estimation component in CARE achieves significantly improvements 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 the RSSI[i] and the $RSSI_{base}$ suffers inevitable errors under low interference scenario and it degrades the corruption detection accuracy. Different from REPE and CARE, our approach carefully calculates the accurate indicator SINR from the per-byte RSSI time series to detect corruptions. Besides, we use the link-layer information (pilot symbols) to further improve the corruption detection accuracy.

III. Limitation of In-packet RSSI Approach

In-packet RSSI values have been used in prior works, e.g., 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 [17], 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:

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

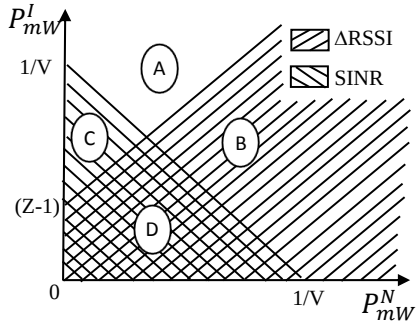


Fig. 1: Corruption detection using different indicators.

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 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:

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

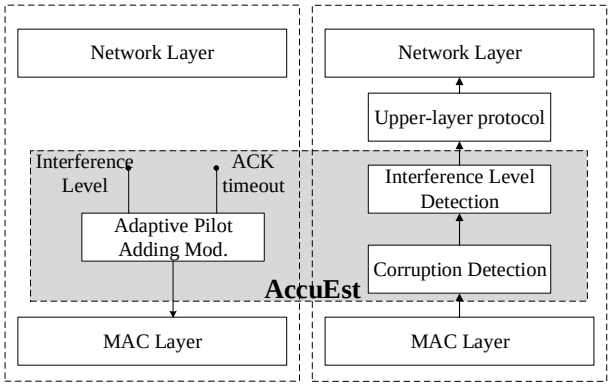


Fig. 2: The AccuEst architecture

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 and it is larger than 1. We plot the Figure 1 to clearly see the inconsistency. The quadrant is split into four regions by above two conditions. The region C and region D satisfy the condition (4), while the region A and region B satisfy the condition (5).

When $P^N_{mW}[i]$ increases to large value comparable to $P^I_{mW}[i]$ (e.g., in the 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 the region C, since the ratio of $P^I_{mW}[i]$ and $P^N_{mW}[i]$ is large but the sum of them is small, then we would make two opposite determinations using the ΔRSSI[i] indicator and the standard indicator SINR, respectively.

It is worth noting that the inconsistency in region C can be removed 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 the region B.

As a conclusion, the ΔRSSI[i] indicator cannot identify corruptions when the noise is large or the signal and interference are relatively small.

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 the noise level is high or the strength of interference is relatively low. To achieve this, we first extract useful features from cross-layer information in a packet, i.e., in-packet RSSI 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.

Figure 2 shows the overall architecture of AccuEst. It sits between the MAC layer and the network layer.

At the sender side, AccuEst instruments the pilots evenly into the packets that are transmitted from the network layer. The number of pilot are computed according to the incoming events. There are two kind of events: (1) the sender receives the ACK carrying the information of interference level (i.e., the strength and the burstiness of the interference). (2) The ACK timer times out. For the former events, the number of pilot is inferred according to interference level. Initially, we instrument more pilot symbols to let the model be converged quickly. For the latter, the sender resets the number of pilot to zero and deliveries the event to upper-layer protocol.

At the receiver side, AccuEst stores a dictionary where the key is packet sequence number and the value is the interference level. When a packet is received, the receiver uses a high-resolution RSSI sampling procedure to sample the RSSI values during packet reception. Then, the interference level is obtained from the dictionary according to the received packet sequence number. The receiver calculates the position of pilot symbols according to the interference level. With the known pilot symbols and the feature extracted from PHY-layer, the model is trained and updated. The pilot position of next packet is inferred from current packet. Thus AccuEst calculates the interference level of current packet according to the strength and the burstiness of the interference. The interference level is transmitted to sender. The key-value pair of next packet sequence number and the interference level are stored in the dictionary.

Implementing the above idea has three important challenges:

- Which features are essential for corruption detection?
- How to build the model and automatically train the model?
- How to adaptively instrument pilots?

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

A. Feature Extraction

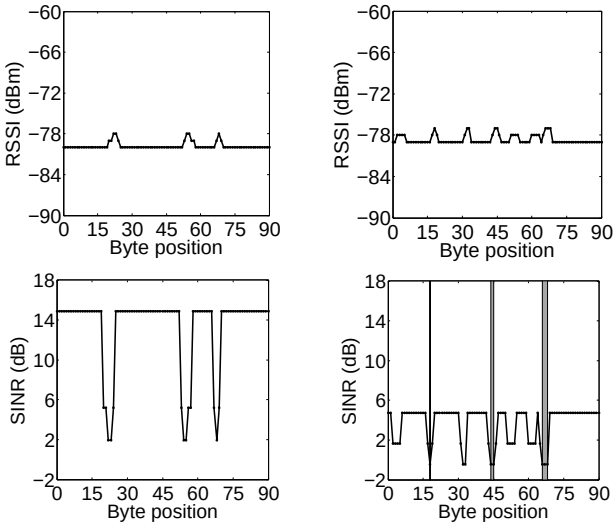
We first introduce the information that can be obtained directly from the PHY-layer and link-layer, and then show how to extract features that are most relevant to corruption detection.

1) PHY-layer feature

Per-packet RSSI. It denotes the average RSSI value for the 8 first symbols in a received packet and can be obtained directly from the end of the packet (i.e., $RSSI\_P$). It only reflects the signal strength of the packet header, therefore it has limited capacity to detect corruptions in whole packet.

Per-packet LQI. The link quality indication (LQI) is a characterisation of the quality of a received packet. It is related to the SNR of current link. CC2420 provides an average correlation value based on the 8 first symbols to denote LQI and appends LQI to the end of the received packet [18]. LQI has been used to detect the sudden changes in the packet header caused by interferers [14]. However, LQI alone provides limited help for determining erroneous bytes in the rest of packet.

In-packet RSSI time series. We modify the radio driver in TinyOS 2.1.1 to sample the RSSI at a rate of about one sample/byte. Our modified driver starts sampling RSSI whenever an SFD (Start Frame Delimiter) interrupt signals an incoming packet, and it keeps sampling until the last byte of the packet is received. Prior works [3, 10] simply utilize the difference between the RSSI[i] of the i-th byte and $RSSI_{base}$ (i.e., ZigBee signal strength without interference) as an indicator to detect the corruption. However, as we have shown in Section III, this indicator is not robust especially under low interference.



(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.

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 the SINR[i] of the i-th byte is larger than 1. For the ΔRSSI indicator, we use the threshold in CARE that when the ΔRSSI[i] of the i-th byte is lower than 2, then we determine the i-th byte as correct.

When the interference and the noise power is relative low (i.e., as shown in Fig. 3-(a)), there is no corrupted bytes and both the standard indicator and the ΔRSSI indicator can identify the correct bytes accurately. 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 erroneous bytes are correctly detected using the standard indicator.

Therefore, we carefully calculate the per-byte SINR[i] as our indicator. To infer the 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 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:

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

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

Due to the deviation of RSSI sampling, samples are not perfectly aligned with each byte, thus we extract the feature of the i-th byte from the $SINR[i-q:i+q]$ as follows.

- $nMedian_i$. The array SINR[n] in a packet is normalized to the range [0,1] for training the model (i.e., $nSINR[n]$). $nMedian_i$ denotes the median normalized SINR in $nSINR[i-q:i+q]$.
- $nRange_i$. It is the range of the $nSINR[i-q:i+q]$ and can be computed as:

$$Max(nSINR[i-q:i+q]) - Min(nSINR[i-q:i+q]) \tag{7}$$

- $nAve_i$. The average value of normalized SINR within the $nSINR[i-q:i+q]$.

In summary, the PHY-layer feature vector $\boldsymbol{X_i}$ of the i-th byte is expressed as follows:

$$\boldsymbol{X_i} = [nMedian_i, nRange_i, nAve_i, LQI, RSSI\_P] \tag{8}$$

2) Link-layer feature

Pilot symbols. Pilot symbols are the symbols which are known before decoding. They provide the ground truth about whether the symbol is correct or not in a packet. Combining the ground truth 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 the size W, train and automatically update the model to determine the corruption probability of the bytes. The sliding window size W is obtained empirically and set to 4. Formally, the j-th training set can be expressed as:

$$\boldsymbol{TrainS_j} = [\boldsymbol{PKT_j}, \boldsymbol{PKT_{j-1}}, ..., \boldsymbol{PKT_{j-W+1}}] \tag{9}$$

Where $\boldsymbol{PKT_j}$ is comprised of the PHY-layer feature as follows:

$$\boldsymbol{PKT_j} = [\boldsymbol{X^j_1}, \boldsymbol{X^j_2}, ..., \boldsymbol{X^j_L}] \tag{10}$$

In order to train the corruption detection model, we have the following label for the i-th pilot symbol in j-th packet.

$$P(Y=1|\boldsymbol{X^j_i}) = \begin{cases} 0, correct \\ 1, erroneous \end{cases} \tag{11}$$

Where $Y=1$ denotes the byte is erroneous. If the pilot symbol is correct then we set the $P(*)=0$ and it means that the error probability of the byte is 0 and vice visa.

The corruption detection model should be lightweight to run on the constraint sensor node. Therefore, We utilize the logistic regression (LR) classification model to detect the corrupted bytes. Logistic regression has been utilized by many prior works [19, 20] and it is easy to implement on sensor node.

The logistic regression classifier can be expressed as:

$$\begin{aligned} P(Y=1|\boldsymbol{X}) &= \frac{1}{1+exp(-f(\boldsymbol{X}))} = h_{\boldsymbol{\beta}}(\boldsymbol{X}) \\ P(Y=0|\boldsymbol{X}) &= \frac{exp(-f(\boldsymbol{X}))}{1+exp(-f(\boldsymbol{X}))} = 1 - h_{\boldsymbol{\beta}}(\boldsymbol{X}) \end{aligned} \tag{12}$$

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

$$J(\boldsymbol{\beta}) = \sum_{i=1}^{L} Y_i log(h_{\boldsymbol{\beta}}(\boldsymbol{X_i})) + (1-Y_i) log(1-h_{\boldsymbol{\beta}}(\boldsymbol{X_i})) \tag{13}$$

Then we apply stochastic gradient descent (SGD) to update the parameters $\boldsymbol{\beta}$. 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 [20]. Then the gradient of the pilot symbol sample $Y, \boldsymbol{X}$ for the k-th feature can be expressed as:

$$\nabla J_k(\boldsymbol{\beta}) = (Y - h_{\boldsymbol{\beta}}(X_k))X_k \tag{14}$$

The update procedure of $\boldsymbol{\beta}$ based on gradient is follows:

$$\beta_k \leftarrow \beta_k + \lambda \nabla J_k(\boldsymbol{\beta}) \tag{15}$$

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

C. Adaptive Pilots Instrumentation

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

We observe that when AccuEst instruments few pilot symbols, the corruption detection accuracy of our model is relative low under low interference levels, while the accuracy is still high under high interference levels (as shown in Section V-B). The accuracy of sampled noise would be degraded due to the superposition of interference, thus reducing the corruption detection accuracy. Especially under the low interference scenario, the reduction is obvious. We instrument more pilot symbols to quickly update the model to compensate for the reduction. When the interference is relative high, the noise has smaller impact on the corruption detection accuracy, thus fewer pilot symbols are needed. The key idea of adaptively instrumenting the pilot symbols is that we instrument fewer pilots under the high interference levels and vice visa.

We introduce following features to quantify the interference level:

PAPR. It is a common measure of the fluctuation of signal power and can be used to distinguish different PHY modulation techniques. We apply PAPR to analyze the WiFi interference level. As previous studies have shown [11], 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 sequences of a N byte packet is $nRSSI$, the PAPR of this packet can be calculated as:

$$PAPR = \frac{max\{nRSSI[i]^2 | 0 \leq i \leq N\}}{\overline{nRSSI^2}} \tag{16}$$

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

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 works have observed that 802.15.4 corruptions under WiFi interference are highly bursty and utilized the bursty density to classify different interference type [1, 14]. 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 be 2dB according to [14]. Given the RSSI series $RSSI[i]$, the sets of start (S) and end (E) position can be expressed as:

$$\begin{aligned} S = \{s | RSSI[s-1] - RSSI_{base} < thd, \\ RSSI[s] - RSSI_{base} \geq thd\} \\ E = \{e | RSSI[e] - RSSI_{base} \geq thd, \\ RSSI[e+1] - RSSI_{base} < thd\} \end{aligned} \tag{17}$$

The bursty level then can be computed as the ratio of the sum of the all bursty length and the number of burst.

The interference level in the current packet is computed as the average of the normalized PAPR and bursty level. Due to dynamic WiFi interference, a simple way to predict the future interference level is to focus more on the most recent interference level to quickly adjust it rather than on a long series. Thus we calculate the exponential moving averages of the W sliding window to predict the interference level in near future.

Algorithm 1 Interference level calculation at receiver

```
1:  function ExtractILFromPkt(RSSI[],Thd,ilW)
2:      /*Get the threshold for determining ZigBee signal and
    burstiness.*/
3:      ZThd = Thd.GetInterfZThd()
4:      BThd = Thd.GetInterfBThd()
5:      BurstLenS = 0
6:      PAPR = CalcPAPR(RSSI[])
7:      if PAPR ≤ ZThd then
8:          IL = 1
9:      else
10:         /*Calculate interference level in current packet.*/
11:         S,E = GetBurstySeg(RSSI[],BThd)
12:         BurstC = len(S)
13:         for i : 0 ≤ i < BurstC do
14:             BurstLenS += (S[i] - E[i])
15:         burstyL = BurstLenS/BurstC
16:         IL = (Norm(PAPR) + Norm(burstyL))/2
17:     /*Calculate EMA of interference level.*/
18:     if len(ilW) > 0 then
19:         multiply = 2/(Wsize + 1)
20:         preEMA = ilW[len(ilW)-1]
21:         return (IL - preEMA)*multiply + preEMA
22:     return IL
```

Algorithm 1 shows the procedure to compute the interference level. There is no WiFi interference when PAPR is lower than the ZigBee threshold ZThd (i.e., 1.3). In this case, we need fewer pilot symbols and it is the same in the high interference scenario. Thus we set the interference level, `IL`, as 1. On the other hand, the `IL` of current packet is calculated as shown in Algorithm 1. If there is no `IL` in the interference level window (i.e., `ilW`), it means that our model has not been converged yet and we simply return the `IL` of current packet. Otherwise, we compute the exponential moving averages of `IL` within the interference level window to predict the interference level.

As shown in Algorithm 2, at the sender side, it extracts the interference level from the ACK. Then, the sender determines the fraction of the pilots according to the interference level. Initially, we instrument more pilot symbols in the first `IniPktC` packets to let the model converge quickly. In order to make sure that AccuEst converges quickly under different interference level, we set a conservative value as 8. The details of convergence time experiments are shown in Section V. We set the lower and upper bound of instrumenting pilot symbols as [5%, 50%] according to Section V-B. The function *GetPilotStep()* shows how to get the instrumented step of

Algorithm 2 Adaptive pilot instrumentation at sender

```
1: function SENDER(event,PBd,PktLen)
2:     Global int IniPktC
3:     if event == ACKTIMEOUT then
4:         return ACK timer times out
5:     if IniPktC ≥ 0 then
6:         Step = PktLen/(PktLen*PBd.up)
7:         IniPktC -= 1
8:     else if event == ACK then
9:         extract interference level IL from ACK.
10:        Step = GetPilotStep(IL,PBd,PktLen)
11:    NewPkt = InsertPilots(Step, NextPkt(event.id))
12:    SendPkt(NewPkt)
13:
14: function GETPILOTSTEP(IL,PBd,length)
15:    Fraction = α*(1-IL)*(PBd.up - PBd.down)+PBd.down
16:    return length/(length*Fraction)
```

Algorithm 3 Adaptive pilot instrumentation at receiver

```
1: function RECEIVER(RSSI[],pkt,ilW,PBd,Thd,HIL)
2:     Global int IniPktC
3:     if IniPktC ≥ 0 then
4:         Step = len(pkt)/(len(pkt)*PBd.up)
5:         IniPktC -= 1
6:     else
7:         /*Extract IL from previous stored HIL*/
8:         IL = HIL.ExtraceSILFromId(pkt.id)
9:         ilW.AddSIL(IL)
10:        Step = GetPilotStep(IL,PBd,len(pkt))
11:    extract the training set and update the model.
12:    List CorrupResult = ApplyModel(RSSI[])
13:    infer SER from the CorrupResult.
14:    CIL = ExtractILFromPkt(RSSI[],Thd,ilW)
15:    if CheckCRC(pkt) == success then
16:        HIL.RemoveILFromId(pkt.id)
17:        HIL.StoreIL(CIL,NextId(pkt.id))
18:        Reply(ACK,CIL)
19:    return CorrupResult
```

each pilot symbol. The number of pilot symbol is determined inversely by the interference level. Then, we compute the step of evenly instrumenting the pilots.

When a packet arrives at the receiver side, AccuEst would detect the corrupted bytes and calculate the interference level as shown in Algorithm 3. AccuEst stores a dictionary where the key is packet sequence number and the value is the interference level. In order to append the new pilot samples into the sliding window of training set, we first extract the interference level from the dictionary according to the received packet sequence number. Then the step of pilot symbols is calculated using the function *GetPilotStep()* in Algorithm 2. We update the corruption detection model as soon as the new pilot samples are appended to the sliding window of training set. Then, the results of the corruption detection are delivered to the upper layer protocol or error correction codes. The interference level of the received packet is computed as shown in Algorithm 1 and we store the next packet sequence number and the interference level into `HIL` for latter usage.

As a byproduct, we can utilize the number of detected corruption bytes to estimate the Symbol Error Rate (SER) in a packet. We will compare our approach with the traditional pilot-based approach with respect to the pilot overhead in Section V-D.

Fig. 4: 8x10 indoor testbed.

Table 1: The range and default value of parameters.

Parameters	Default value	Range	Meaning
PBd	N/A	[0,1]	The range of instrumented pilot fractions.
α	1	[0, 1]	The aggressive levels for instrumenting pilots.
q	0	[0, 8]	The variances from the (i-q)-th byte to the (i+q)-th byte.
IniPktC	1	[1, 20]	The initial number of packets to let model be converged quickly.

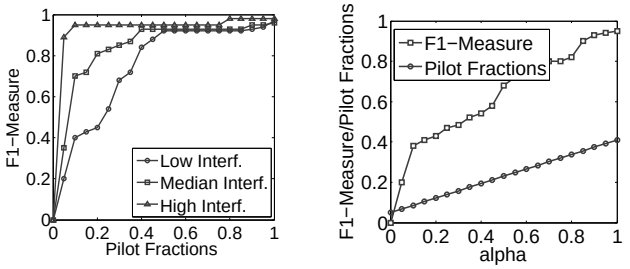
V. EVALUATION

In this section, we evaluate the performance of AccuEst, and compare AccuEst with the state-of-the-arts, i.e., CARE [3] and REPE [10] with respect to the corruption detection accuracy. In addition, we deploy AccuEst on an indoor testbed to further evaluate the system and discover its benefits to assist a coding-based transmission protocol (i.e., ACR [4]). The indoor testbed consists of 8x10 TelosB nodes (see Fig. 4). The distance between any two nodes is about 0.5m. The radio power of each node is set to -32.5 dBm, and this results in a 5-hop wireless sensor network. AccuEst is implemented on the TelosB platform running TinyOS 2.1.1. The code size is ∼8.9 KB in ROM, and ∼3.6 KB in RAM. Considering a TelosB node has a total of 48 KB ROM and 10 KB RAM, this overhead is acceptable.

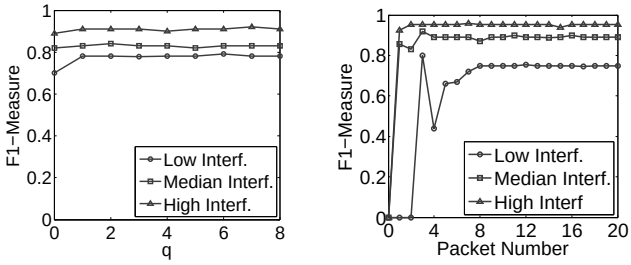
A. Experimental Methodology

We use two TelosB nodes running TinyOS 2.1.1 as a transceiver pair. They communicate with each other using the CC2420 radio and the power level is set to 5 at a distance of 1.5m. The sender node sends data packets with a payload of 97 bytes to the receiver node with an interval of 512ms. We use a laptop and a wireless access point running 802.11g protocol as an interference pair. To study the performance of AccuEst, we generate different interference levels (i.e., the strength and the bursty level of the interference). To achieve this, we select an overlapped channel from WiFi and CC2420 radio (i.e., channel 11 for WiFi and channel 21 for CC2420 radio), and generate three levels of the interference level, i.e., low, median, and high. We use iperf [21] on the laptop to generate different levels of the bursty (i.e., 3∼6M, 9∼12M, and 15∼18M TCP traffic, respectively). We vary the distance between the transceiver pair and the interference pair (i.e., 17m, 10m, and 3m, respectively) to generate different levels of WiFi signal strength.

The coding-based transmission protocol ACR [4] 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, to compare the end-to-end perfor-



(a) The range of instrumented pilot fractions. (b) The aggressive levels for instrumenting pilots.



(c) The number of close by SINR values. (d) The initial number of packets for quickly converging.

Fig. 5: Parameters evaluation.

mance including packet delivery rate and data latency. The experiments are done on our indoor testbed.

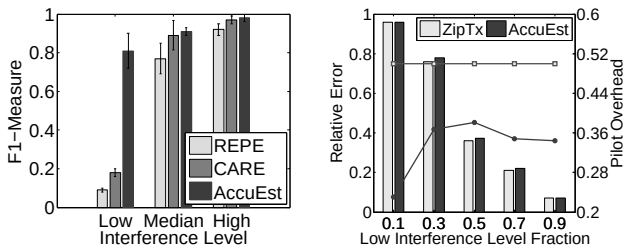
We use F1-Measure [22], the harmonic mean value of precision and recall, as our metric to measure the performance. Let TP_{pre} and TP_{recall} denote precision and recall of the true positive, respectively, F1-Measure is calculated by $\frac{2*TP_{pre}*TP_{recall}}{TP_{pre}+TP_{recall}}$:

B. Parameters Selection

Table 1 summarizes the range and the default values of the parameters mentioned in Section IV. We combine the use of pilot symbols with the accurate SINR indicator to improve the corruption detection accuracy. To strike a good balance between accuracy and overhead, we design an adaptive pilots instrumentation scheme which takes into account of interference level. We first instrument different fixed pilot fractions to evaluate the best range of the pilot fractions (i.e., the parameter `PBd`) for adaptive pilots instrumentation scheme. We then enable the adaptive pilots instrumentation component to evaluate the rest of parameters.

Figure 5-(a) shows the impact of instrumented pilot fractions on F1-Measure. The results show that to achieve F1-Measure which is higher than 80%, we need different least pilot fractions for different interference levels (i.e., 0.05, 0.18, and 0.36 pilot fractions for high interference, median interference and low interference, respectively). We also observe that the improvement becomes not significant (i.e., $\sim$5%) when the pilot fractions are larger than 0.5. Hence, `PBd` is set to [0.05, 0.5].

We define a parameter α for users to adjust the bias to low pilot overhead or high corruption detection accuracy. Figure 5-(b) shows the impact of aggressive levels α on F1-Measure and the pilot overhead. The result shows that by setting the aggressive level α to 1, F1-Measure achieves more than 90% on average with the relative high pilot overhead (i.e., 40%). In practice, to meet the basic requirement of the corruption



(a) Corruption detection accuracy. (b) Impact of dynamic interference.

Fig. 6: Performance comparison with REPE, CARE and ZipTx.

detection accuracy, we set α to 0.6 to achieve F1-Measure which is more than 80% with the relative low pilot overhead (i.e., 23%).

Due to the deviation of RSSI sampling, samples are not perfectly aligned with each byte, thus we calculate the SINR of the $(i-q)$-th byte to the $(i+q)$-th byte as the feature of the i-th byte. Figure 5-(c) shows the impact of parameter q on the F1-Measure. We observe that the improvement of F1-Measure is small (i.e., $\sim$2%) when q is larger than 1. We hence set q to 1.

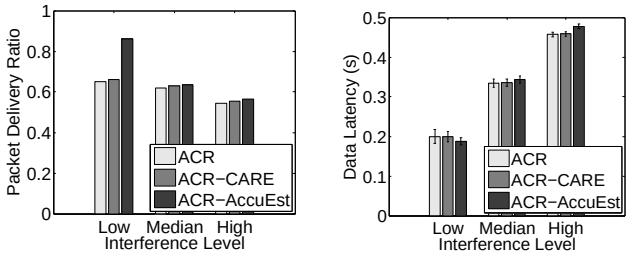
Figure 5-(d) shows the initial number of packets for converging. In the high interference scenario, our approach converges quickly after 2 packets. In both median interference and low interference scenarios, 4 and 8 packets are needed for converging. In practice, we set the initial number of packets `IniPktC` to 8 to ensure our approach quickly converges under any scenario.

C. Corruption Detection Accuracy

We compare AccuEst with REPE and CARE under different interference levels. In fact, CARE is a retransmission protocol and we only compare AccuEst with the corruption estimation component of CARE. The result in Fig. 6-(a) shows that AccuEst achieves the best performance among threes. Sepcifically, AccuEst improves F1-Measure significantly by 72% and 63% on average compared to REPE and CARE, respectively, under low interference.

D. Impact of Dynamic Interference Levels

In this experiment, we evaluate our adaptive pilots instrumentation component under a dynamic interference experiment. We vary interference between the high interference level and the low interference level. The low interference level fraction is computed as the ratio of the low interference time duration and the total duration (i.e., 100 minutes in our evaluation). As we shown in Section IV, we can utilize the number of detected corruption bytes to estimate the symbol error rate in a packet. We compare our approach with the pilot-based approach, i.e., ZipTx [6], with respect to the relative error of estimating symbol error rate. We let the ZipTx instrument the pilot symbols, instead of bits, into the packet. We modify the aggressive level α to achieve an approximately equivalent relative error to ZipTx. We conduct each experiment 10 times and show the average results in Fig. 6-(b). The result shows that our approach significantly reduces the pilot overhead by 16.1% on average while achieving the approximately equivalent relative error compared to ZipTx.



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

Fig. 7: End-to-end performance comparison

E. Testbed Experiments

ACR actively converts most potential collisions into the long and short packet collision patterns (called LS-collisions) to enable a lightweight FEC scheme to recover collided packets. An ACR packet consists of 10 blocks where the first 6 blocks are data and the rest are redundancy by default. The block size is 10 bytes. When the number of erroneous blocks is larger than 4, 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. The block is regarded as corrupted when the difference between $RSSI[i]$ and $RSSI_{base}$ is larger than 5. We replace the corruption detection component to compare the end-to-end performance including packet delivery ratio and data latency. The block is corrupted when there is at least one erroneous byte.

Packet delivery ratio. Figure 7-(a) shows the packet delivery ratio (PDR) when we apply different corruption detection approaches to estimate the number of erroneous blocks. The result shows that ACR-AccuEst improves PDR by 18.1% and 17.3% on average compared to ACR and ACR-CARE, respectively. We observe that the PDR of ACR and ACR-CARE is relatively low under low interference because the erroneous bytes are sparsely spread in the whole packet and the number of corrupted blocks exceeds the redundancy easily. Both ACR and ACR-CARE cannot detect the erroneous blocks and lead to multiple retransmissions of the whole packet.

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 expectation 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 90-byte packet with a data rate of 250kbps on the CC2420 radio [10]. The major overhead in AccuEst falls into two factors – feature calculation and parameter updating. It typically costs 0.93ms to determine the corrupted bytes, and therefore, it is acceptable in terms of the MAC backoff time and packet transmission time. Fig. 7-(b) shows the data latency in a 5-hop network. Results show that ACR-AccuEst incurs relatively low data latency by 4.4% at most compared to ACR.

VI. Conclusion

Accurate corruption estimation is very important for improving the resilience of ZigBee transmissions. This paper reveals the limitations of prior in-packet RSSI approach and carefully calculates the accurate indicator SINR to detect the corruptions. We combine the link-layer information (pilot symbols) and the calculated SINR to further improve the corruption detection accuracy. Besides, we design an adaptive pilot instrumentation scheme to strike a good balance between the accuracy and overhead. We have extensively conducted the experiments including single-link and multi-hop to evaluate our approach. Testbed results show that our approach can significantly improve the packet delivery ratio while incurring acceptable data latency.

In the future, there are multiple directions to explore. First, we would like to develop a better way to instrument pilots for improving the efficiency of parameter updating. Second, we would like to design a more computation efficient algorithm to reduce the computation 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, pp. 309–322.
[2] K. Jamieson and H. Balakrishnan, "Ppr: Partial packet recovery for wireless networks," in *Proc. of ACM SIGCOMM*, 2007, pp. 409–420.
[3] 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, pp. 235–244.
[4] Y. Wu, G. Zhou, and J. A. Stankovic, "Acr: active collision recovery in dense wireless sensor networks," in *Proc. of IEEE INFOCOM*, 2010, pp. 1–9.
[5] S. Sen, N. Santhapuri, R. R. Choudhury, and S. Nelakuditi, "Accurate: Constellation based rate estimation in wireless networks," in *Proc. of USENIX NSDI*, 2010.
[6] C. J. Lin, N. Kushman, and D. Katabi, "Ziptx: Harnessing partial packets in 802.11 networks," in *Proc. of ACM MobiCom*, 2008, pp. 351–362.
[7] J. Huang, Y. Wang, and G. Xing, "Lead: Leveraging protocol signatures for improving wireless link performance," in *Proc. of ACM MobiSys*, 2013, pp. 333–346.
[8] 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, pp. 409–420.
[9] M. Vutukuru, H. Balakrishnan, and K. Jamieson, "Cross-layer wireless bit rate adaptation," in *Proc. of ACM SIGCOMM*, 2009, pp. 3–14.
[10] J.-H. Hauer, A. Willig, and A. Wolisz, "Mitigating the effects of rf interference through rssi-based error recovery," in *Proc. of EWSN*, 2010, pp. 224–239.
[11] 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, pp. 119–133.
[12] 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,To apear.
[13] A. Hithnawi, H. Shafagh, and S. Duquennoy, "Tiim: Technology-independent interference mitigation for low-power wireless networks," in *Proc. of ACM/IEEE IPSN*, 2015, pp. 1–12.
[14] 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, pp. 55–66.
[15] S. S. Hong and S. R. Katti, "Dof: A local wireless information plane," in *Proc. of SIGCOMM*, 2011, pp. 230–241.
[16] 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,To apear.
[17] 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, pp. 89–94.
[18] C. Datasheet, "http://focus.ti.com/lit/ds/symlink/cc2420.pdf," Texas Instruments, 2007.
[19] T. Liu and A. E. Cerpa, "Foresee (4c): Wireless link prediction using link features," in *Proc. of ACM/IEEE IPSN*, 2011, pp. 294–305.
[20] T. Liu and A. E. C., "Talent: temporal adaptive link estimator with no training," in *Proc. of ACM SenSys*, 2012, pp. 253–266.
[21] J. D. J. F. A. Tirumala, F. Qin and K. Gibbs, "Iperf: The tcp/udp bandwidth measurement tool," http://dast.nlanr.net/Projects, 2005.
[22] T. M. Mitchell, *Machine Learning*, 1st ed. McGraw-Hill, 1997.