Embracing Corruption Burstiness: Fast Error Recovery for ZigBee under Wi-Fi Interference

Zhiwei Zhao, *Member, IEEE,* Wei Dong, *Member, IEEE,* Gonglong Chen, *Student Member, IEEE,* Geyong Min, *Member, IEEE,* Tao Gu, *Senior Member, IEEE,* and Jiajun Bu, *Member, IEEE*

Abstract—The ZigBee communication can be easily and severely interfered by Wi-Fi traffic. Error recovery, as an important means for ZigBee to survive Wi-Fi interference, has been extensively studied in recent years. The existing works add upfront redundancy to in-packet blocks for recovering a certain number of random corruptions. Therefore the bursty nature of ZigBee in-packet corruptions under Wi-Fi interference is often considered harmful, since some blocks are full of errors which cannot be recovered and some blocks have no errors but still requiring redundancy. As a result, they often use interleaving to reshape the bursty errors, before applying complex FEC codes to recover the re-shaped random distributed errors. In this paper, we take a different view that burstiness may be helpful. With burstiness, the in-packet corruptions are often consecutive and the requirement for error recovery is reduced as "recovering any k *consecutive* errors" instead of "recovering any random k errors". This lowered requirement allows us to design far more efficient code than the existing FEC codes. Motivated by this implication, we exploit the corruption burstiness to design a simple yet effective error recovery code using XOR operations (called ZiXOR). ZiXOR uses XOR code and the delay is significantly reduced. More, ZiXOR uses RSSI-hinted approach to detect in packet corruptions without CRC, incurring almost no extra transmission overhead. The testbed evaluation results show that ZiXOR outperforms the state-of-the-art works in terms of the throughput (by 47%) and latency (by 22%).

Index Terms—ZigBee, Bursty corruptions, error recovery, XOR coding

1 Introduction

The ZigBee technology enables low power, reliable and scalable wireless communications for various energy-constrained devices. These devices further support emerging Internet-of-Things applications including smart home [1], health monitoring [2], emergency response [3], etc. Most of these applications are real-time, and they have stringent requirements on both throughput and delay. However, ZigBee operates in crowded unlicensed ISM band (e.g., Wi-Fi, Bluetooth, microwave oven), and the performance of ZigBee communications can be easily and severely interfered by Cross Technology Interference. Among these technologies, Wi-Fi is the most common interferer due to its pervasive deployment. Further, Wi-Fi's signal power (i.e., 20dBm usually) is far stronger than ZigBee (i.e., 0dBm at maximum), and hence it can easily interfere ZigBee communication even without sensing its existence [4]–[6].

The problem of ZigBee packet error recovery under Wi-Fi interference has recently attracted much research attention [7]–[13]. There are basically two kinds of approaches: partial retransmission (e.g., [7]–[9]), and forward error correction (FEC) (e.g., [10]–[12]). In the partial retransmission approach, a data payload is divided into several small blocks, which are then transmitted in one packet. In case of errors occurred, a receiver replies NAKs and the sender then retransmit only the erroneous blocks, rather than the entire packet as traditional ARQ scheme. However, the inter-packet transmission delay in this approach is still unavoidable due to the nature of retransmission. The FEC approach obviates retransmission by means of sending upfront error-correcting information, along with the original data. Before transmission, the packet payload is divided into small blocks which are further encoded using error-correcting codes (e.g., Reed-Solomon code). When certain levels of bit errors occur, the receiver can recover the original packet without retransmission.

Recent studies have observed that ZigBee corruptions under Wi-Fi interference are highly bursty [11], [14]–[16]. The burstiness leads to the case that some blocks may be full of bit errors which cannot be recovered by the FEC, while other blocks may have no bit errors at all but still require redundant bits. As a result, existing works consider burstiness harmful for error recovery. To deal with burstiness, most of these works first use interleaving to reshape the bursty errors before applying RS/BCH codes [10], [12]. Unfortunately, such methods may still lead to substantial reduction in throughput *due to the high decoding delay*. For example, RS(15,7) decoding consumes over 100ms on TelosB motes, and the block-level CRCs consume tens of bytes in the limited ZigBee packet length (127 octets). This further poses a significant challenge for applying the FEC coding scheme in real time ZigBee communications.

Different from the traditional point of view, we argue in this paper that burstiness can be indeed helpful for error recovery. Our key insight with burstiness is that the requirement for error recovery turns out to be recovering "any k *consecutive* blocks" instead of "any k blocks" since the block errors are most likely

- *Z. Zhao is with the College of Computer Science, Zhejiang University and the College of Computer Science and Engineering, University of Electronic Science and Technology of China, China. E-mail: zzw@uestc.edu.cn*
- *W. Dong, G. Chen and J. Bu are with the College of Computer Science, Zhejiang University, China. E-mail: {dongw, chengl, bjj}@zju.edu.cn* (Corresponding author: Wei Dong)
- *G. Min is the College of Engineering, Mathematics and Physical Sciences, University of Exeter, U.K. E-mail: g.min@exeter.ac.uk*
- *T. Gu is with School of Computer Science and IT, RMIT University, Australia. Email: tao.gu@rmit.edu.au*

consecutive. This lowered requirement motivates us to design a far more efficient FEC coding scheme for ZigBee under Wi-Fi interference.

Inspired by this implication, in this paper, we propose a novel forward error recovery scheme to improve *Zig*Bee communication performance under Wi-Fi interference with *XOR* (called ZiXOR). Different from the traditional FEC coding schemes which try to recover "any k blocks", ZiXOR uses a simple yet effective approach based on XOR operations to distribute *any k consecutive* erroneous blocks into *k* separate XORed redundant blocks. In this way, each redundant block contains only one erroneous block, and *any k consecutive* block errors can be recovered by simple XOR operations. The coding delay can be significantly reduced.

When under non-bursty scenarios (e.g., outdoor communication), ZiXOR may not work well. To deal with this problem, we propose an adaptive switch scheme to switch to fountain code mode when there are multiple error bursts within one packet. The switch is designed "seamingless", which means the previous transmitted ZiXOR blocks can be directly used as fountain encoded blocks for decoding. Therefore the performance of ZiXOR is similar with fountain code (e.g., DLT [17]) under non-bursty scenarios.

Despite the coding delay can be greatly reduced by leveraging the burstiness, the block-level CRC bytes could still greatly degrade the throughput given that the maximum ZigBee packet length is only 127 bytes, which is largely different from the 802.11 networks. Take RAT [12] as an example, when a packet payload is divided into 12 blocks, 24 CRC bytes will be required. In order to reduce the overhead, we sample fine-grained RSSI values during packet reception. The correlation between RSSI samples and byte errors allows us to detect block errors without CRCs, thus saving room in the payload for data transmissions.

We implement ZiXOR in a testbed with 8x10 TelosB nodes. We then incorporate it into an existing routing protocol CTP [18] and compare its performance with the state-of-the-art works [7], [11], [12], [17]. We conduct both trace-driven and testbed experiments, and the results demonstrate that ZiXOR provides real-time forward error recovery with nearly no overhead (less than 1ms delay). Compared with the existing works, ZiXOR greatly improves the end-to-end protocol performance in terms of throughput (47%), transmissions (37%) and latency performance (22%), respectively.

The major contributions of this paper are summarized as follows:

1) We identify the opportunity to leverage the bursty nature of ZigBee corruptions under Wi-Fi interference for error recovery.
2) We design ZiXOR, a fast forward error correction scheme for ZigBee communication under Wi-Fi interference. ZiXOR is lightweight in both transmission and coding overhead, which is highly applicable for resource-constrained ZigBee devices.
3) We implement ZiXOR on a real sensor testbed and conduct extensive evaluations. The results show that ZiXOR outperforms existing works in terms of both throughput and latency.

The remainder of this paper is organized as follows: Section 2 introduces related work and positions ZiXOR in the literature. Section 3 presents our motivation, observation and key idea. Section 4 presents the main design of ZiXOR. Section 5 presents the trace-driven study and testbed evaluation of ZiXOR. Section 6 concludes this work.

2 Related works

The problem of ZigBee error recovery under Wi-Fi interference has been extensively studied. Basically, all existing approaches can be classified into two categories: retransmission-based and FEC-based. Both categories divide a packet payload into several blocks. The difference is that the first category retransmits the erroneous native blocks when errors occur; the second category encodes the blocks by adding redundant bits before packet transmission and can recover the native packet without retransmission.

Partial packet retransmission. In this category of approaches, the sender retransmits only the erroneous blocks when errors occur. Existing works try to reduce the block CRC overhead in order to achieve better link performance.

Seda [7] adds a 1-byte sequence number and a 1-byte CRC to each block. Then the receiver can identify and request the erroneous blocks. Maranello [8] is a similar approach, but the difference is that Maranello does not transmit block CRC in the first round transmission. When errors occur, the receiver computes the block CRCs and reply them to the sender for retransmission. Such design incurs no block CRC when the packet is loss free. REPE [9] equips each low power node with a high resolution timer (i.e., 62.5 kHz) and periodically samples the RSSI values for each received symbol. Based on the RSSI series, the receiver then requests the detected erroneous symbols. Our work differs from REPE in two important ways. First, we do not require additional hardware support. Second, instead of using a threshold to detect errors, we use a probability-based approach, which achieves more accurate error estimation. Third, the coding scheme significantly reduces the retransmission rounds, resulting in higher channel utilization.

FEC-based Approach. FEC-based approaches add upfront redundancy to each block for forward recovery. Many works on the spatial-temporal wireless behaviors [19]–[22] have observed that ZigBee corruptions under Wi-Fi interference are often bursty. The burstiness is often considered harmful for FEC, because with bursty errors, it is most likely that some blocks are full of errors which cannot be recovered as the redundant bits are insufficient, while some other blocks have no errors and the redundant bits are wasted.

To deal with burstiness, existing works first use interleaving to spread the bursty errors, and then apply different types of codes. Since the errors are reshaped as random distributed, the FEC codes have to be able to recover "any random k errors", which further leads to complex coding designs. ZipTx [10] is originally proposed for 802.11 networks. It selectively uses Reed-Solomon (RS) code to recover block errors when error rate is low. TinyRS [11] is an implementation of RS code in TinyOS/TelosB platform. Although it can recover the errors in relatively high-error scenarios (e.g., severely interfered links), it introduces too much coding and transmission overhead (RS(15,7)'s decoding time is $\sim$100ms and two redundant bits are required to recover one bit error). To reduce the high coding delay of RS code, RAT [12] is proposed to selectively exploit BCH-code and hamming code according to the estimated channel conditions, which consume less decoding time than RS-code. However, considering the transmission time is 4ms, the decoding delay is still remarkably large (e.g., RS(15,7) takes $\sim$104ms for decoding in MSP430 platform, see Section 5).

TABLE 1
ZigBee error recovery approaches under Wi-Fi.

	Forward Correction	No ctrl bytes	Decoding Delay	No H/W modification
ZiXOR	✓	✓	Very low	✓
RAT [12]	✓	×	High	✓
BuzzBuzz [11]	✓	×	Very high	✓
ZipTx [10]	✓	×	Very high	✓
REPE [9]	×	✓	N/A	×
Maranello [8]	×	×	N/A	✓
Seda [7]	×	×	N/A	✓
DLT [17]	✓	×	Low	✓

Different from the above works, we consider the corruption burstiness as a chance for designing lightweight and effective FEC code. ZiXOR has two main differences from the above works. First, it disables the interleaving, and exploits the consecutiveness of the block errors to design highly lightweight code using only XOR operations. Second, ZiXOR exploits fine-grained RSSI to detect block errors, avoiding the extra bytes for block checksums.

DLT [17] is the state-of-the-art work that exploits subtly optimized fountain code for in-packet error recovery. The sender continuously transmits encoded blocks. The receiver recovers the native packet when sufficient linear independent blocks are received. While DLT is applicable for general scenarios with random errors, ZiXOR is more suitable for bursty errors. First, DLT decoding still requires Gaussian elimination while ZiXOR requires only XOR operations. Second, DLT requires one byte CRC for each block, while ZiXOR identifies block errors without block CRCs. It is also worth noting that, we design an adaptive switch scheme with which ZiXOR can switch to fountain codes under random error patterns.

Table I compares ZiXOR with existing works in respect to the following key desired features.

- **Forward Error Correction.** The ability to correct errors in advance, which is important for reducing inter-transmission delay.
- **Control overhead.** The extra control bytes introduced to the packet payload (e.g., block CRC, sequence number, etc.). It has a significant impact on the channel utilization of resource constrained ZigBee devices.
- **Decoding delay.** Decoding delay has a significant impact on link throughput.
- **Incremental retransmission.** When a packet transmission fails, the already received correct data is still effective for further decoding.
- **No H/W Support.** This is important for practical applications of the approach on existing devices.

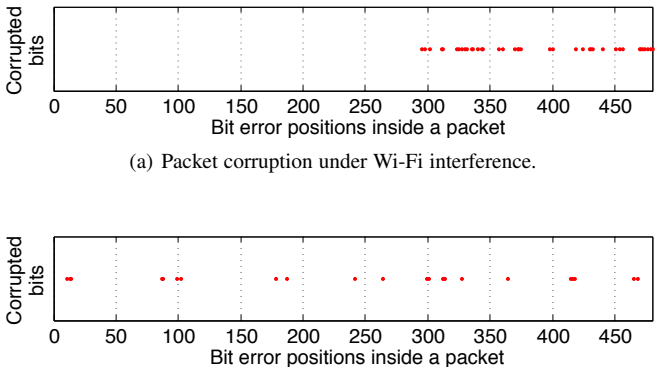
We can see that none of the existing works meet all these features simultaneously, which motivates our work.

3 Motivation and Key Idea

In this section, we first describe the empirical observations and motives, then we present the key idea using a simple example.

3.1 Empirical Observations and Motivation

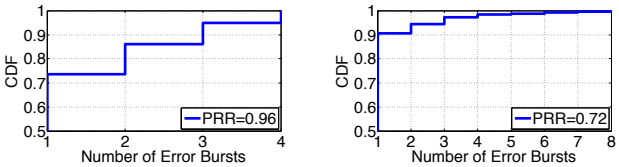
Error distribution under Wi-Fi interference. Recent studies have shown that Wi-Fi traffic is usually bursty and clustered [14], [15], [23], implying that corruptions of ZigBee packets are also expected to be bursty and clustered. We first conduct an experiment to study the corruption patterns of in-packet ZigBee packets under Wi-Fi interference. We use two TelosB nodes (one sender and one receiver) to form a link. The sender keeps transmitting full-payload packets to the receiver. To simulate Wi-Fi interference, a physically nearby laptop is operated to surf the Internet (including web browsing, data downloading and video streaming) through W-Fi connection. We turn off the CC2420 hardware checksum such that the receiver can receive corrupted packets. By comparing the received data and the original data, we can know the error positions in corrupted packets.



(a) Packet corruption under Wi-Fi interference.

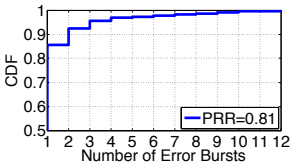
(b) Packet corruption with weak link.

Fig. 1. Typical bit error patterns in packets under Wi-Fi interference and with weak link.

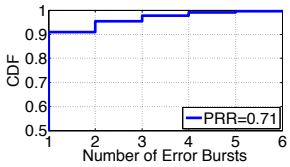


(a) Wi-Fi off

(b) Video streaming



(c) Web browsing



(d) Mixed

Fig. 2. Corruption patterns under Wi-Fi interference.

Figure 1 shows the typical error patterns with and without Wi-Fi interference. We can see that when under Wi-Fi interference, the errors are highly bursty. Figure 2 shows the statistical characteristics of in-packet corruptions under different typical Wi-Fi interferences. The blue lines denote the cumulative distribution function (CDF) of the number of erroneous bytes bursts[1]. We can see that compared with non-interference scenario, most corrupted packets under Wi-Fi interference have only one single error burst (consisting one or several consecutive block errors).

Existing FEC coding designs often consider burstiness harmful [10]–[12] and use interleaving to spread the errors. The coding approaches are often designed to recover *any k random* errors.

1. Similar with [16], [24], we allow up to one correct byte inside a corruption burst

Due to this requirement, existing works choose to use RS/BCH codes instead of XOR code, despite XOR code is much more lightweight.

However, different from the traditional views, we take the burstiness as a chance to reduce the coding requirement. With burstiness, the coding requirement could be reduced as "recovering *any k consecutive* errors" due to the block error consecutiveness. This offers a chance for reducing the coding complexity.

3.2 Challenges

XOR is promising for designing such code due to its fast operation and 1× recovery overhead (one redundant block recovers one erroneous block). For each packet transmission, XORed redundant blocks can be added, such that the receiver can recover the native packet by XORing the correct native blocks and redundant blocks. However, there are two significant challenges for the XOR-based framework.

Stringent coding requirement. One redundant block combines multiple native blocks. The key to successful recovery is to ensure that *no more than one combined native block is corrupted in the transmission*. Otherwise, the packet may not be recovered or the recovery will require Gaussian elimination (like fountain code). However, the above requirement is very challenging because we can never know which blocks will be corrupted before the actual transmission. This is the main reason why the existing approaches gave up the XOR-based framework, despite XOR is lightweight in both coding delay and transmission.

Block-level checksum and limited packet length. Like many existing works, ZiXOR divides the packet payload into several blocks. In order to help receivers identify which blocks are erroneous, two kinds of information are essential for decoding: 1) block level CRC bytes. 2) block length and number of blocks information. Considering that the maximum ZigBee packet length is only 127 bytes, the above two additional fields will still greatly degrade the channel utilization.

3.3 The Key Idea

Modulo-k XOR coding. The key idea is to exploit the burstiness to isolate the erroneous blocks. Although it is impossible to know which blocks are erroneous before the actual transmission, however, based on the bursty nature of ZigBee packet corruptions under Wi-Fi interference, we can know that the erroneous blocks are most likely consecutive. This allows us to effectively distribute any k consecutive erroneous blocks into k redundant blocks without knowing the exact block errors, ensuring that each redundant block covers only one block error.

The idea works as follows: Suppose there are k erroneous blocks, we encode each redundant block by combining one native block in every k native blocks. As the k blocks are consecutively distributed, these erroneous blocks can be encoded into different redundant blocks. Then *each redundant block combines only one corrupted block, and the corrupted block can thus be decoded*. In this way, we can recover any k consecutive erroneous blocks by simple XOR operations.

Figure 3 shows an illustrative example in which 7 native blocks are to be transmitted. The gray rectangle denotes the Wi-Fi interfered interval. Three redundant blocks are needed for recovering the three corrupted blocks. We pick one block every three blocks for encoding redundant blocks, as shown in

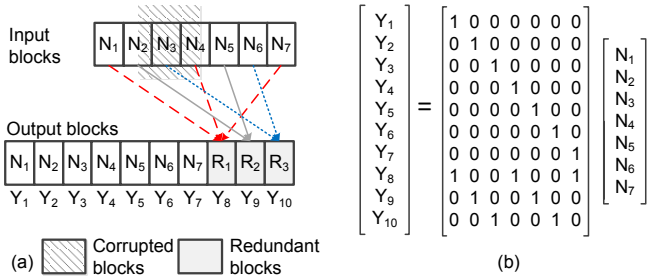


Fig. 3. ZiXOR encoding: an example.

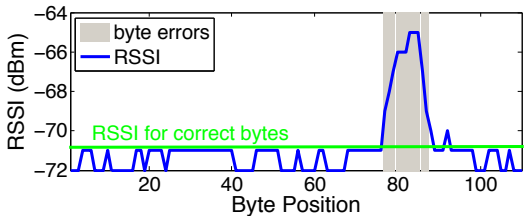


Fig. 4. Typical RSSI and block error patterns.

Figure 3(a) (R_1=N_1⊕N_4⊕N_7, R_2=N_2⊕N_5, R_3=N_3⊕N_6). Figure 3(b) shows the coefficient matrix for encoding. We can see that, every three consecutive blocks are encoded to different redundant blocks, such that the receiver can decode any error burst within three erroneous blocks.

Exploiting RSSI for block error identification. Recent study [16] shows that the in-packet RSSI values are highly correlated with byte errors. We record the in-packet RSSI values and the corrupted positions to study the correlation. Figure 4 shows a typical packet transmission and the received byte-level RSSI. We can see that at the corrupted positions, the measured RSSI samples are higher than those of correct bytes.

This phenomenon can be explained as follows. Generally, the bit error rate is determined by SNR (signal-to-noise ratio). It could be assumed that the transmission signal during one packet will not severely change. Therefore, the sudden RSSI variation is more likely to be caused by noise and interference, and SINR will decrease in this case. This explains why byte errors are along with RSSI rises.

This observation allows us to exploit RSSI readings for block error detection, thus reducing the block-level CRC overhead. When RSSI rises are detected, we can know that it is likely that the corresponding bytes are corrupted.

4 ZiXOR Design

We incorporate the above ideas into a novel XOR coding framework, ZiXOR. With corruption burstiness, ZiXOR can efficiently distribute the errors into different redundant XORed blocks. Using the RSSI patterns, ZiXOR can detect block errors without CRCs.

4.1 Overview

Figure 5 shows the system framework of ZiXOR. A sender node first estimates the number of redundant blocks (Section 4.2), and then encodes the redundant blocks with the ZiXOR.encode procedure (Section 4.3). After that, the native payload (with

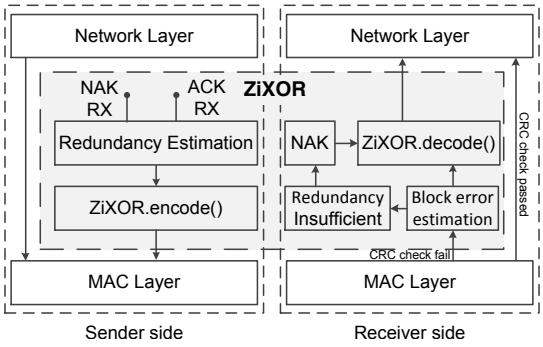


Fig. 5. ZiXOR overview.

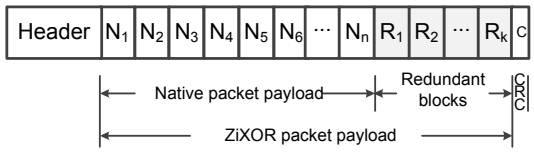


Fig. 6. ZiXOR packet format.

CRC) and the redundant blocks are combined into one packet for transmission, as depicted in Figure 6. The block size is stored in the 5 reserved bits in the header, and the number of redundant blocks is calculated at both sender and receiver side (Section 4.2). The CRC is the checksum calculated with the original data payload, with which the receiver is able to check whether the decoded packet is correct. It is worth noting that the native packet payload is divided as blocks but not encoded.

When receiving the packet and the CRC check is not passed, the receiver first estimates the block errors (described in Section 4.4). If the receiver finds that the received blocks cannot be decoded, the receiver directly transmits an NAK indicating the erroneous blocks. After that, the receiver still puts the received blocks into ZiXOR.decode in case that there may be false negative (FN) estimation results. Otherwise, if the blocks are identified decodable, the receiver directly decodes the received blocks. When decoding fails, retransmission starts (Section 4.5). The sender extracts the block error information when receiving the NAK, and encodes the new blocks for the retransmission. The receiver then decodes for the native packet when receiving the retransmissions.

4.2 Redundancy Estimation

When a sender prepares to transmit a packet, it should first decide how many redundant blocks should be added. We estimate the number of redundant blocks according to the block error rate collected in previously transmitted packets.

At the end of each transmission (either success or failure), the receiver replies an ACK/NAK to the sender (as depicted in Figure 5) which contains a bitmap indicating the block errors of the last packet transmission. For example, a bitmap of "0011000000" means that the third and fourth blocks are erroneous in the last packet. With this bitmap, the sender can calculate the fraction of "1"s as block error rate, p_e. Like many link estimation approaches [25], we apply moving average using multiple history packets to calculate p_e. Suppose there are n native blocks and we add x redundant blocks to ensure the receiver correctly receives n blocks, we can get the following equation.

$$(n+x)(1-p_e) = n \tag{1}$$

Solving the above equation, we can obtain the number of redundant blocks as:

$$x = \frac{np_e}{1-p_e} \tag{2}$$

Discussion on redundancy estimation errors. The issue of how to add appropriate redundancy is critical to all FEC approaches. When redundancy is over-estimated (i.e., excessive redundancy), both ZiXOR and other approaches will have unnecessary redundancy transmission overhead. Fortunately, ZiXOR's coding delay remains much smaller than other approaches since only several extra XOR operations are added. When redundancy is under-estimated, however, the FEC based approach will have to reassemble a new packet because the previous transmissions have no enough redundancy for decoding. Although in ZiXOR, previously transmitted redundant blocks are still useful for future decoding, the inter-packet interval may increase the overall transmission delay. At least ∼4.9ms will be incurred into the overall delay. Therefore we consider over-estimation less harmful than under-estimation, and deliberately over-estimate the redundancy by one. We can also employ various machine learning algorithms [25]–[28] for redundancy estimation if they can be optimized lightweight for low power devices. The redundancy estimation is evaluated in Section 5.

4.3 ZiXOR Coding

Encoding. For an estimated error burst of k blocks, we encode the redundant blocks as follows. We select 1 block in every k blocks, say m, $m+k$, $m+2k$, ... Then, we tune the starting offset m to obtain k different redundant blocks. With such encoding, any k (or $\leq k$) consecutive block errors can be separately covered by the redundant blocks. The ZiXOR encoding is formulated as follows.

$$R_i = \bigoplus_{k\%S_e=i} b_k, k \in [0, N_b - 1] \tag{3}$$

where R_i denotes the ith redundant block, $\oplus$ denotes the XOR operation, S_e denotes the size of error burst (i.e., the number of consecutive erroneous blocks) and N_b denotes the total number of native blocks. By such design, any k (or $\leq$k) consecutive block errors can be recovered because each redundant block would cover one different erroneous block. Moreover, different from fountain codes and random linear codes, the encoding rule is shared by both senders and receivers, and does not rely on any random seed or explicit coefficients. When receiving a ZiXOR packet, a receiver can identify which blocks are combined by each redundant block, using only the number of redundant blocks. For example, if a receiver receives a packet containing k redundant blocks, it can infer the coefficients using Eq. (3) with k and further decode the native packet.

Compared with the retransmission based approaches, ZiXOR adds k redundant blocks *in advance* to recover k consecutive erroneous blocks. This greatly reduces retransmission delay. Compared with the FEC based approach, ZiXOR 1) is quite lightweight in decoding, and 2) requires 1× redundancy to recover 1× errors while most FEC based approaches require 2× redundancy.

Decoding. In this section, we formally give the algorithm for ZiXOR decoding, as described in Algorithm 1. Decoding is called

Algorithm 1: ZiXOR.decode

Input : received blocks (rxBlocks) and the number of redundant blocks (rbn)
Output: native blocks

```
1  decodable = TRUE;
2  buffer[][],err[][];
3  blks = sizeof(rxBlocks);
4  for i : 0 ≤ i < blks do
5      if !est(rxBlocks[i] then
6          push rxBlocks[i].blockNumber,err[i % rbn];
7          if sizeof(err[i % rbn]) >1 then
8              decodable = FALSE;
9      else
10         push rxBlocks[i].blockNumber,buffer[i % rbn];
11 if decodable then
12     temp = {0x00} ;
13     for i : 0 ≤ i < sizeof(err) do
14         for j : 0 ≤ j < sizeof(buffer[i%rbn]) do
15             temp ⊕= buffer[ i % rbn ][j];
16         rxBlocks[err[i][0]] = temp;
17     if CRC_check(rxBlocks) then
18         flash.write(rxBlocks);
19         composeACK();
20     else
21         composeNAK(blockCrc(rxBlocks));
22 else
23     composeNAK(blockCrc(rxBlock));
```

when the packet CRC is not passed. The receiver first identifies the decodability of the received packet. If one redundant block covers more than one block errors, the packet will not be decodable. When decodable, the receiver can just recover the erroneous block by XORing its corresponding redundant block and other native blocks (with the same MOD k remainder). When decoding is done, we compare the payload CRC to check whether the decoded packet is correct. Since each encoded block can be identified by its position and redundant block number, our decoding does not need Gaussian elimination and is thus highly computation efficient.

4.4 Block Error Estimation

In order to save the space for block transmission, we use the fine-grained in-packet RSSI sampling (IRS) [9] to identify the block errors. The estimation results further give block error rate and block error bitmaps, which are of significance for coding efficiency.

In-packet RSSI sampling. In order to exploit the correlation between RSSI and byte errors, we should first measure RSSI samples while receiving a packet. By using a 32.5kHz timer in IRS (supported by most low power platforms), we are able to obtain one RSSI value in *per byte* granularity [9], [16], enabling the identification of byte-level errors. We implement such a high-resolution sampling procedure (i.e., at least one sample per byte) without extra hardware. We modify the existing radio driver in TinyOS 2.1.2 to support sampling at a rate of *one sample per byte*. This is different from REPE [9], which requires additional 62.5kHz timer hardware.

Error detection. With byte level RSSIs, the next step is to identify block errors using these values. We use the smallest RSSI value as the RSSI base, and study the relationship between byte error rate and the RSSI distance from the RSSI base. Figure 7 shows the empirical results of byte error probabilities with different RSSI distances. We can see that the error rate increases when RSSI distance becomes larger. Using this table, we are able to estimate the byte error rate using the RSSI distances.

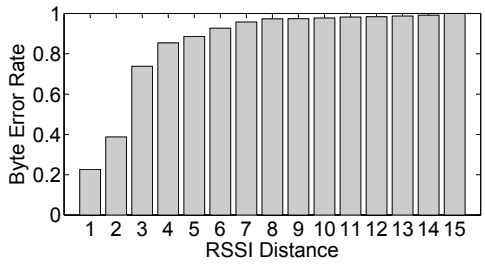


Fig. 7. The relationship between byte error rate and RSSI distance.

To estimate whether a block is erroneous or not, we sum up all the expected byte error rates within the block. If the sum exceeds 1, the block is expected to contain at least one byte error and is judged erroneous. We can see that compared with byte error detection, block error detection is more accurate. The reason is that a block error can be dismissed only when all byte errors are not detected, of which the probability is much less. The relationship between RSSI distance and BER could be learned from the several pilot packets, of which the data payload is shared by the sender and receiver, such that the receiver is able to measure the real BER.

REPE [9] also uses IRS for block error detection. In REPE, a byte is considered corrupted when the RSSI exceeds a certain threshold. Our scheme has two major differences with REPE. First, our goal is to identify block errors. We do not require strict position correspondence between the RSSI values and byte positions, thus is more tolerant to the RSSI and byte positions offset. Second, we use the corresponding error rates of certain RSSI values, instead of the threshold-based detection. This is expected to achieve the better block error detection accuracy.

Some works also exploit error correction codes for error detection and correction [29]–[31]. We compare error detection using RSSI samples and error correction codes, e.g., Hamming codes as follows.

1) Both error detection codes and byte-level RSSI samples can be used to detect in-packet errors.
2) Error detection codes are more reliable than the RSSI-based detection since it is based on the probabilistic relationship between RSSI and the byte errors.
3) RSSI-based detection can identify the positions of the byte errors while error detection codes can only detect the number of byte errors. This additional information allows us to design far more efficient error recovery codes such as ZiXOR.

We will evaluate the detection scheme in Section 5.

Discussion. *False negatives (FNs)*. An FN indicates that a correct block is estimated erroneous. Obviously, our estimation scheme is more prone to FNs. When FNs happen, a receiver may find there are more block errors than redundant blocks, and a decodable packet may be judged undecodable, incurring an unnecessary retransmission. To deal with this problem, we need to

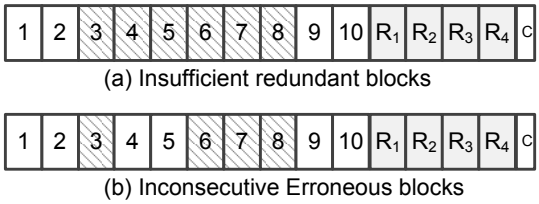


Fig. 8. Illustration of decoding failure at the receiver.

ignore the FNs and proceed to decode. When a receiver identifies that a packet is not decodable, it replies an NAK and proceeds to decode in case that FNs occur. If the packet CRC check is passed after the decoding, it can be inferred that the ignored blocks are FNs. Otherwise, the node waits for the sender's retransmission (Sec. 4.5).

False positives (FPs). An FP indicates that an erroneous block is estimated correct. FPs are harmful because the receiver cannot identify which blocks are incorrect when the packet CRC check is not passed. In case that FPs occur, retransmission is required (Sec. 4.5).

From the above analysis, we can see that FNs incur much less extra overhead than FPs. Fortunately, as we sum up all byte error probabilities for block error detection, most detection errors are FNs. We will empirically study the accuracy of IRS based block error estimation in Section 5.

4.5 Retransmission

ZiXOR adds redundant blocks based on the burst length estimation using historical data. Although the accuracy is high under Wi-Fi interference, there are still two cases in which the first round transmission fails, where retransmission is needed. First, the number of erroneous blocks is under-estimated[2] (as shown in Figure 8(a)). R1 covers 1/5/9, R2 covers 2/6/10, R3 covers 3/7, R4 covers 4/8. We can see that blocks 5 and 6 can be recovered while 3/4/7/8 cannot be recovered. Second, the erroneous blocks are not consecutive. We observe in Figure 2 that there are small portions of inconsecutive errors. As shown in Figure 8(b), although there are 4 block errors, they are not consecutive. R1 covers 1/5/9, R2 covers 2/6/10, R3 covers 3/7 and R4 covers 4/8. In this case, R1 is useless for error recovery.

When retransmission is required, it is likely that FPs or FNs of block error detection happen. Therefore, we should first confirm the real erroneous blocks. Then we find out which blocks are essential for retransmission recovery. Finally, we retransmit the necessary blocks using ZiXOR code.

Identifying erroneous blocks. We adopt a mechanism similar to [8]. When a receiver estimates all blocks are correct but the CRC does not match, it calculates the CRCs for each block, and replies the CRCs to the sender in an NAK message. The sender compares the block level CRCs to identify incorrect blocks at the receiver side. Then, the sender composes a retransmission packet by combining all incorrect blocks and redundant blocks.

Identify bottleneck blocks. Now that we obtain the erroneous blocks. However, not every erroneous block is required to be retransmitted, since some of them may already be covered by the redundant blocks in the last round transmission. Hence we should identify which erroneous blocks cannot be decoded and are necessary for further decoding, denoted as bottleneck blocks.

2. The decoding will be successful when over-estimated

Algorithm 2: ZiXOR.retransmit

Input : ErroneousBlocks(errorneousBlocks) and redundant block number(rbn) for current round transmission
Output: Packet payload for retransmission

```
1  blks = sizeof(rxBlocks);
2  for i : 0 ≤ i < blks do
3      blkNo = erroneousBlocks[i].blockNumber;
4      push blkNo , arrary[blkNo % n];
5  for i : 0 ≤ i < n do
       /* identify and record the encoding
          offset of k erroneous blocks        */
6      if sizeof(array[i])>1 then
           /* bottle-neck blocks detected     */
7          for j : 0 ≤ j < sizeof(array[i]) − 1 do
               /* prepare the blocks to
                  retransmit natively          */
8              push b_array[i][j] , retransmitBlocks ;
9      RSSI_base = min_{0≤i<L} RSSI[i];
10 if rbn>1 then
11     ZiXOR.encode(retransmitBlocks , rbn);
```

The key insight is that with ZiXOR encoding, each block is supposed to be used for encoding *only once*. Therefore, to find bottleneck blocks, we can simply calculate how many *erroneous* blocks are used for encoding each redundant block. If k erroneous blocks ($k>1$) are encoded into one redundant block, the first k-1 blocks are identified as bottleneck blocks. The reason is that each block is encoded only once and other blocks contain no information about these k blocks.

After obtaining bottleneck blocks, we treat these blocks as new blocks to send, i.e., transmit these native blocks and encoded redundant blocks using ZiXOR encoding (if redundant blocks are required by the redundancy estimation module).

We now revisit the examples shown in Figure 8. For the example in Figure 8(a): Though decoding fails, we still have the information of 3⊕7 and 4⊕8. Consequently, if 3 or 4 is obtained, 7 or 8 can be recovered and vice versa. To this end, we retransmit blocks 3 and 4. Considering the block error rate is 0.6, we expect one block error in the two retransmission blocks and add one more redundant block. This block is encoded by 3⊕4. For the example in Figure 8(b), there are multiple bursts and ZiXOR redundant blocks carries less information. We solve the problem of multiple bursts with the coding switch scheme as in the next sub-section.

4.6 Seamingless code switching for non-bursty error patterns

From Figure 2, we can see that there can be multiple bursts within one packet. In such scenario, ZiXOR may no longer be effective. To deal with this problem, we adaptively switch ZiXOR to fountain code when there are multiple bursts, in a "seamingless" manner as follows.

Burstiness estimation. We first use the short-term statistics to estimate the burstiness in the transmissions and then switch coding strategy between ZiXOR and fountain code accordingly. Specifically, we estimate whether the packets contain single bursts or multiple bursts. We first obtain a series of single burst probability using the windowed history packet trace (with window

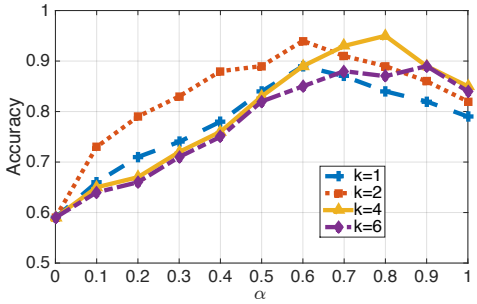


Fig. 9. Estimation accuracy with moving average.

size, k). Then we can obtain the probability that the next packet contain single-burst as:

$$p_{new} = \alpha p_{last} + (1-\alpha) p_{history} \tag{4}$$

where p_{last} denotes the single burst probability of the last window and $p_{history}$ denotes the long-term single-burst probability.

We study the estimation accuracy by tuning the window size k and the weighting factor α. Figure 9 shows the results. We can see that in our experimental settings, $k = 4$ and $\alpha = 0.8$ achieves the highest accuracy (92%). For practical use in different scenarios, we can periodically tune α and k and find the values that achieve the highest accuracy using the continuously collected data trace.

It is also worth noting that even the bursts are incorrectly estimated, ZiXOR is still able to switch to fountain code in time when it detects there are multiple bursts in the receiving packet.

Switching. While receiving a packet, a receiver accounts the number of error bursts using the fine-grained RSSI samples (similar with Sec. 4.4) and estimate the burstiness using the above moving average scheme. If there are multiple bursts which means ZiXOR.decode might fail, it turns into fountain decoding mode and invokes the Accumulative Gaussian Elimination [17] module. Then it notifies the sender to switch to fountain encoding. We call it "seamingless switch" because the received ZiXOR blocks can be directly fed into fountain decoders. This is practical because ZiXOR encoded blocks are specialized combinations of native blocks, which is essentially compatible with fountain code. When the receiver detects there are long single bursts in the received packets, it notifies the sender to switch back to ZiXOR mode for more efficient transmission.

4.7 System optimization

Block Size. Intuitively when block size increases, (1) for block error estimation, the number of FPs decreases and the number of FNs increases. (2) the redundant bits are likely to increase. (3) decoding delay is likely to decrease. Therefore, deciding the block size is to find a good tradeoff between the redundancy and coding efficiency. We use the expected goodput as the end-to-end metric for block size optimization.

We denote block size as s_b, then the probability of FPs can be denoted as $E_{fp}(s_b)$, redundant bits can be denoted as $R(s_b)$, and decoding delay can be calculated as $D_{decode}(s_b)$.

Then we can model the throughput using the variable s_b as follows. The throughput is calculated as:

$$T = \frac{S}{D} \tag{5}$$

where S denotes the useful transmission bits (without redundancy) and D denotes the transmission time. S is calculated as:

$$S = N_{header} + N_{pkt} + N_{crc} \tag{6}$$

where N_{header} is the packet header (12 bytes) and N_{crc} is the packet footer (2 bytes checksum).

D is calculated as:

$$D = (D_{backoff} + \frac{S + R(s_b)}{R_b} + D_{decode}(s_b))(1 + E_{fp}(s_b)) \tag{7}$$

where $D_{backoff}$ is the backoff time (random between 0∼9.8ms, 4.9ms expected), R is the redundancy, R_b denotes the transmission bitrate, and $E_{fp}(s_b)$ is the false positive rate with s_b. Then, we can obtain the optimal block size by maximizing the throughput T according to Eq. (5)∼(7).

Coding information. In the design of ZiXOR, although the receiver does not need the random seed for decoding, it should be aware of the number of redundant blocks and block size, such that it can obtain the encoding vectors. The number of redundant blocks can be simultaneously calculated at both sender and receiver sides (Section 4.2). Therefore, only block size is required to be transmitted to the receiver. To avoid extra transmission overhead, we use the 5 reserve bits in the Frame Control Field (FCF) in the packet header to store the block size. With the block size and redundant block number, the receiver can figure out which blocks are used for encoding certain redundant blocks, and further decode the native packet.

Overall, ZiXOR does not introduce any extra bits as compared with original ARQ.

Link selection. Since ZiXOR adds upfront redundant blocks in packet transmissions, the metrics such as packet reception rate (PRR) and expected number of transmissions (ETX) may no longer accurately represent the transmission efficiency.

1) Link selection metrics. Similar with PRR, we can evaluate the link efficiency using the data delivery rate (DDR). DDR is calculated as $DDR = \frac{D}{T} = \overline{n/n+k}$, where D is the effective data delivery, T is the amount of transmitted data, n is the number of native blocks and k is the number of redundant blocks. Similar with ETX, we can evaluate the transmission overhead using the expected transmission for one byte data delivery (ETD) as $ETD = \frac{T}{D} = \overline{n/n+k}$. The metric ETD can be accumulated along multi-hop paths.
2) ZiXOR nodes reaction to the link selection process. Since beacons are too short for burstiness measurement, the nodes are unaware of the burstiness on corresponding links when they are not selected and activated by upper layer protocols. Therefore, when a link is newly selected, ZiXOR needs to determine two parameters for efficient transmission: the block size and the number of redundant blocks. These two parameters are determined according to Section 4.2 and Section 4.7, which requires an initial measurement on the error bursts. As a result, the parameters are first randomly set after the selection and then adjusted according to the measurement results. We discuss the impact as follows. a) When the random redundancy is insufficient for error recovery, more retransmissions will be incurred. b) When the redundancy is more than enough, extra redundancy will be added to the packets. In either case, the throughput will be low in

Fig. 10. The 8x10 TelosB motes testbed.

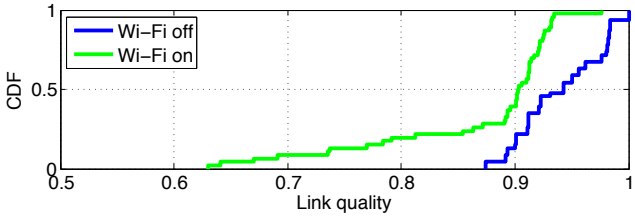


Fig. 11. The link quality of the testbed.

the beginning and then increases as the two parameters are adjusted according to the continuous measurement.

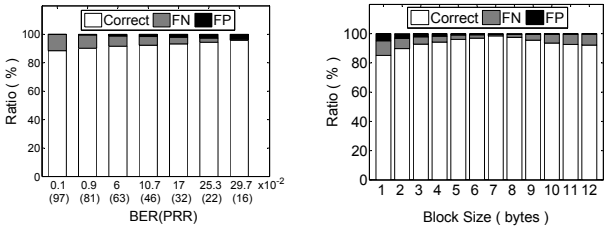
5 Evaluation

To evaluate ZiXOR, we first use trace-driven studies to discover optimized parameters used in ZiXOR, and then conduct testbed experiments to study the performance of ZiXOR. More specifically, we compare ZiXOR with the state-of-the-arts such as RS-code, Seda [9], DLT [17] and RAT [12].

5.1 Experimental Methodology

Implementation issues. We implement ZiXOR on TelosB nodes with TinyOS 2.1.2. The block size is set to 8 bytes for fair comparison with other approaches. The redundancy is calculated using Eq. (2), and the block error rate (BLER) is calculated using moving average. We vary the weighing parameter of the moving average, and choose the value of 0.8 because it achieves the most accurate BLER estimation in our experiments. Namely, intermediate BLER is weighed 0.8 and the historical BLER is weighed 0.2. For block error estimation, we use the minimum RSSI value as the RSSI base, and sum up all the error probabilities according to the RSSI distances to the base. When the sum exceeds 1, the block is estimated erroneous.

Evaluation for each building block. We first conduct separate experiments to study the impacts of each building blocks of ZiXOR, i.e., block error estimation and redundancy estimation. After that, we also study the impact of varying block sizes. For block error estimation, we mainly study the relationship between estimation accuracy and various parameters such as BER (bit error rate), RSSI threshold and block size. For redundancy estimation, we define an accurate estimation as the case where the number of redundant blocks is the same with (or larger by 1) the number of block errors. Then, we study the redundancy estimation accuracy, and evaluate the extra overhead when the redundancy is over/under estimated. For block size, we fix other parameters and tune block



(a) Block error detection with different BER (b) Block error detection with different block size

Fig. 12. Block error estimation.

size to study its impact. We will also discuss the further design spaces regarding the block size adaptation in Section 5.2.

Testbed experiments. ZiXOR can be generally used in many network layer protocols [18], [32]–[34] We incorporate ZiXOR into the collection tree protocol (CTP) and conduct experiments with our 8x10 TelosB nodes testbed (as shown in Figure 10). The radio power is set to -32.5 dBm to enable a 5-hop network. The channel is set to 26 to minimize the impact of Wi-Fi interference, since it overlaps the least with Wi-Fi communication channels [11] (It can still be interfered by Wi-Fi). Figure 11 shows the CDF (cumulative distribution function) of link qualities of neighboring nodes in the testbed. We can see that when there is no Wi-Fi interference, almost all links are good links (with link qualities $\geq$90%). When Wi-Fi interference presents, about 30% turn into intermediate links (with link qualities in (40%$\sim$90%)). This confirms that the dominating interference for indoor WSNs like ours is Wi-Fi interference. We also perform different laptop actions to study different interference patterns, i.e., web browsing, video streaming and mixed.

We change CTP's routing metric ETX (expected number of transmissions) into EBTX (expected number of block transmissions), such that the most effective relay nodes can be selected in the context of blocked transmission. The calculation of EBTX is as follows:

$$E_{p+1} = E_p + \frac{N}{1-e_b} \tag{8}$$

where E_{p+1} is EBTX from the node with hopcount $p+1$ to the sink, and $\frac{N}{1-e_b}$ is the EBTX from the node at hop $p+1$ to its parent at hop p (e_b is the block error rate and N is the number of blocks). We compare the end-to-end performances (e.g., latency, data yield, and transmissions) of the revised CTP using different approaches.

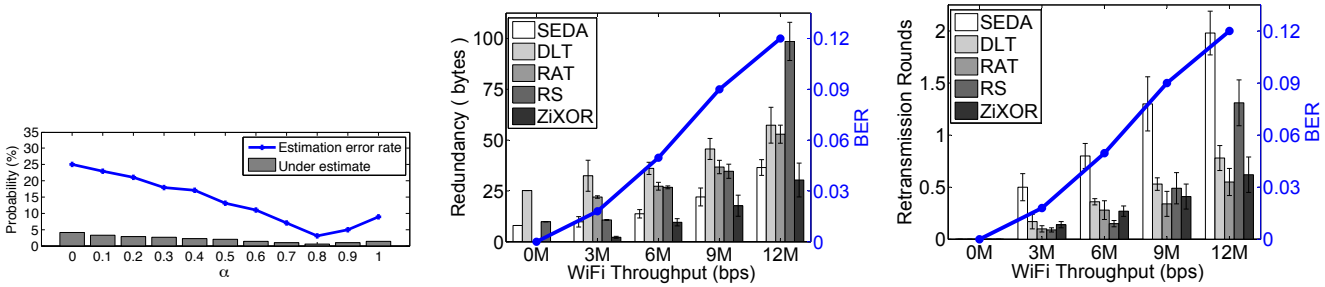
5.2 Evaluating The Building Blocks

5.2.1 *ZiXOR Coding*

Table 2 shows the coding efficiencies of different coding approaches in terms of delay performance with different MCUs. RAT selectively employs BCH code or Hamming code according to the interference, thus the decoding time is large. DLT uses fountain codes and requires Gaussian elimination (GE) for decoding. Though the decoding delay is reduced by paralleling the block receiving and GE, the decoding is still considerable when used for typical ZigBee communications (250Kbps with CC2420 radios). Comparatively, ZiXOR encodes only the redundant blocks, and its decoding requires only simple XOR operations. As a result, we can see that ZiXOR indeed achieves the most lightweight encoding/decoding on MSP430/Cortex-M0+/Cortex-M3.

TABLE 2
Coding efficiency on different platforms.

Platform \ Delay(ms)		BCH (15,1)	BCH (15,5)	BCH (15,7)	Hamming (12,8)	Hamming (7,4)	Hamming (16,11)	RS (15,13)	RS (15,7)	RS (15,3)	DLT	**ZiXOR**
MSP430	Encoding	0.70	0.30	0.20	0.20	0.40	3.10	23.30	25.10	29.70	0.20	**0.10**
	Decoding	83.90	74.80	54.70	4.30	3.90	3.90	31.20	101.90	142.30	4.40	**0.50**
cortex-M0+	Encoding	0.34	0.16	0.10	0.11	0.22	1.60	13.54	14.69	16.66	0.10	**0.06**
	Decoding	41.81	42.34	27.62	2.30	2.24	2.17	16.58	53.87	69.42	2.20	**0.26**
cortex-M3	Encoding	0.15	0.06	0.05	0.05	0.08	0.70	5.89	6.34	7.89	0.05	**0.02**
	Decoding	20.48	16.48	10.66	1.13	1.08	1.01	6.29	21.88	27.99	1.05	**0.11**
cortex-M4	Encoding	0.02	0.01	0.01	0.01	0.01	0.10	0.42	0.75	1.20	0.01	**<0.01**
	Decoding	3.26	2.15	0.83	0.13	0.15	0.09	0.65	3.11	3.51	0.07	**0.02**
cortex-M7	Encoding	<0.01	<0.01	<0.01	<0.01	<0.01	0.01	0.04	0.08	0.13	<0.01	**<0.01**
	Decoding	0.11	0.14	0.04	0.01	<0.01	0.01	0.02	0.13	0.28	<0.01	**<0.01**



(a) Redundancy with different block error rates. (b) Redundancy percentage comparison. (c) Transmission rounds comparison.

Fig. 13. Redundancy estimation.

We also notice that the improvement becomes very small as the MCU becomes more powerful (Cortex-M4/Cortex-M7). On the other hand, we should notice that the energy consumption of Cortex-M4 and Cortex-M7 (with run mode power of 38mA and 208mA, respectively) is much higher than that of MSP430 (with run mode power 1.8mA). Therefore, when used for low power networks without dense computational tasks, ZiXOR can achieve improvement in terms of coding delay.

5.2.2 *Block error estimation*

As described in Section 4.4, ZiXOR uses the IRS-based block error estimation. This approach reduces the transmission overhead whereas possibly increases the retransmission overhead when the estimation is inaccurate.

We first recall the impact of FPs and FNs before presenting the results. When FPs happen (an erroneous block is estimated correct), the further recovery requires extra negotiations and calculations. When FNs happen (an correct block is estimated erroneous), the packet can still pass the packet level CRC check and the block that is estimated erroneous will not incur any extra overhead. When both FPs and FNs happen within a packet, the retransmitted blocks will not match the erroneous blocks and retransmissions are inevitable. The approach of summing up all probabilities has an inherent bias on FN, thus most errors are likely to be FNs.

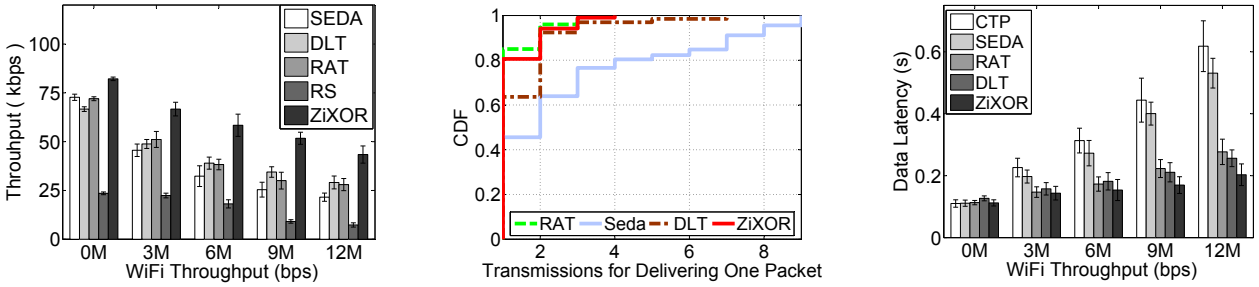
Figure 12(a) shows the estimation accuracy with different BERs. The corresponding packet reception rates (PRRs) are denoted in the brackets. we observe that, when BER increases, the estimation error rate decreases. The reason is that when BER increases, there are more corrupted packets and less correct packets. Then the fractions of FNs decrease and the fractions of FPs increase. As the probability of FPs is inherently smaller than that of FNs (Section 4.4), the overall error rate decreases.

Figure 12(b) shows the accuracy with different block sizes. When block size increases, the probability of FNs first decreases and then increases. The reason is that when block size rises, more bytes can be used for error estimation, and the estimation accuracy would increase. However, when the block size continues to increase, more small error rates would be summed up, and there will be more FNs. Similarly, since FPs happen only when all byte errors are estimated correct, the probability of FPs will decrease when more bytes are included in a block.

5.2.3 *Redundancy estimation*

We use moving average for redundancy estimation. The weight of instant redundancy sample (expected number of erroneous blocks) α (0∼1) is the key for accurate estimation (the weight of history is 1-α). We change α and conduct separate redundancy estimations. The results are shown in Figure 13(a). We can see that, in our experiment, $\alpha = 0.8$ achieves the most accurate estimation, which means the network condition is bursty and instant samples should be weighed more. We can also see that the retransmission probability is always much smaller than the estimation error probability. The reason is that, retransmissions happen only when the redundancy is under-estimated. From Figure 12, we can see that most errors of block error estimation are FNs, thus the redundancy is more likely to be over-estimated, with which retransmissions are not necessary.

We then compare the actually transmitted redundancies of ZiXOR with other approaches under different interference levels in Figure 13(b). Similar to [11], we use Iperf [35] to explicitly control the Wi-Fi transmission rates, so as to tune the interference conditions. We can see that under different interference conditions, ZiXOR achieves the least number of redundancy. The reason is two-fold: first, due to the bursty corruptions, bit errors are likely clustered in several consecutive blocks; Second, XOR coding



(a) Throughput comparison. (b) Transmission rounds comparison. (c) Latency comparison.

Fig. 14. Evaluation with CTP.

based approach recovers errors at the block level granularity, i.e., one block error can be recovered by one redundant block no matter how many bits are incorrect in the block. DLT also recover errors in block level. However, its redundancy depends on the linearity of the coefficients of the received blocks. E.g., an 8-block DLT packet can be expectedly recovered using 10 blocks.

We further study the retransmission rounds, which will be necessary when the redundancy is under-estimated. It is worth noting that we always use optimal parameters for RS/BCH code in the experiment, such that the bit errors can never exceed its recovery ability. Figure 13(c) shows the retransmission rounds. We can see that the optimal RS code achieves the least number of retransmission rounds when the Wi-Fi throughput is under 6Mbps. The reason is that RS code can recover in-consecutive corruptions (about 13% in our measurement) as long as the errors do not exceed the recovery ability. We can also see that ZiXOR outperforms all other approaches except RS/RAT, the reason is that ZiXOR distributes errors to different blocks, and can thus recover one block error using one redundant block.

5.3 Testbed Results

Recall that we incorporate different coding schemes into CTP to compare the performance. We compare three end-to-end important metrics: throughput, transmission count, and data yield. Throughput is the per-second number of bytes delivered from the source node to the sink node. Transmission count is the number of transmissions used for successfully delivering one packet to the sink node.

Figure 14(a) compares the throughput of CTP using Seda, DLT, RAT, RS and ZiXOR under different interferences. We can see that (1) ZiXOR achieves the highest throughput under different scenarios. The reason is that the decoding of ZiXOR has a $1\times$ time relation with the number of block errors, while RAT/RS's adds $2\times$ redundancy and DLT adds $\approx 1.25\times$ redundancy for block errors. (2) The improvement of ZiXOR over RAT and Seda increases along with the interference. The reason is that when there are fewer block errors, RS/RAT can select more lightweight coding schemes (e.g., RS(15,13) and Hamming (7,4)). (3) The improvement over DLT also increases. The reason is that ZiXOR coding can benefit more from more bursty errors in packets, while DLT cannot benefit from the burstiness. As a result, although both throughput decreases, the improvement of ZiXOR over DLT increases.

Next, we take a step further to study why ZiXOR outperforms other approaches. Figure 14(b) shows the transmission rounds for delivering one packet. We can see that, (1) FEC approaches (ZiXOR,DLT,RAT) have much fewer transmission rounds. More specifically, ZiXOR has the least number of transmission rounds. The reason is two fold: a) it adaptively adds redundancy according to the recent block error rates, and tends to cover the possible block errors. b) Moreover, when decoding failures happen, the already received blocks are still useful, which is likely to reduce further retransmissions. (2) In ZiXOR, there are also some fractions (about 19%) of packets that have 2 or more transmission rounds. The reason is that not all corruptions are consecutive, where retransmissions are needed. (3) Though RAT has fewer transmission rounds than ZiXOR, its decoding time is much larger than ZiXOR, thus the overall throughput of RAT is worse than ZiXOR.

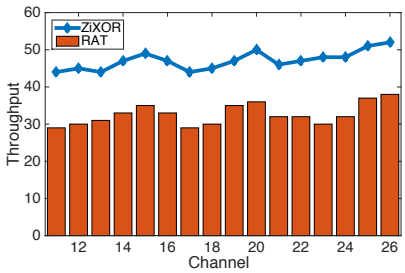


Fig. 15. Comparison on different channels.

Figure 14(c) shows the 5-hop data delivery latency of ZiXOR and other approaches. We can see that when the Wi-Fi interference becomes severe, the latencies of all approaches increase. Specifically, when Wi-Fi is off, ZiXOR has similar latency performance with other approaches. When there is Wi-Fi interference, ZiXOR's latency is smaller than others. The reason is that ZiXOR's encoding/decoding is much more lightweight than other approaches (as in Table 2).

Figure 15 shows the performance improvement of ZiXOR compared to RAT, with different channels. We can see that 1) Channels 15, 20, 25, 26 achieve higher throughput than other channels. 2) The improvement of ZiXOR to RAT on these channels is smaller than those in other channels. The reason is two-fold: Firstly, the PRRs on these channels are higher than those in other channels, leaving less room for improvement. Secondly, these channels have smaller fraction of single-burst packets than other channels.

5.4 Comparison with interleaving

RS code assumes the use of symbol level interleaving to distribute the errors to the whole packet and then recover any k random

errors with RS code for each block. Differently, ZiXORs modulo-k coding can directly distribute the bursty errors into different redundant blocks and recover the errors using simple XOR code. ZiXOR is more computationally lightweight. Besides, the difference between ZiXOR and interleaving-based RS code is listed as follows.

1) RS code allows for random error distribution and can be used in both bursty and non-bursty scenarios while ZiXOR can be mainly used in bursty scenarios.
2) The recovery ability is different. RS(15,7) can recover up to four bit errors using eight redundant bits. Two bits are required to recover one bit error. Interleaving is used to avoid too many errors clustered in one block, such that RS code can be applied. Differently, ZiXOR recovers errors in block level. Eight redundant bits can recover up to eight bit errors. Compared to RS code, one bit is able to recover one bit error, which allows us to greatly reduce the amount of redundancy.
3) Interleaving-based RS code requires the whole payload to be buffered before transmission while ZiXOR transmits native payload with encoded blocks, and do not require the whole payload to be buffered. This could be a potential advantage for scenarios with dense network traffic demands.

6 Conclusions

In this paper, we study the problem of ZigBee error recovery under Wi-Fi interference. Motivated by the bursty nature of Wi-Fi interfered corruptions, we propose a novel forward error recovery scheme for improving ZigBee communication performance under Wi-Fi interference. While bringing the ability of FEC with extremely low decoding overhead using only XOR operations, we also eliminate all control overhead by using the in-packet RSSI sampling technique. Overall, ZiXOR is lightweight in terms of both transmission and coding, and can indeed improve ZigBee performance under Wi-Fi interference. We implement ZiXOR with TelosB/TinyOS, and the evaluation results show that ZiXOR greatly outperforms state-of-the-art works.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 61602095 and No. 61472360), the Fundamental Research Funds for the Central Universities (No. ZYGX2016KYQD098 and No. 2016FZA5010), National Key Technology R&D Program (Grant No. 2014BAK15B02), CCF-Intel Young Faculty Researcher Program, CCF-Tencent Open Research Fund, China Ministry of Education—China Mobile Joint Project under Grant No. MCM20150401 and the EU FP7 CLIMBER project under Grant Agreement No. PIRSES-GA-2012-318939. Wei Dong is the corresponding author.

References

[1] C. Reinisch, M. J. Kofler, and W. Kastner, "Thinkhome: A smart home as digital ecosystem," in *Proc. of IEEE DEST*, 2010.
[2] A. Milenković, C. Otto, and E. Jovanov, "Wireless sensor networks for personal health monitoring: Issues and an implementation," *Computer communications*, vol. 29, no. 13, pp. 2521–2533, 2006.
[3] A. Wood, G. Virone, T. Doan, Q. Cao, L. Selavo, Y. Wu, L. Fang, Z. He, S. Lin, and J. Stankovic, "Alarm-net: Wireless sensor networks for assisted-living and residential monitoring," *University of Virginia Computer Science Department Technical Report*, vol. 2, 2006.
[4] J.-H. Hauer, V. Handziski, and A. Wolisz, "Experimental study of the impact of wlan interference on ieee 802.15. 4 body area networks," in *Wireless sensor networks*. Springer, 2009, pp. 17–32.
[5] X. Zheng, Z. Cao, J. Wang, Y. He, and Y. Liu, "Zisense: towards interference resilient duty cycling in wireless sensor networks," in *Proc. of ACM SenSys*, 2014, pp. 119–133.
[6] J. Hou, B. Chang, D.-K. Cho, and M. Gerla, "Minimizing 802.11 interference on zigbee medical sensors," in *Proc. of the Fourth International Conference on Body Area Networks*, 2009.
[7] R. K. Ganti, P. Jayachandran, H. Luo, and T. F. Abdelzaher, "Datalink streaming in wireless sensor networks," in *Proc. of ACM SenSys*, 2006.
[8] B. Han, A. Schulman, F. Gringoli, N. Spring, B. Bhattacharjee, L. Nava, L. Ji, S. Lee, and R. R. Miller, "Maranello: Practical partial packet recovery for 802.11." in *Proc. of USENIX NSDI*, 2010.
[9] J.-H. Hauer, A. Willig, and A. Wolisz, "Mitigating the effects of rf interference through rssi-based error recovery," in *Wireless Sensor Networks*. Springer, 2010, pp. 224–239.
[10] K. C.-J. Lin, N. Kushman, and D. Katabi, "Ziptx: Harnessing partial packets in 802.11 networks," in *Proc. of ACM MobiCom*, 2008.
[11] 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.
[12] P. Guo, J. Cao, K. Zhang, and X. Liu, "Enhancing zigbee throughput under wifi interference using real-time adaptive coding," in *Proc. of IEEE INFOCOM*, 2014.
[13] B. Chen, Z. Zhou, Y. Zhao, and H. Yu, "Efficient error estimating coding: Feasibility and applications," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 1, pp. 29–44, 2012.
[14] J. Huang, G. Xing, G. Zhou, and R. Zhou, "Beyond co-existence: Exploiting wifi white space for zigbee performance assurance," in *Proc. of IEEE ICNP*, 2010.
[15] X. Zhang and K. G. Shin, "Enabling coexistence of heterogeneous wireless systems: case for zigbee and wifi," in *Proc. of ACM MobiHoc*, 2011.
[16] F. Barac, M. Gidlund, and T. Zhang, "Scrutinizing bit-and symbol-errors of ieee 802.15. 4 communication in industrial environments," *Instrumentation and Measurement, IEEE Transactions on*, vol. 63, no. 7, p. 1783C1794, 2014.
[17] W. Du, Z. Li, J. C. Liando, and M. Li, "From rateless to distanceless: enabling sparse sensor network deployment in large areas," in *Proc. of SenSys*, 2014.
[18] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proc. of ACM SenSys*, 2009.
[19] K. Srinivasan, M. A. Kazandjieva, S. Agarwal, and P. Levis, "The β-factor: measuring wireless link burstiness," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 2008, pp. 29–42.
[20] K. Srinivasan, M. Jain, J. Choi, T. Azim, E. Kim, P. Levis, and B. Krishnamachari, "The κ Factor: Inferring Protocol Performance Using Inter-link Reception Correlation," in *Proc. of ACM MobiCom*, 2010.
[21] I. Hou, Y. Tsai, T. Abdelzaher, and I. Gupta, "AdapCode: Adaptive Network Coding for Code Updates in Wireless Sensor Networks," in *Proc. of INFOCOM*, 2008.
[22] M. H. Alizai, O. Landsiedel, J. Á. B. Link, S. Götz, and K. Wehrle, "Bursty traffic over bursty links," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2009, pp. 71–84.
[23] K. Srinivasan, P. Dutta, A. Tavakoli, and P. Levis, "Understanding the causes of packet delivery success and failure in dense wireless sensor networks," in *Proc. of SenSys*, 2006.
[24] F. Hermans, O. Rensfelt, T. Voigt, E. Ngai, L.-Å. Norden, and P. Gunningberg, "Sonic: classifying interference in 802.15. 4 sensor networks," in *Proc. of IPSN*, 2013.
[25] T. Liu and A. E. Cerpa, "Temporal Adaptive Link Quality Prediction with Online Learning," *ACM Transactions on Sensor Networks*, vol. 10, no. 3, 2014.
[26] G. Wei, Y. Ling, B. Guo, B. Xiao, and A. V. Vasilakos, "Prediction-based data aggregation in wireless sensor networks: Combining grey model and kalman filter," *Computer Communications*, vol. 34, no. 6, pp. 793–802, 2011.
[27] T. Bujlow, T. Riaz, and J. M. Pedersen, "A method for classification of network traffic based on c5. 0 machine learning algorithm," in *Computing, Networking and Communications (ICNC), 2012 International Conference on*. IEEE, 2012, pp. 237–241.
[28] Z. Chen and D. Wu, "Prediction of transmission distortion for wireless video communication: Analysis," *Image Processing, IEEE Transactions on*, vol. 21, no. 3, pp. 1123–1137, 2012.

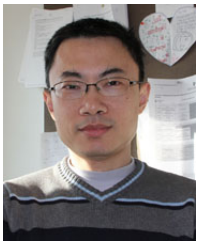
[29] H. Dubois-Ferrière, D. Estrin, and M. Vetterli, "Packet combining in sensor networks," in *Proc. of ACM SenSys*, 2005, pp. 102–115.
[30] T. Jin, G. Noubir, and B. Sheng, "Wizi-cloud: Application-transparent dual zigbee-wifi radios for low power internet access," in *Proc. of IEEE INFOCOM*, 2011, pp. 1593–1601.
[31] ——, "Wizi-cloud: Application-transparent dual zigbee-wifi radios for mobile internet," in *Tech. Report*, 2012.
[32] F. Sutton and L. Thiele, "Wake-up flooding: an asynchronous network flooding primitive," in *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*. ACM, 2015, pp. 360–361.
[33] Z. Zhao, W. Dong, J. Bu, Y. Gu, and C. Chen, "Link-correlation-aware data dissemination in wireless sensor networks," *Industrial Electronics, IEEE Transactions on*, vol. 62, no. 9, pp. 5747–5757, 2015.
[34] J. J. Pérez-Solano and S. Felici-Castell, "Adaptive time window linear regression algorithm for accurate time synchronization in wireless sensor networks," *Ad Hoc Networks*, vol. 24, pp. 92–108, 2015.
[35] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The tcp/udp bandwidth measurement tool," *htt p://dast. nlanr. net/Projects*, 2005.

Zhiwei Zhao (S'11-M'16) received his PhD degree in computer science from Zhejiang University in 2015. He is currently an assistant professor at the College of Computer Science and Engineering in University of Electronic Science and Technology of China (UESTC). His research interests include on wireless computing, heterogeneous wireless networks, protocol design and network coding. He is a member of IEEE.

Wei Dong (S'08-M'11) received the B.S. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 2005 and 2011, respectively. He is currently an Associate Professor with the College of Computer Science, Zhejiang University .His research interests include network measurement, wireless and mobile computing, and sensor networks. He is an member of IEEE and ACM, and a senior member of CCF.

Gonglong Chen (S'15) received his BS degree from the College of Criminal Justice at East China University of Political Science and Law. He is currently a second year Ph.D student at the College of Computer Science in Zhejiang University. His current research interests include wireless and mobile computing. He is a student member of IEEE.

Geyong Min (M'01) is the Chair Professor and Director of High Performance Computing and Networking (HPCN) Research Group at the University of Exeter, UK. He received the PhD degree in Computing Science from the University of Glasgow, UK, in 2003, and the B.Sc. degree in Computer Science from Huazhong University of Science and Technology, China, in 1995. He joined the University of Bradford as a Lecturer in 2002, became a Senior Lecturer in 2005 and a Reader in 2007, and was promoted to a Professor in Computer Science in 2012. His main research interests include Next-Generation Internet, Wireless Networks, Mobile Computing, Cloud Computing, Big Data, Multimedia Systems, Information Security, System Modelling and Performance Optimization.



Tao Gu (S'03-M'07-SM'14) is currently an Associate Professor in Computer Science at RMIT University, Australia. He received his Bachelor degree from Huazhong University of Science and Technology, M.Sc. from Nanyang Technological University, Singapore, and Ph.D. in computer science from National University of Singapore.His current research interests include mobile computing, ubiquitous/pervasive computing, wireless sensor networks, distributed network systems, sensor data analytics, Internet of Things, and online social networks. He is a Senior Member of IEEE and a member of ACM.

Jiajun Bu (M'06) received the B.S. and Ph.D. degrees in computer science from Zhejiang University, China, in 1995 and 2000, respectively. He is a professor in College of Computer Science and the deputy dean of the Department of Digital Media and Network Technology at Zhejiang University. His research interests include embedded system, mobile multimedia, and data mining. He is a member of IEEE and ACM.