CARE: Corruption-Aware Retransmission with Adaptive Coding for the Low-Power Wireless

Wei Dong, Jie Yu, and Xiaojin Liu
College of Computer Science, Zhejiang University
Email: {dongw, yuj, liuxj}@emnets.org

***Abstract*—Wireless communications are inherently unreliable, especially for low-power wireless networks like 802.15.4. Packet corruptions typically occur because of interference, fading or noise. In this paper, we investigate the problem of corruption-aware retransmission with adaptive coding in commercial-off-the-shelf (COTS) low-power devices. We propose an accurate corruption detection algorithm for identifying the number of corrupted bytes in a packet based on the RSSI time series during packet reception. Based on the corruption level, we design an adaptive coding scheme based on Reed Solomon (RS) codes. We carefully select the coding parameters so as to optimize the network performance. Finally, we design and implement CARE, a Corruption-Aware REtransmission protocol by incorporating corruption detection and adaptive coding. We conduct extensive experiments based on COTS low-power devices. Results show that CARE significantly improves the performance for weak and highly interfered links while incurring no additional overhead for good links.**

I. Introduction

Wireless communications are inherently unreliable, especially for low-power wireless networks like 802.15.4. Packet corruptions typically occur because of interference, fading or noise.

In existing 802.15.4 networks, Automatic Repeat reQuest (ARQ) is typically used to ensure successful packet reception in the face of packet losses or packet corruptions. However, ARQ is costly. On one hand, when there are small errors, retransmitting the entire frame may unnecessarily send redundant bits that may have already been received correctly by the receiver. On the other hand, when there are severe errors, many retransmission rounds are needed to ensure successful packet reception.

Forward Error Correction (FEC) has been proposed to reduce retransmission overhead. FEC adds additional upfront redundancy to each packet and can correct a certain level of errors. A key question for FEC is how much redundancy we need to add. If there is little redundancy, we may not be able to recover a packet in high-error scenarios. If there is high redundancy, we may introduce additional overhead in low-error scenarios.

Retransmission based on packet combining [1] integrates both the above techniques: the retransmitted packets encode the redundancy information to help recovery. When there are no errors, a single transmission will succeed without incurring additional overhead. In high-error scenarios, a relatively small number of retransmission rounds is needed since we can jointly exploit the redundancy information in multiple copies of the same packet for recovery. On a single link, packet combining behaves similarly to a class of techniques called Hybrid ARQ (HARQ) [2].

HARQ has been widely used in cellular networks. *Unfortunately*, existing HARQ techniques are not optimized for low-power wireless networks like 802.15.4. One important limitation is that existing techniques cannot adapt their coding rate *accurately and quickly enough* according to the corruption level the original packet has been experienced. This limitation leads to performance degradations. For example, if a packet experiences a single bit error, an entire (parity) packet will be retransmitted, incurring unnecessary overhead. On the other hand, if a packet experiences severe errors, many retransmission rounds will be involved because the fixed coding rate (e.g., $\sim$2/3 in TinyRS [3]) may not be sufficient to correct the errors induced by the channels.

In this paper, we investigate the problem of corruption-aware retransmission with adaptive coding in commercial-off-the-shelf (COTS) low-power devices like TelosB sensor nodes. We utilize the recently proposed technique to sample RSSI values during packet reception with a high frequency [4]. The RSSI time series enables us to decide the corruption level of a packet instantly and accurately. Based on the corruption level, we design an adaptive coding scheme to retransmit the redundancy information. Finally, we propose CARE, a Corruption-Aware REtransmission protocol by incorporating corruption detection and adaptive coding. In no error scenarios, CARE does not incur additional overhead. In low/high error scenarios, CARE can retransmit an appropriate level of redundancy information. Therefore, our approach is adaptive to dynamically changing channel conditions and performs well for good links, weak links, or highly interfered links.

The contributions of this paper are summarized as follows:

- We propose a novel corruption detection algorithm considering the transitional region of wireless packet reception, resulting in significantly better accuracy compared with state-of-the-art algorithm [5] on COTS devices.
- We design an adaptive coding scheme based on Reed Solomon (RS) codes. We carefully select the coding parameters to optimize the network performance on resource-constrained low-power devices.
- We design and implement the CARE retransmission protocol. We conduct extensive experiments based on COTS low-power devices. Results show that CARE improves the performance significantly for highly interfered links

while incurring no additional overhead for good links.

The rest of this paper is structured as follows. Section II introduces the large body of related work. Section III presents the design of CARE. Section IV presents an analytical model to compare different approaches including CARE. Section V shows the evaluation results, and finally, Section VI concludes this paper and gives future research directions.

II. Related Work

There is a large body of work on improving the reliability of wireless communication.

ARQ is a well-known technique to recover packet losses, where packets without ACKs are simply retransmitted. Seda [6] treats the packet from the upper layer as a continuous stream of bytes. It breaks the data stream into blocks, and retransmits erroneous blocks only (as opposed to the entire erroneous frame). Maranello [7] is a similar approach designed for 802.11 systems. Instead of transmitting the per-block CRCs, it puts the per-block CRCs in NAKs, eliminating the overhead for correct packets. A key challenge of these approaches is how to determine the block size. If the block size is too small, the per-block CRC overhead will be large. If the block size is too large, the retransmission of erroneous blocks might be costly when there are only few bit errors.

PPR [8] uses SoftPHY hints to evaluate which bits are more likely in error and only retransmits the erroneous parts rather than the entire packets. SoftRate [9] and AccuRate [10] exploit PHY layer information to estimate the error rate after receiving a packet, and then adapt coding/modulation schemes to optimize network performance. In μACK [11], instead of waiting for the entire transmission to end before sending the ACK, the receiver sends smaller μACKs for every few symbols, on a separate narrow feedback channel. Based on these μACKs, the sender only retransmits the lost symbols after the last data symbol in the frame, thereby adaptively changing the frame size to ensure it is successfully delivered. MISC [12] is a packet retransmission scheme that merges incorrect symbols from multiple (re)transmissions to produce correct ones. All the above works are designed for 802.11 networks and require significant modifications in the hardware. In contrast, our approach can be directly used for COTS low-power devices.

ZipTx [13] is a software only solution for recovering partial packets in 802.11 wireless networks. While ZipTx uses pilot bits for estimating the BER, we exploit in-packet RSSI information, incurring no extra overhead for the correct packets. REPE [5] equips each sensor nodes with a high resolution timer (i.e., 62.5kHz) and periodically samples the RSSI values for each received symbol. Based on the RSSI time series, the receiver then requests the detected erroneous symbols for retransmission. Our current work differs from REPE in two important ways. First, we do not require additional hardware, yet yielding a higher corruption estimation accuracy than REPE. Second, we employ adaptive coding which significantly reduces the number of retransmission rounds, resulting in higher channel utilization.

FEC typically adds upfront information in a packet for error recovery. TinyRS [3] is an implementation of Reed Solomon codes on the TelosB/TinyOS platform. Although it can correct errors in relatively high-error scenarios (e.g., highly interfered links), it introduces too much overhead when the channel condition is good enough. ACR [14] intelligently determines the amount of redundancy for FEC by actively generating short packets and long packets in dense sensor networks. It is possible because the corruption pattern for long packets are predictable when the short packets collide with long packets. ACR is tailored to handle packet corruption caused by in-network interference, and cannot well handle packet corruption caused by external interference (e.g., 802.11) where the corruption pattern is unpredictable. RAT [15] is a recent work exploiting WiFi traffic pattern for adaptive coding. Our current work differs from RAT in three important ways. First, our work obtains the corruption level by observing the received RSSI time series while RAT uses statistical model to derive the corruption level. Hence, our corruption detection is more accurate since we directly utilize PHY-level information. Second, our work requires passive RSSI sampling during packet reception while RAT requires active RSSI sampling when no actual communication is ongoing. Hence, our approach is more energy efficient. Third, we have established mathematical equations by explicitly considering the error correcting capability of the RS codes as well as the corruption level, while RAT only gives heuristics for adaptive coding.

Hybrid ARQ (HARQ) reduces the retransmission overhead using FEC, in addition to ARQ. Type I Hybrid ARQ resends the entire packet (along with FEC and error detection (ED)) when the packet is lost. Type II Hybrid ARQ reduces the ARQ overhead when the medium is mostly loss-free. Packets include ED but no FEC the first time they are sent. Retransmissions include both FEC and ED. The packet can be decoded by combining the original transmission (plain packet) and the encoded retransmission (parity packet). Generally, the packet cannot be decoded by only using the parity packet. In type-III Hybrid ARQ, each retransmitted packet is self-decidable, i.e., the packet can be decoded by only using the parity packet. SPaC [1] is packet combining scheme for low-power sensor networks. On point-to-point links, packet combining behaves similarly to HARQ. Our work differs from the above work in two key aspects. First, the coding rate in our work is self-adaptive according to the corruption level of the previously transmitted packet. Second, different from HARQ in WLAN (with 1/3 Turbo coding) and SPaC in sensor networks (with Hamming(8,4) coding), our approach employs Reed Solomon codes with a set of carefully selected parameters customized for the low-power wireless.

Table 1 compares various approaches with respect to three key desired features:

- *No extra overhead for correct packets*. This is important as to optimize the common case of successful packet transmission.
- *No H/W support*. This is important to practically apply

Table 1: Comparison of different approaches. The above 7 approaches are based on 802.11 while the rest are based on 802.15.4.

Approach	No extra bits for correct packets	No H/W support	Accurate corrupt. detect.
ZipTx	×	√	×
PPR	√	×	√
SoftRate	√	×	√
AccuRate	√	×	√
Maranello	√	√	×
MISC	√	×	×
uACK	√	×	√
CARE	√	√	√
Seda	×	√	×
SPaC	√	√	×
RAT	√	√	×
TinyRS	×	√	×
ACR	×	√	×
REPE	√	×	√

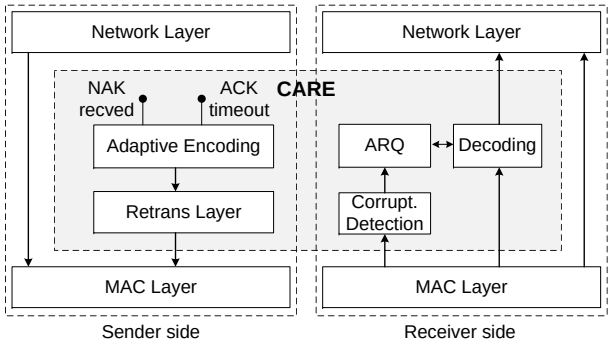


Fig. 1: CARE architecture

the approach on existing devices.

- *Accurate corruption detection*. We need a high corruption detection accuracy (e.g., down to the level of a few symbols or bytes) so that the extra redundancy can accurately match the channel conditions.

We see that none of the previous approaches achieve all these features simultaneously, motivating the design of CARE.

III. CARE Design

Fig. 1 shows the architecture of CARE. CARE sits between the MAC layer and the network layer.

At the sender side, the network layer transmits packets which are handled at lower layers. If a packet is transmitted for the first time, it is directly passed to the MAC layer. If the packet is successfully transmitted (i.e., a link-layer ACK is successfully received), no further actions are taken. Therefore, CARE incurs no additional overhead when the link is perfect. Otherwise, the sender will retransmit the packet with adaptive coding. There are two cases: (1) the sender receives a NAK carrying the corruption level (detected at the receiver). (2) no ACK/NAK are received before the ACK timer times out. In the first case, the sender will retransmit a parity packet considering the corruption level. In the second case, the sender will alternate between retransmitting a plain packet and retransmitting a self-decidable parity packet.

At the receiver side, when a packet is received, the receiver uses a high-resolution RSSI sampling procedure to sample the RSSI values during packet reception. The RSSI time series is used to estimate the corruption level for automatic repeat request. If a packet is corrupted, the receiver attempts to decode it by exploiting the information in both the plain packet and parity packet.

In the following subsections, we detail the design of CARE, including the corruption detection algorithm (Section III-A), the adaptive coding scheme (Section III-B), and the CARE retransmission protocol (Section III-C).

A. Corruption Detection

As a key building block of CARE, the corruption detection algorithm deals with the following issue: how to accurately detect the number of corrupted bytes in a received packet. Based on this corruption information, the sender can decide how to efficiently encode the retransmitted packet.

We design CARE to deal with interference especially. Recent studies show that low-power 802.15.4 transmissions can be significantly interfered by WiFi. To detect corruption, we employ a high-resolution RSSI sampling procedure during packet reception [5], [4], [16].

We implement the above procedure for the TelosB/TinyOS 2.1.1 platform. TelosB nodes are equipped with CC2420 radio which is compliant with 802.15.4. Without any modification, the maximum RSSI sampling rate the hardware can support is one reading per 200–250μs, which means that this method can sample a RSSI value for about 10 bytes [14]. The sampling frequency can be increased to one RSSI value per symbol (i.e., 4 bits for 802.15.4), at the cost of the incorporating a 62.5kHz external timer [5].

We would like to implement a high-resolution sampling procedure (at least one sample per byte) without extra hardware. To this end, we modify the radio driver in TinyOS 2.1.1. In particular, we modify the operating frequency of the SPI bus from 1MHz to 4MHz, significantly improving the reading efficiency from the CC2420 RSSI register. Our modified driver starts sampling RSSI whenever an SFD (Start Frame Delimiter) interrupt signals an incoming packet, and it keeps sampling at a rate of one sample/byte until the last byte of the packet is received.

Fig. 2 shows the obtained RSSI time series during a packet reception when there is WiFi interference. We can see that WiFi interference causes several peaks in signal strength.

Based on this observation, we design a simple but effective corruption detection algorithm outlined in Algorithm 1. The input of the algorithm is the RSSI time series during packet reception. The output is the corruption level which is defined to be the fraction of corrupted bytes in a packet. We first initialize the number of detected corrupted bytes to be 0, and the

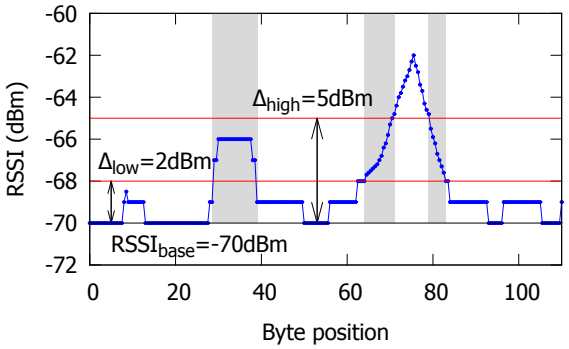


Fig. 2: Illustration of the corruption detection algorithm. The shaded areas indicate the transitional region where symbol receptions are probabilistic.

Algorithm 1: Corruption detection

Input : RSSI time series (RSSI[L]) of a L-byte packet
Output: Corruption level, CL, which is the fraction of corrupted bytes in a packet

```
CB=0; CL=1;
if range(RSSI[])≤ 2 then
    return CL;
RSSI_base=min_{0≤i<L} RSSI[i];
for i : 0 ≤ i < L do
    if RSSI[i] ≥ RSSI_base + Δ_high then
        CB++;
        Mark the i-th byte as corrupted;
    else if RSSI_base + Δ_low ≤ RSSI[i] ≤ RSSI_base + Δ_high then
        Pr = (RSSI[i] − RSSI_base − Δ_low)/(Δ_high − Δ_low) ;
        CB+=Pr ;
        Mark the i-th byte as corrupted with a probability of Pr
    else
        Mark the i-th byte as correct;
CL=CB/L;
return CL;
```

corruption level to be its default value of 1. We use the range of RSSI time series to differentiate non-interfered links from interfered links. If the current link is a non-interfered link, the algorithm returns the default corruption level of 1, i.e., CARE does not detect the corruption level for non-interfered links and will retransmit a self-decidable parity packet if needed. Otherwise, the algorithm detects the number of corrupted bytes (recorded in the variable CB) caused by interference. We mainly utilize the perceived signal strength for corruption detection. $RSSI_{base}$ is the lowest perceived signal strength which can be seen as the 802.15.4 signal strength. If the signal strength is higher than a threshold of $RSSI_{base}+\Delta_{high}$, we mark the byte as corrupted due to interference. If the signal strength is lower than a threshold of $RSSI_{base}+\Delta_{low}$, we mark the byte as correct. Otherwise, the signal strength is in the transitional region of $(RSSI_{base}+\Delta_{low}) \sim (RSSI_{base}+\Delta_{high})$, we mark the byte as corrupted with a probability of Pr. Having iterated over all RSSI values, we can estimate the number of corrupted bytes and the corruption level.

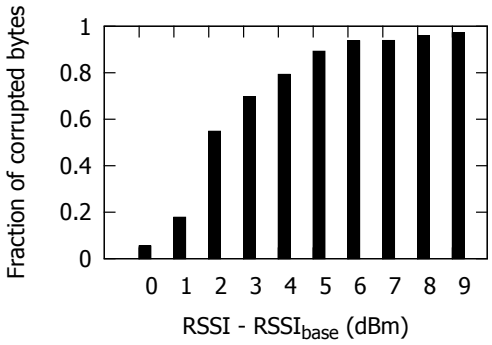


Fig. 3: Impact of received signal strength on byte corruptions.

Fig. 3 shows how the received per-byte RSSI affects corruptions under high and low interference (see experiment settings in Section V-A). Fig. 3 shows the existence of the transitional region of $RSSI_{base}+2 \sim RSSI_{base}+5$. Therefore, we select $\Delta_{low}=2$ and $\Delta_{high}=5$. Despite its simplicity, our algorithm carefully considers the impact of transitional region and thus can yield much better accuracy compared with the the prior algorithm proposed in [5]. We will show the comparison results in Section V-B.

B. Adaptive Coding

Once the corruption level is determined, the receiver will send back a NAK, piggybacking the corruption level information. Based on the received corruption level, the sender will employ an adaptive encoding scheme to retransmit the parity packet.

Among the various codes, we employ the Reed Solomon (RS) codes since RS code is found be to particularly effective against the bursty error patterns caused by interference [3]. In addition, RS code has been implemented on the low-power wireless devices like TelosB.

A RS code can be specified as $RS(N,K,m)$ where m is the symbol size in bits, N is the total number of symbols in a block, and K is the number of useful symbols in the block. Typically, $N=2^m-1$. The error recovery capability depends on $N-K$, i.e., the number of parity symbols in a block. Without specifying the error positions, a total of $(N-K)/2$ errors can be corrected in a block. With all error positions specified (these known error locations are called "erasures"), a total of $N-K$ errors can be corrected in a block. In the most general case, there are both erasures and non-erased errors. Each non-erased error counts as two erasures and the number of erasures plus twice the number of non-erased errors cannot exceed $N-K$.

Fig. 4 shows how we use RS in CARE. The plain packet has K symbols per block while the parity packet has $N-K$ symbols per block. A RS codeword is formed by the K plain symbols and $N-K$ parity symbols. Suppose a packet with L bytes payload, the number of blocks is: $n=\lceil 2L/K \rceil$.

We employ the RS code library developed by Phil Karn [17] in which N always equals to $N^*=2^m-1$ or less than N^* (i.e., shortened codes). Now we discuss how to adaptively select

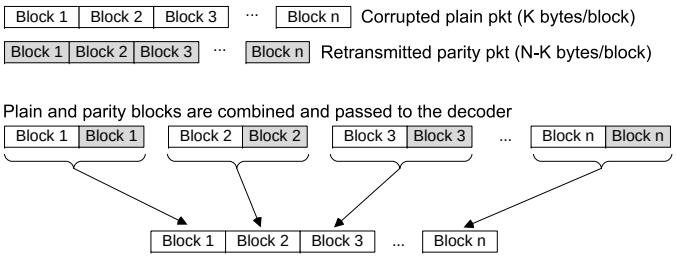


Fig. 4: Combining the plain packet and the retransmitted parity packet

the parameters so that the code is best suited to the observed corruption level. First, we select m to be 4 since 802.15.4 uses 4-bit symbols and bit errors happen at 4-bit boundary [2]. Generally, we select two types of RS codes.

Type I RS codes. In this configuration, $N = 15$. We need to determine K. We specify the following constraints for the type I RS codes:

$$\text{Goal: find } K \text{ to minimize } n(N-K)$$
$$\text{Subject to: } \begin{cases} 1 \le K < N \\ (N-K)\cdot n \le 2L_{\max} \\ CL \cdot N \le (N-K)/2 \end{cases} \quad (1)$$

We explain some notations as follows. L is the packet payload length in bytes. $n = \lceil 2L/K \rceil$ is the number of blocks. CL is the perceived corruption level of the previous plain packet. $L_{\max} = 114$ is the maximum packet length in bytes for the CC2420 radio. We want to minimize the the number of parity bytes in the retransmitted packet.

The first two constraints are given according to the definitions. The third constraint states that the retransmitted parity packet cannot exceed the maximum packet length. The fourth constraint states that the error correcting capability of the codes should exceed the corruption level, assuming that the corruption level of the retransmitted packet equals to the perceived corruption level of the previously transmitted packet.

Type II RS codes with a coding rate of 1/2. In this configuration, we select $N = 14$, $K = 7$. This is the default encoding scheme which works when we cannot find a feasible K for type I encoding, or when a packet/ACK/NAK loss event occurs. This encoding scheme is both *systematic* and *invertible*. A block code is systematic if the first k bits of a codeword are the same as the input message bits. A block code is invertible if there is a one-to-one mapping from the plain packet to the parity packet and vice-versa. Note that an invertible code is always half-rate [1]. With invertible codes, if a parity packet is received without errors, the original packet can be directly obtained. Since a parity packet has the same length as the corresponding plain packet, the system does not transmit any redundant overhead on good links, whether it transmits a plain or a parity packet.

To summarize, Fig. 5 depicts the K values given the corruption level (CL) and the packet payload length (L). CARE employs type II encoding ($N = 14$, $K = 7$) in the white area, and type I encoding ($N = 15$) in other areas.

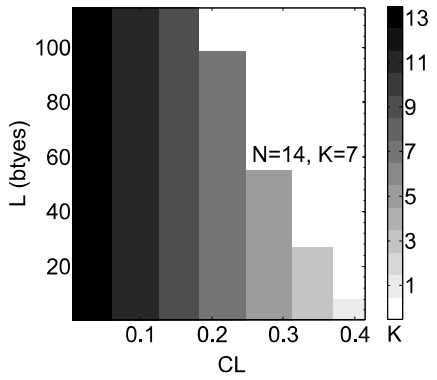


Fig. 5: Coding parameters varied with the corruption level (CL) and the packet payload length (L).

We can see that when type I encoding is employed and the corruption level increases, K decreases and the size of parity packet increases so that more errors can be corrected. There exists a corruption threshold beyond which CARE transitions from type I encoding to type II encoding. The corruption threshold becomes smaller for larger packets. This is because the parity bits in a block should be fewer so that they can be accommodated in the maximum allowable packet size.

We employ symbol interleaving to spread bursty errors to the whole packet, thus fully utilizing the correction capability of all the blocks in the packet. The receiver reverses the interleaving after receiving the packet. Thus, the order of symbols in each group is recovered while some bursty errors in the packet are spread in different blocks in the packet [18], [15].

It is also worth mentioning other options like fountain codes which are also widely used in the sensor network community [19]. Fountain codes are erasure rateless codes. Suppose a packet is divided into n blocks, the use of fountain codes allows the decoding of the packet when n' blocks are received where n' is only slightly larger than n. Fountain codes incur additional encoding and decoding overhead when the link is perfect. In addition, fountain codes usually require adding a CRC field for each block in order to decide whether the block is correct or not. This is another source of overhead.

C. The Retransmission Protocol

Based on the corruption detection algorithm and adaptive coding scheme described earlier, CARE can perform more efficient retransmissions. The CARE retransmission protocol works in the following way.

Algorithm 2 shows the code at the sender side. There are four cases. In the first case, when the network layer transmits a packet for the first time, we directly transmit the packet by invoking mac_send(). We also need to store the packet and start the ACK timer for possible retransmissions. In the second case, when an ACK is received, we know that the previous packet is successfully received. We remove the packet from the sent_buffer and notify the network layer. In the third case, when a NAK is received, we know that the corresponding

Algorithm 2: Retransmission protocol at the sender side

```
case pkt received from network layer
    put pkt in sent_buffer;
    mac_send(pkt);
    start ACK timer;
    state = next_parity;
case ACK received
    find pkt in sent_buffer corresponding to this ACK;
    remove pkt from sent_buffer;
    notify network layer;
case NAK received
    find pkt in sent_buffer corresponding to this NAK;
    extract corruption level, CL, from NAK;
    care_resend1(pkt, CL);
case ACK timer times out
    if state == next_plain then
        mac_send(pkt);
        state = next_parity;
        start ACK timer;
    else if state == next_parity then
        care_resend2(pkt);
        state = next_parity;
```

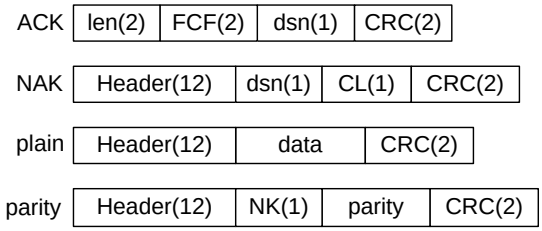


Fig. 6: Packet format with CARE

packet is corrupted at the receiver. We extract the corruption level, CL, from the NAK and retransmit a packet using type I RS encoding (or possibly type II encoding when CL is so large that the maximum allowable packet size cannot accommodate the required redundancy, see Section III-B). In the last case, when the ACK timer times out, we turn to the default retransmission scheme of alternating between sending a plain packet and a parity packet with type II encoding. There are several possible causes including the losses of ACK, NAK, or the previously transmitted packet. Algorithm 3 shows the code at the receiver side.

It is worth mentioning the packet format with CARE. Fig. 6 shows the formats of packets we described earlier. The ACK packet format complies with 802.15.4 standard. It consists of one-byte length field, two-byte FCF field, one-byte dsn field, and two-byte CRC field. The NAK packet consists of 12-byte 802.15.4 header, one-byte dsn field (so that it can be matched to the transmitted packet), one-byte CL field containing the corruption level, and two-byte CRC field. In order to differentiate plain packet and parity packet, we reuse one reserved bit (i.e., bit 8) in the FCF field in the 802.15.4 header. The parity packet contains one additional byte indicating the parameters of RS coding: the codeword length

Algorithm 3: Retransmission protocol at the receiver side

```
case SFD rising edge
    start RSSI sampling and store values in RSSI[L];
case SFD falling edge
    stop RSSI sampling;
case pkt received
    case pkt is plain and correct
        reply ACK;
        deliver_to_network(pkt);
    case pkt is plain and error
        put pkt in recv_buffer;
        CL = corruption_detection(RSSI[L]);
        reply NAK(CL);
    case pkt is parity with type II coding and is correct
        reply ACK;
        plain = rs_decode(pkt);
        deliver_to_network(plain);
    case Other
        find plain from recv_buffer corresponding to pkt;
        pkt2 = rs_decode(plain, pkt);
        if pkt2 is correct then
            reply ACK;
            deliver_to_network(pkt2);
        else
            reply NAK(CL);
```

N consumes 4 bits and the useful data length K consumes another 4 bits.

IV. Analysis

In this section, we build a model to analyze the performance of our CARE protocol. We also compare its performance to most relevant works including ARQ, SPaC [1], and REPE [5].

We use a similar model as [11]. We consider the symbol error rate since 802.15.4 PHY layer transforms 32-bit chip sequences to 4-bit symbols. Suppose the symbol error rate is e_s and a packet contains $L = 30$ bytes (i.e., $2L$ symbols). The packet error probability is thus $e_p = 1-(1-e_s)^{2L}$. To compare different techniques, we define retransmission overhead (RO) as the fraction of additional time to successfully transmit a packet.

$$RO = T_{tx}/T_{data} - 1 \tag{2}$$

where T_{tx} is the time taken to successfully deliver the packet at the receiver including retransmissions, and T_{data} is time to transmit a data packet of payload length L. $T_{data} = T_{backoff} + L/R$ where $R = 250$kbps is the data rate for the CC2420 radio and $T_{backoff} \approx 4.9$ms for the TinyOS CSMA MAC protocol. RO is zero if there is no loss, and greater than zero if there is any retransmissions. We assume the sender will persistently retransmit the lost packet until it is successfully received.

1) ARQ: The expected number of transmissions (including retransmissions) is:

$$N_{tx} = 1/(1-e_p) \tag{3}$$

Therefore,

$$T_{tx} = T_{data} N_{tx} \tag{4}$$

2) REPE: REPE always retransmits the erroneous symbols by incorporating a 62.5kHz timer. In this analysis, we assume REPE can always accurately detect the erroneous symbols, i.e., we analyze the best performance of REPE.

We denote s_k as the size of k-th retransmission packet in symbols. $s_0 = 2L$. The size of the k-th retransmission packet can be recursively calculated as

$$s_k = e_s s_{k-1} \tag{5}$$

The packet error rate of the k-th packet is

$$e_p(k) = 1 - (1 - e_s)^{s_k} \tag{6}$$

According to the calculation in [11],

$$T_{tx} = T_{data} + \sum_{k=1}^{\infty} \left(\prod_{i=1}^{k-1} e_p(i) \right) T_{data}^k \tag{7}$$

where T_{data}^k is the time to send the k-th retransmission (s_k symbols).

3) SPaC: When preambles are always detected, according to Eq.(6) in [1],

$$\frac{1}{N_{tx}} = \eta = \frac{1 - e_p \rho}{1 + e_p (1 - \rho)} \tag{8}$$

where ρ is the decoding failure probability given one plain and one parity packet.

With Hamming(8,4) coding which can correct up to one bit error per codeword (8 bits), jointly decoding two packets fails if any codeword has more than one bit error. We revise Eq.(1) in [1] to consider symbol error probability, instead of bit error probability:

$$\rho = 1 - ((1 - e_s)^2 + 2 e_s (1 - e_s) * Pr_1)^{2L} \tag{9}$$

where Pr_1 is the 1-bit error probability given a 4-bit symbol error. According the symbol mutation probability of [20], $Pr_1 \approx 0.238$ given a chip error rate of 0.3. That means, it is highly probable that there are more than one bit errors given a symbol error.

Similar to ARQ,

$$T_{tx} = N_{tx} T_{data} \tag{10}$$

4) CARE: In order to better understand CARE's performance, we analyze CARE with fixed coding rates of 1/2 (i.e., type II coding with RS(14,7,4)), 2/3 (i.e., RS(15,10,4)), as well as the general CARE with adaptive coding rate.

CARE with fixed coding rates. The analysis of CARE with fixed coding rates is similar to SPaC. We need to analyze the decoding failure probabilities $\rho_{1/2}$ and $\rho_{2/3}$ for coding rates 1/2 and 2/3 respectively.

$$\rho_{1/2} = 1 - \left(\sum_{i=0}^{3} \binom{14}{i} e_s^i (1 - e_s)^{14-i} \right)^{\lceil 2L/7 \rceil} \tag{11}$$

$$\rho_{2/3} = 1 - \left(\sum_{i=0}^{2} \binom{15}{i} e_s^i (1 - e_s)^{15-i} \right)^{\lceil 2L/10 \rceil} \tag{12}$$

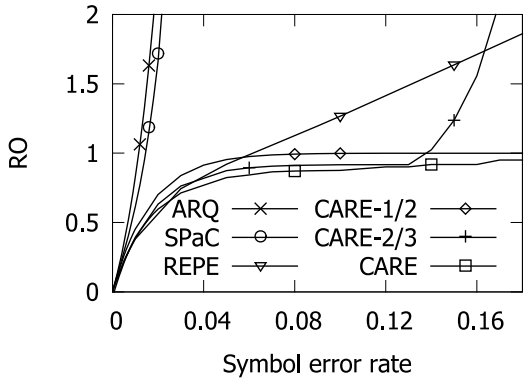


Fig. 7: Analysis results

CARE with adaptive coding rate. CARE uses type I RS coding when CL is relatively small so that the maximum allowable packet size can accommodate the required redundancy. CL can be regarded as the byte error probability e_b and it relates to symbol error probability e_s according to:

$$e_b = 1 - (1 - e_s)^2 \tag{13}$$

The time needed to successfully transmit a packet is

$$T_{tx} = T_{data} + e_p T_{data}^1 \tag{14}$$

where T_{data}^1 is time for the first retransmission. CARE can recover the packet by at most two (re)transmissions in expectation since we always set enough redundancy in the second parity packet (see Section III-B).

$$T_{data}^1 = T_{backoff} + L'/R \tag{15}$$

where $L' = \lceil 2L/K \rceil \cdot (N - K)/2$ is the size of parity packet in bytes.

CARE switches to type II RS coding when CL is too large. The analysis is similar to SPaC with $\rho_{1/2}$ given in Eq. (11).

5) Results: Fig. 7 shows the analysis results under different symbol error rates. When the symbol error rates are low, the link is very reliable and the retransmission overhead is also low. As the link becomes lossy, the retransmission overhead of ARQ increases exponentially and soon hits a wall at symbol error rate of ∼0.02, meaning that no matter how many times it retransmits, the packet cannot be delivered. REPE reduces the retransmission overhead by resending only the erroneous bits. Therefore, the retransmission overhead increases slower than ARQ, but the header and contention overhead is still high. CARE, however, maintains a very low retransmission overhead even in very lossy environments, i.e., the symbol error rate is as high as 0.2.

V. Evaluation

We implement CARE on the TelosB/TinyOS 2.1.1 platform. The RAM overhead is ∼2.4 KB, and the ROM overhead is ∼5.2 KB. Overall, this overhead is acceptable for the current TelosB nodes with a total of 10 KB RAM and 48 KB ROM.

Fig. 8: 10x2 indoor testbed

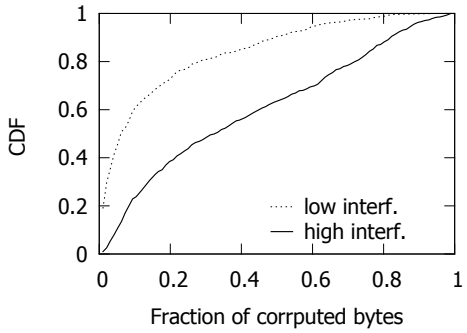


Fig. 9: Corruption levels under high and low interference.

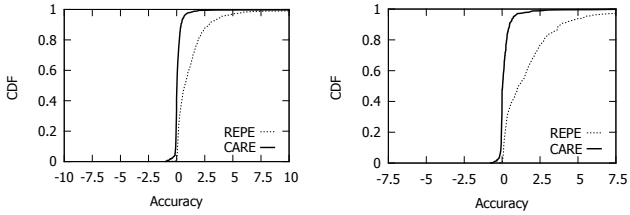


Fig. 10: Comparison of corruption detection algorithms. Left: high interference. Right: low interference. Accuracy=(# of detected corrupted bytes)/(# of real corrupted bytes)-1.

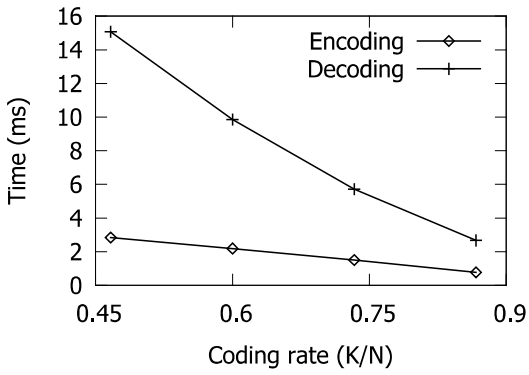


Fig. 11: Encoding/decoding overhead

A. Methodology

We evaluate our approach when there is no interference as well as high/low interference. Two TelosB nodes with CC2420 radio communicate with each other. One node sends data packets with 96 bytes payload to another node with an interval of 512ms for a total of one hour. There are two cases when there is no interference. *Good links*: two TelosB nodes are closely placed and the packet reception probability (PRR) is larger than 99%. *Weak links*: two TelosB nodes are configured with a power level of 2, at a distance of 1.5 m. With this configuration, the received signal strength is low. We measured the symbol corruption ratio to be roughly 0.04. In the presence of WiFi interference, the TelosB nodes are placed close to a laptop and a wireless access point with 802.11g mode. We use iperf to generate ∼5MB WiFi traffic with 1500 bytes TCP segment size. The channel of WiFi is 11. There are two cases when there is interference. *High interfered links*: the channel of 805.15.4 is 22. *Low interfered links*: the channel of 802.15.4 is 17. In the experiments, nodes send packets with a known content so that we know the actual byte corruptions.

We also evaluate our approach in more complex experimental settings. Section V-E evaluates CARE over one link. The sender transmits packets with 90 bytes payload to the receiver with a period of 512ms. We periodically start and stop iperf to cause different levels of interference to the link, i.e., the interference is present for t_i, absent for t_i, present for t_i, ... A large t_i represents a stable environment while a small t_i represents a dynamic environment in which the interference is often changing.

Section V-F evaluates CARE in the face of multiple 802.15.4 links. The experiments are conducted in our indoor testbed consisting of 10x2 TelosB nodes (see Fig. 8). Two nodes have a distance of about 0.6m. 10 senders transmit packets with 90 bytes payload to 10 receivers simultaneously, with an interval of 64ms. We conduct the experiment in the presence of WiFi interference. In order to see the impact of multiple 802.15.4 links, we vary the power level so that the transmissions over a single link can be interfered by different levels of interference from 802.15.4.

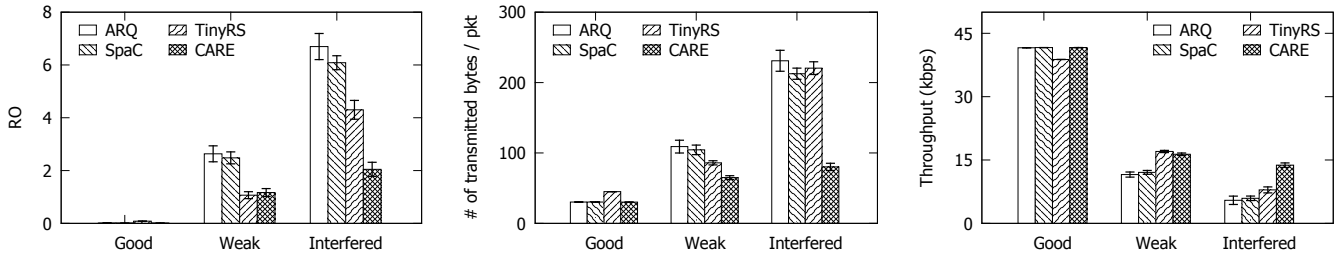
B. Accuracy of Corruption Detection

We employ the corruption detection algorithm described in Section III-A to detect the number of corrupted bytes. We also compare our approach with that employed in REPE [5]. The accuracy is measured as the difference between the detected number of corruptions and the real number of corruptions.

Fig. 9 characterizes the corruption levels under high and low interference. Fig. 10 compares the corruption detection accuracy with REPE. We can see that CARE achieve more accurate detection results.

C. Encoding and Decoding Overhead

We also evaluate the encoding and the decoding overhead of a block on the TelosB nodes with 4MHz MSP430f1611 microcontroller with varying coding rates ($= K/N$). We can see from Fig. 11 that the results are consistent with prior results reported in [3], [15]. We also see that the decoding overhead is much larger than the encoding overhead. To mitigate the impact on the throughput performance, we employ the Selective Repeat mechanism so that the decoding time will



(a) Retransmission overhead (RO) for 30-byte pkts (b) # of retransmitted bytes for 30-byte pkts (c) Peak throughput for 30-byte pkts

(d) Retransmission overhead (RO) for 90-byte pkts (e) # of retransmitted bytes for 90-byte pkts (f) Peak throughput for 90-byte pkts

Fig. 12: Link performance

not defer the next transmission.

Note that we directly employ the RS code library developed by Phil Karn [17] and the decoding performance is not yet optimized. A recent study in [21] shows the encoding/decoding time can be significantly improved. For example, the best case decoding time can be reduced ∼87% compared with the traditional approach we employed.

D. Performance of the CARE Retransmission Protocol

We compare state-of-the-arts protocols which can be implemented on current TelosB nodes. ARQ is the default retransmission protocol in TinyOS. SPaC is a packet combining protocol with Hamming(8,4) coding. TinyRS is a transmission protocol using FEC with RS(15,10,4).

Two TelosB nodes communicate at the maximum transmission rate with a maximum retransmission threshold of 30. Each experiment lasts for one hour. We compare these protocols in terms of three performance metrics including retransmission overhead (RO, see Section IV), the number of transmitted bytes (normalized to a packet), and the throughput. Fig. 12 shows the results for two different packet payload sizes of 30 bytes and 90 bytes. In good link conditions, TinyRS with FEC is less efficient since it transmits unnecessary redundancy overhead. In weak link conditions, both ARQ and SPaC are less efficient. In terms of RO and throughput, TinyRS achieves the best performance since it reduces the retransmission rounds with the upfront redundancy added to the packet. The performance of CARE is close to TinyRS in terms of RO and throughput. The transmission overhead of CARE is slightly less than TinyRS. In interfered link conditions, CARE achieves significantly better performance than all other approaches.

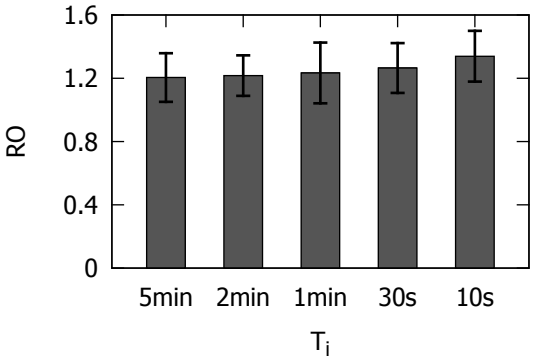


Fig. 13: Impact of changing interference

E. Impact of Changing Interference

We introduce different levels of interference to a link by changing the interval t_i to start/stop iperf. A large t_i represents a stable environment while a small t_i represents a dynamic environment in which the interference is often changing. We conduct each experiment 10 times and the results are shown in Fig. 13 with varying t_i. We can see that the increase of the retransmission overhead (RO) is not significant. For example, the retransmission overhead with $t_i = 10$s increases by 8.9% compared with that of $t_i = 5$ min. This result implies that CARE can quickly adapt to environmental dynamics.

F. Impact of Multiple 802.15.4 Links

We conduct the experiment in our indoor testbed. We set the power level from 1 to 5. A low value of power level indicates a low level of interference from 802.15.4 while a high value of power level indicates a high level of interference from 802.15.4. In all the experiments, WiFi interference is present.

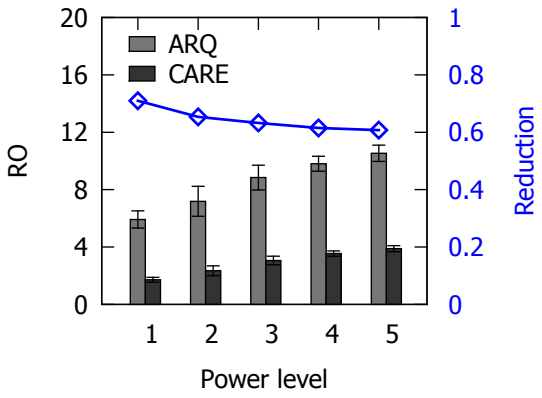


Fig. 14: Impact of multiple 802.15.4 links

We repeat each experiment 10 times. Fig. 14 compares CARE with ARQ in terms of the retransmission overhead (RO) by varying the power level. We can see that the performance of ARQ and CARE degrades with increasing 802.15.4 interference. Compared with ARQ, CARE reduces the retransmission overhead. The percentage of reduction monotonically decreases from 71% to 63%, indicating that CARE still performs well even in the presence of multiple 802.15.4 links.

VI. Conclusion

In this paper, we investigate the problem of corruption-aware retransmission with adaptive coding in commercial-off-the-shelf (COTS) low-power devices like TelosB sensor nodes. We propose an accurate corruption detection algorithm for identifying the number of corrupted bytes in a packet based on the RSSI time series during packet reception. Based on the corruption level, we design an adaptive coding scheme based on Reed Solomon (RS) codes. We carefully select the coding parameters so as to optimize the network performance on resource-constrained low-power devices. Finally, we design and implement CARE, a Corruption-Aware REtransmission protocol by incorporating corruption detection and adaptive coding. We conduct extensive experiments based on COTS low-power devices. Results show that CARE significantly improves the throughput performance for weak and highly interfered links while incurring no additional overhead for good links.

There are multiple future directions to explore. First, we would like to devise accurate corruption detection algorithms for weak links, possibly utilizing recent advances in error estimating codes [22], [23]. Second, we would like to develop fast encoding and decoding algorithms for RS codes [21].

Acknowledgment

We sincerely thank our shepherd Patrick P. C. Lee and the ICNP reviewers for their valuable comments and feedback. This work is supported by the National Science Foundation of China under Grant No. 61472360, Zhejiang Provincial Platform of IoT Technology (2013E60005), Zhejiang Commonweal Project (2015C33077).

References

[1] H. Dubois-Ferri‘ere, D. Estrin, and M. Vetterli, “Packet Combining in Sensor Networks,” in *Proc. of ACM SenSys*, 2005.
[2] T. Mandel and J. Mache, “Practical Error Correction for Resouce-Constrained Wireless Networks: Unlocking the Full Power of the CRC,” in *Proc. of ACM SenSys*, 2013.
[3] 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.
[4] F. Hermans, O. Rensfelt, T. Voigt, E. Ngai, L.-A. Nord’en, and P. Gunningberg, “SoNIC: Classifying Interference in 802.15.4 Sensor Networks,” in *Proc. of ACM/IEEE IPSN*, 2013.
[5] J.-H. Hauer, A. Willig, and A. Wolisz, “Mitigating the Effects of RF Interference through RSSI-Based Error Recovery,” in *Proc. of EWSN*, 2010.
[6] R. K. Ganti, P. Jayachandran, H. Luo, and T. F. Abdelzaher, “Datalink Streaming in Wireless Sensor Networks,” in *Proc. of ACM SenSys*, 2006.
[7] 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 NSDI*, 2010.
[8] K. Jamieson and H. Balakrishnan, “PPR: Partial Packet Recovery for Wireless Networks,” in *Proc. of ACM SIGCOMM*, 2007.
[9] M. Vutukuru, H. Balakrishnan, and K. Jamieson, “Cross-Layer Wireless Bit Rate Adaptation,” in *Proc. of ACM SIGCOMM*, 2009.
[10] S. Sen, N. Santhapuri, R. R. Choudhury, and S. Nelakuditi, “AccuRate: Constellation Based Rate Estimation in Wireless Networks,” in *Proc. of USENIX NSDI*, 2010.
[11] J. Zhang, H. Shen, K. Tan, R. Chandra, Y. Zhang, and Q. Zhang, “Frame Retransmissions Considered Harmful: Improving Spectrum Efficiency Using Micro-ACKs,” in *Proc. of ACM MobiCom*, 2012.
[12] J. Ou, Y. Zheng, and M. Li, “MISC: Merging Incorrect Symbols using Constellation Diversity for 802.11 Retransmission,” in *Proc. of IEEE INFOCOM*, 2014.
[13] K. C.-J. Lin, N. Kushman, and D. Katabi, “ZipTx: Harnessing Partial Packets in 802.11 Networks,” in *Proc. of ACM MobiCom*, 2008.
[14] Y. Wu, G. Zhou, and J. A. Stankovic, “ACR: Active Collision Recovery in Dense Wireless Sensor Networks,” in *Proc. of IEEE INFOCOM*, 2010.
[15] 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.
[16] 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.
[17] P. Karn, “Reed solomon coding/decoding package v1.0.” [Online]. Available: http://www.piclist.com/tecHREF/method/error/rs-gp-pk-uoh-199609/index.htm
[18] F. Barac, M. Gidlund, and T. Zhang, “Scrutinizing Bit and Symbol Errors of IEEE 802.15.4 Communication in Industrial Environments,” *IEEE Trans. Instrumentation and Measurement*, vol. 63, no. 7, pp. 1783–1794, 2014.
[19] M. Rossi, N. Bui, G. Zanca, L. Stabellini, R. Crepaldi, and M. Zorzi, “SYNAPSE++: Code Dissemination in Wireless Sensor Networks Using Fountain Codes,” *IEEE Transactions on Mobile Computing*, vol. 9, no. 12, pp. 1749–1765, 2010.
[20] F. Hermans, H. Wennerström, L. McNamara, C. Rohner, and P. Gunningberg, “All is not Lost: Understanding and Exploiting Packet Corruption in Outdoor Sensor Networks,” in *Proc. of EWSN*, 2014.
[21] M. S. Srouji, T. Bonny, and J. Henkel, “High-speed Encoding/Decoding Techniques for Reliable Data Transmission in Wireless Sensor Networks,” in *Proc. of IEEE SECON*, 2014.
[22] B. Chen, Z. Zhou, Y. Zhao, and H. Yu, “Efficient Error Estimating Coding: Feasibility and Applications,” in *Proc. of ACM SIGCOMM*, 2010.
[23] W. Dong, P. Zhang, C. Chen, and J. Bu, “Exploiting Error Estimating Codes for Packet Length Adaptation in Low Power Wireless Networks,” *IEEE Transactions on Mobile Computing*, vol. 8, no. 15, pp. 1601–1614, 2015.