

Forward Error Correction in Sensor Networks

Jaemin Jeong, Cheng-Tien Ee
EECS Department, University of California, Berkeley
Berkeley, California 94720, USA
{jaemin,ct-ee}@eecs.berkeley.edu

Abstract—From the empirical observation that packet errors are infrequent and multiple-bit errors rarely occur, we have implemented a few versions of error-correction code (ECC) that correct single-bit or double-bit errors on Chipcon CC1000 radio-based wireless sensor nodes. We have evaluated that our ECC implementation reduces packet error rate close to zero in outdoor environment where most errors are single-bit. In indoor environment where increased multiple-bit errors occur, our ECC implementation has a bit higher packet error rate, but its packet error rate is still lower than that of not using ECC.

Index Terms—Forward Error Correction, Error Correction Code, Wireless Sensor Networks

I. INTRODUCTION

In any network, there are two basic methods to recover erroneous packets. One way is to use automatic repeat request (ARQ), where a receiver is supposed to acknowledge the message from the sender and the sender retransmits the message if it is not acknowledged within a certain timeout. Another way for error recovery is forward error correction (FEC), where a sender sends redundant information along with the data bits, and a receiver receives the data bits and can correct possible errors using the redundant information. Since, in wireless sensor networks, packets are commonly broadcast over shared channel and forwarded over multiple hops, using FEC is preferable because it can reduce the need to retransmit data packets, thereby reducing the power consumed in the process.

Various encoding schemes, such as Reed-Solomon codes [5] and LT codes [4], are available and can be readily implemented. However, the choice of the encoding scheme depends on the application and the error characteristics of the wireless channel. For wireless sensor networks, most data transmitted are that of readings taken periodically. For instance, the sensors might be gathering data every 5 minutes on the temperature distribution of the geographical region over which they are scattered, sending just one packet per node. This is in contrast to the Internet, where the transfer of data can be rapid and frequent as in the case of streaming video. Also, the encoding scheme should be of relatively low complexity, since a typical sensor currently has low processing power and a small memory. Since Reed-Solomon and LT codes are relatively complex and require high processing power and storage, they may not be ideal for our application. Furthermore, packets are sent between long intervals, thus encoding schemes that need to work with several packets at a time may result in an increase in latency to an unacceptable level. Thus, depending on the

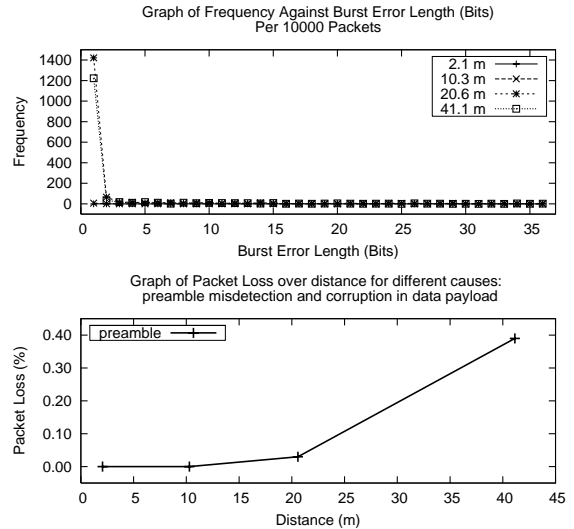


Fig. 1. Measurement was taken with Chipcon CC1000 radio without using error-correction code. A sender node sends the same encoded packet 10,000 times with the received packets logged into PC at the receiver end. This experiment was conducted outside Computer Science building in UC Berkeley. (a) Histogram of burst bit error length, (b) Packet loss rate for different causes

error characteristics, we would prefer using a simpler encoding scheme.

From a preliminary measurement in Figure 1(a), we can observe that most bit errors are single-bit or double-bit errors and burst errors are present but rare. Thus, it is likely that an encoding scheme that corrects single and double-bit errors can reduce a significant portion of the errors. Another measurement in Figure 1(b) shows that the packet loss due to preamble misdetection P_{pre} is very small. For example, at a distance of 41.1m, $P_{pre} = 0.4\%$. Thus, we confirmed that preamble misdetection does not cause significant packet loss and most of the packet losses are due to errors in other parts of the packet.

Based on these findings, we have implemented a few versions of error-correction code (ECC) that corrects single-bit or double-bit errors in each data byte of the payload at MAC layer using Mica2dot sensor node with ChipCon CC1000 radio as our target platform. We have evaluated that our ECC implementation reduces packet rate close to zero in outdoor environment where most errors are single-bit. A version of single-bit error-correction code SECDED(13,8) has packet error rate 0.00% in outdoor experiment compared to 0.22%

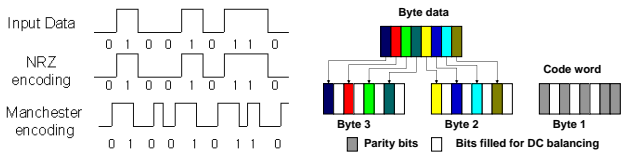


Fig. 2. Comparison of NRZ encoding and Manchester encoding

Fig. 3. Bit encoding of RFM radio

for not using ECC. In indoor environment where increased multiple-bit errors occur, our ECC implementation has a bit higher packet error rate (0.02% for SECDED(13,8)), but its packet error rate is still lower than that of not using ECC (2.31% for non-ECC).

The rest of this paper is organized as follows: we review previous implementation of error-correction code for wireless sensor networks in section II; we describe the theory on error-correction code and its implementation for wireless sensor networks in section III and section IV; then, we show the experimental results in section V and conclude the paper in section VI.

II. BACKGROUND

Among various error-correction codes, we consider single-bit correction code and double-bit correction code in favor of its small memory footprint and simplicity. While other error-correction codes, such as Reed-Solomon code [5] and LT code [4], are known to have better error-correction capability. These codes require more complex computation and larger memory space.

A version of single-bit error-correction code [3] has been used for Mica sensor platform with RFM TR1000 radio. [3] uses more bits than needed for data and parity bits because it DC-balances as well as encoding parity bits for the data. As a bit-modulation scheme, RFM TR1000 radio employs NRZ encoding. This bit-encoding scheme has a problem that the transceiver may not correctly interpret the transmitted digital signal when there is a long sequence of 0s or 1s. [3] avoids this problem by inserting additional bits among data and parity bits. On the other hand, the Chipcon CC1000 radio transceiver for Mica2 and Mica2dot platforms addresses bit-modulation in a better way by supporting Manchester encoding at hardware level. Manchester encoding avoids DC-balancing by generating same number of chips of 0 and 1 for any bit sequence (Figure 2). Thus, the MICA RFM radio error-correction code is unsuitable for Chipcon CC1000 radio that does not require additional bit stuffing (Figure 3).

III. THEORY

In this section, we elaborate on the basics of coding theory that is used in our implementation. In particular, we concentrate on *linear block codes* over field $GF(2)$ because they are simple and efficient in practice. The general process of encoding, transmission and decoding of the message is shown in Figure 4.

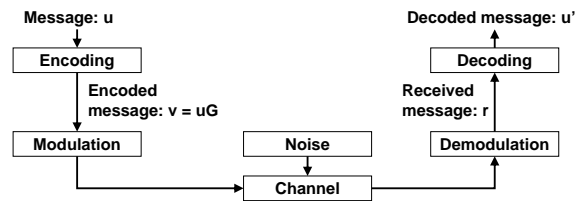


Fig. 4. Encoding, transmission and decoding of message

The encoding of message \mathbf{u} into codeword \mathbf{v} can be achieved by multiplying the message \mathbf{u} with the generation matrix \mathbf{G} . For data of width k -bits, \mathbf{G} is of the form $[\mathbf{I}_k : \mathbf{C}]$, where \mathbf{I}_k is the $k \times k$ identity matrix and \mathbf{C} is the $k \times r$ binary matrix.

On the receiver end, a *syndrome* is calculated for detecting and possibly correcting errors. The *syndrome* \mathbf{s} is calculated from the received signal \mathbf{r} and the *parity matrix* \mathbf{H} . The *parity matrix* \mathbf{H} is constructed from the generator matrix \mathbf{G} and is of the form $\mathbf{H} = [\mathbf{C}^T : \mathbf{I}_r]$, where r is the number of parity bits. Denoting the *error vector* by \mathbf{e} , we have

$$\mathbf{s} = \mathbf{r}\mathbf{H}^T = (\mathbf{v} + \mathbf{e})\mathbf{H}^T = \mathbf{u}\mathbf{G}\mathbf{H}^T + \mathbf{e}\mathbf{H}^T = \mathbf{e}\mathbf{H}^T$$

Here, $\mathbf{G}\mathbf{H}^T = [\mathbf{I}_k : \mathbf{C}][\mathbf{C}^T : \mathbf{I}_r]^T = \mathbf{C} + \mathbf{C} = \mathbf{0}$ (addition is in $GF(2)$). The non-zero \mathbf{s} implies the location of an error. Depending on the capability of the error-correction code, the non-zero syndrome is compared with a row or a sum of rows of \mathbf{H}^T . If there are such rows, the correct codeword can be got by flipping corresponding bits in the received signal.

The receiver can decode the corrected codeword by solving the equation $\mathbf{v} = \mathbf{u}\mathbf{G}$. Especially, for a systematic code where first k -columns of generator matrix \mathbf{G} form an identity matrix, \mathbf{u} is just first k -bits of \mathbf{v} .

A. Odd-weight-column code

Odd-weight-column code can correct single-bit errors and detect double-bit errors (SECDED) [2]. As its name implies, each column of the parity matrix \mathbf{H} for an odd-weight-column code has an odd number of 1s.

To construct an odd-weight-column code for an input of k data bits, the parity matrix \mathbf{H} should include a sufficient number of parity bits r so that the number of columns of \mathbf{H} is at least $k + r$. For example, $r = 5$ parity bits are needed to recover $k = 8$ bits of data, and $r = 6$ parity bits are needed to recover $k = 24$ bits of data.

The columns of \mathbf{H} are constructed as follows:

- The last r columns of \mathbf{H} form the identity matrix \mathbf{I}_r .
- The first k columns of \mathbf{H} are chosen from r -row odd-weight-column vectors except the ones used in \mathbf{I}_r .

For example, \mathbf{G} , \mathbf{H} for $k = 8$, $r = 5$ are:

$$\mathbf{G} = [\mathbf{I}_8 : \mathbf{C}] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{H} = [\mathbf{C}^T : \mathbf{I}_5] = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We now give an example of how data is encoded, and how the received message can be corrected. Let the message being sent be $\mathbf{u} = [0100\ 0010]$. The encoded message \mathbf{v} is therefore

$$\mathbf{v} = \mathbf{u}\mathbf{G} = [0100\ 0010\ 10111]$$

Assume that the second bit of the encoded message is inverted due to the noise in the wireless channel. Then, received message \mathbf{v}' is $[0000\ 0010\ 10111]$. Multiplying the received message by the transpose of the parity matrix \mathbf{H} , we get the syndrome \mathbf{s} as follows:

$$\mathbf{s} = \mathbf{v}'\mathbf{H}^T = [0\ 1\ 0\ 1\ 1]$$

We note that the syndrome is the same as the second row of \mathbf{H}^T , which implies that second bit of the codeword is inverted. Thus, we can get correct codeword $\mathbf{v} = [0100\ 0010\ 10111]$. Since the generator matrix \mathbf{G} is a systematic code, the data bits can be decoded by taking first- k bits of the codeword. Thus $\mathbf{u} = [0100\ 0010]$.

B. Double error correction code

In this section, we describe a (16,8) systematic, quasi-cyclic code that can correct 2-bit errors, as well as detect 3-bit errors.

A quasi-cyclic code is defined to be one whereby a cyclic shift of n_0 digits always produces another codeword [1]. Thus, encoding of the message can be done serially using linear feedback shift registers, or in parallel if low latency is required. Our implementation checks for and corrects single-bit and double-bit errors. If any errors are detected, we simply signal to the higher layer that an error has occurred.

Since the syndrome value is represented by 8-bits, this implies that there are a total of 256 possible syndrome values. Of these, the zero syndrome ($\mathbf{0}$) is used when there are no errors in the received message, 16 syndrome values are mapped to single-correctable errors, and 120 syndrome values are mapped to double-correctable errors.

The generator matrix \mathbf{G} is given by

$$\mathbf{G} = [\mathbf{I}_8 : \mathbf{C}] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Note that the minimum Hamming distance between any pair of rows is 5. The parity matrix \mathbf{H} is given by

$$\mathbf{H} = [\mathbf{C}^T : \mathbf{I}_8] = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We now give an example of how data is encoded, and how the received message can be corrected. Let the message being sent be $\mathbf{u} = [0100\ 0010]$. The encoded message \mathbf{v} is therefore

$$\mathbf{v} = \mathbf{u}\mathbf{G} = [0100\ 0010\ 1001\ 1100]$$

Assume that the second and third bits are inverted due to noise in the wireless channel. Then, the received message is $\mathbf{v}' = [0010\ 0010\ 1001\ 1100]$. Multiplying the received message by the transpose of the parity matrix \mathbf{H} , we get the syndrome \mathbf{s} as follows:

$$\mathbf{s} = \mathbf{v}'\mathbf{H}^T = [1010\ 1111]$$

We note that the syndrome obtained is the exclusive-or of the 2nd and 3rd rows of \mathbf{H}^T , thus at the receiver end, 2nd and 3rd bits are inverted to correct the errors $\mathbf{v} = [0100\ 0010\ 1001\ 1100]$. Since the generator matrix \mathbf{G} is a systematic code, we can get data bits by taking first 8 bits of the corrected codeword $[0100\ 0010]$.

IV. IMPLEMENTATION

While an error-correction code (ECC) can be located in any layer in the network stack, we have decided to have the error-correction code module in MAC layer as in Figure 5. This approach does not change the interface to the application layer; thus, any applications written for non-ECC version of MAC can run without source code modification. As for error-correction code, three different versions were written:

- `SecDedEncodingOneByte`: odd-weight-column code with 8-bit data and 13-bit codeword (SEDED (13,8)).
- `SecDedEncodingThreeByte`: odd-weight-column code with 24-bit data and 30-bit codeword (SEDED (30,24)).
- `DecTedEncoding`: quasi-cyclic code with 8-bit data and 16-bit codeword (DECTED (16,8)).

As is shown in Figure 5, our error-correction code implementation interacts with MAC layer through the interface

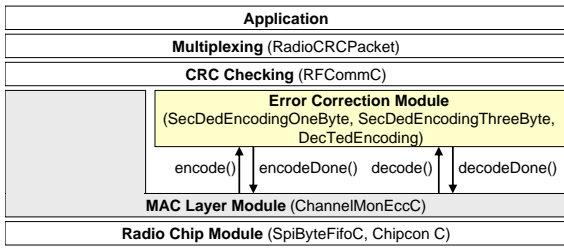


Fig. 5. Interfacing ECC module with network stack

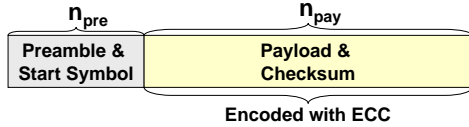


Fig. 6. Packet Length

RadioEncoding. With a packet to be sent, the MAC layer module `ChannelMonEccC` calls the method `encode` for each byte of data in the packet. After a sufficient number of input data bytes have been received, the input data bytes are encoded by an internal encoding function `radio_encode_thread` and codeword is passed to the `ChannelMonEccC` through `encodeDone` event. With data bytes being passed by the physical layer, the MAC layer module `ChannelMonEccC` calls the method `decode` for each byte of its received packet. After a sufficient number of input data bytes have been received, the received data bytes are decoded by the internal decoding function `radio_decode_thread` and the original data bytes are sent to `ChannelMonEccC` through `decodeDone` event.

The internal encoding function `radio_encode_thread` calculates parity bits by comparing each bit of input data bytes. This corresponds to calculating $m\mathbf{G}$, where m is the message and \mathbf{G} is the generator matrix. The internal decoding function `radio_decode_thread` calculates the syndrome s by looking at each bit of the received data bits. This is equivalent to $r\mathbf{H}^T$ where r is the received data and \mathbf{H}^T is the transpose of the parity matrix. A non-zero syndrome implies an error and the position of bit errors can be found by comparing the syndrome with column vectors of \mathbf{H} matrix. We made this lookup fast by using an array that maps any possible syndrome value to the error bit position.

In general, the MAC layer of WSN consists of the following fields: preamble, start symbol, payload and checksum (Figure 6). When an error-correction code is used, the data bytes in the payload and checksum are encoded into codeword while the data bytes for the preamble and start symbol are not encoded. Then, we can estimate the transmission overhead of an error-correction code for the following parameters:

- n_{pre} : length of preamble and start symbol (bytes)
- n_{pay} : length of payload and checksum (bytes)
- r_{ecc} : length of codeword for one-byte data
- n_{ecc} : data bytes transmitted with ECC (bytes)
- n_{no_ecc} : data bytes transmitted without ECC (bytes)

TABLE I
TRANSMISSION OVERHEAD FOR ECC IMPLEMENTATION

	SECEDED(8,13)	SECEDED(30,24)	DECTED(16,8)
r_{ecc}	2B/1B	4B/3B	2B/1B
Overhead	64.3%	21.4%	64.3%

$$\begin{aligned}
 \text{Overhead} &= \frac{n_{ecc} - n_{no_ecc}}{n_{no_ecc}} \\
 &= \frac{(n_{pre} + n_{pay} \cdot r_{ecc}) - (n_{pre} + n_{pay})}{n_{pre} + n_{pay}} \\
 &= \frac{n_{pay}}{n_{pre} + n_{pay}} \cdot (r_{ecc} - 1)
 \end{aligned}$$

Since the TinyOS distribution has $n_{pre} = 20$ and $n_{pay} = 36$ for CC1000 radio, the overhead for our ECC implementation can be calculated as Table I.

As well as the error-correction code implementation on the mote side, we have also implemented a few utility programs to measure the performance of the error-correction code:

- **Wireless sensor node application:** This application sends packets at maximum rate by sending the next packet whenever the previous packet finishes transmission. This program sends the same packet and the content of the packet is predetermined so that the receiver node can tell whether the packet is corrupted without using the checksum.
- **Data logging program:** This java application receives a fixed size packet from the receiver node and writes the packet into a file.
- **Data analysis program:** The analysis program compares the logged data with the predetermined contents, and computes the following statistics: percentage of packets with corrupted bits, frequency of burst error length, number of bits corrupted per packet, and consecutive packet losses.

V. EXPERIMENT

A. Setup

To see the effectiveness of our ECC implementation, we conduct experiments for the following schemes: **NO FEC**, **SECEDED (13,8)**, **SECEDED (30,24)** and **DECTED (16,8)**.

We perform the tests using the same pair of sender and receiver nodes for different schemes. This removes possible effects due to the manufacturing deviation amongst the sensor nodes. The transmitter node sends the same packets 5,000 times for each iteration of the experiments. The data received at the receiver node are logged in a PC for further processing. Since the received packets may be from other nodes than the destined transmitter, the data logging program filters out packets from other nodes.

For the experiments, we perform two kinds of tests: (1) outdoor test to determine the effects of distance on packet errors, (2) indoor tests to see the effects of environment noise on packet errors. In the outdoor test, the sender and the receiver

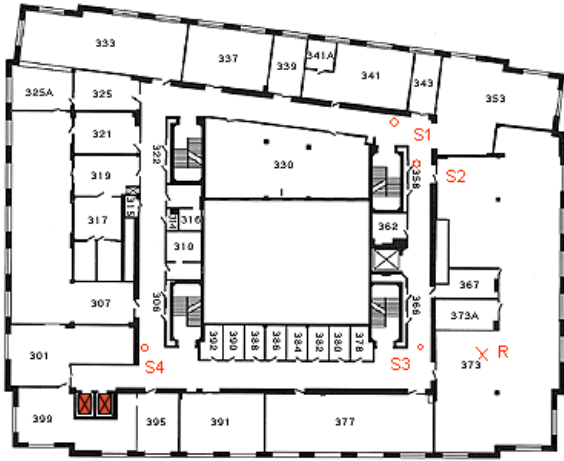


Fig. 7. Locations of indoor tests

nodes are placed 182.9m apart with direct line of sight along the South Hall Drive in the central UC Berkeley campus. In indoor tests, the receiver node is placed in an office room on the 3rd floor in UC Berkeley Electrical Engineering building and the sender node was placed at four different places along the corridor on the same floor of the building in turn. The locations for the receiver and the sender nodes are shown in Figure 7.

B. Results

Our ECC implementation helps reduce the packet drop rate both outdoors and indoors. But, for the case of indoor tests, using ECC does not reduce the packet drop rate close to zero as it does in the outdoor test. This is because the frequency of multiple-bit errors is higher in indoor tests and our ECC can handle only single-bit or double-bit errors.

Table II shows the packet drop rate in the outdoor test. With our ECC implementation, the packet drop rate is 0.00% to 0.08% depending on ECC implementation. With no ECC, we have a bit higher error rate of 0.22%. Table III summarizes the packet drop rate for the indoor tests. The packet drop rate for using ECC is 0.02% to 1.32% on average compared to 2.31% for not using ECC. Figure 8(a) through Figure 12(a) show the detailed cause of packet drop rate (1-bit, 2-bit or multiple-bit).

Among different versions of ECC implementation, SECDED (13,8), which encodes 8-bit into 13-bit codeword, produces smallest packet drop rate. Another version of single-bit correction code, SECDED (30,24), which encodes 24-bit data into 30-bit codeword, does not perform as well as SECDED (13,8) in terms of packet drop rate. This is because it corrects up to 1-bit error over 24-bits compared to over 8-bits for SECDED (13,8). Thus, its error-correction capability is weaker than SECDED (13,8) although it has space saving advantage. A double-bit correction code, DECTED (16,8), is no more effective than SECDED (13,8) because most errors are single-bit or multiple-bit errors.

The distribution of burst bit errors (Figure 8(b) through

TABLE II
PACKET LOSS RATE AND RUNNING TIME IN OUTDOOR TESTS

	No FEC	SECDED (13,8)	SECDED (30,24)	DECTED (16,8)
Running time	534.5s	682.3s	581.9s	682.6s
Total packets	4,998	4,999	5,000	4,999
Dropped packets	11 (0.22%)	0 (0.00%)	0 (0.00%)	4 (0.08%)
- 1-bit error	6 (0.12%)	0	0	0
- 2-bit error	0	0	0	0
- multibit error	5 (0.10%)	0	0	4 (0.08%)

TABLE III
PACKET DROP RATE IN INDOOR TESTS

	No FEC	SECDED (13,8)	SECDED (30,24)	DECTED (16,8)
Indoor Location 1				
Total pkts	4,651	4,648	4,729	4,528
Pkt drops	394 (8.47%)	0 (0.00%)	26 (0.55%)	0 (0.00%)
1-bit	335 (7.20%)	0	26 (0.55%)	0
2-bit	19 (0.41%)	0	0	0
multibit	40 (0.86%)	0	0	0
Indoor Location 2				
Total pkts	4,779	4,552	4,758	4,540
Pkt drops	29 (0.61%)	0 (0.00%)	20 (0.42%)	0 (0.00%)
1-bit	24 (0.50%)	0	20 (0.42%)	0
2-bit	3 (0.06%)	0	0	0
multibit	2 (0.04%)	0	0	0
Indoor Location 3				
Total pkts	4,769	4,510	4,787	4,390
Pkt drops	0 (0.00%)	0 (0.00%)	170 (3.55%)	0 (0.00%)
1-bit	0	0	17 (0.36%)	0
2-bit	0	0	1 (0.02%)	0
multibit	0	0	152 (3.18%)	0
Indoor Location 4				
Total pkts	4,787	4,474	4,757	4,394
Pkt drops	8 (0.17%)	3 (0.07%)	35 (0.74%)	232 (5.28%)
1-bit	4 (0.08%)	0	21 (0.44%)	8 (0.18%)
2-bit	2 (0.04%)	1 (0.02%)	1 (0.02%)	10 (0.23%)
multibit	2 (0.04%)	2 (0.04%)	13 (0.27%)	214 (4.87%)
Average				
Pkt drops	2.32%	0.02%	1.19%	1.32%

Figure 12(b)) and burst packet losses (Figure 8(c) through Figure 12(c)) and give us some ideas on how to handle burst errors. While burst errors can happen in long succession of bits, the frequency of multiple successive packet drops is low. Most packets are dropped in succession of one or two packets and drops of four or more successive packets are rare. This implies that when retransmission is employed in combination with ECC, a small number of retransmissions would be enough.

VI. CONCLUSION

Forward error correction is a way of correcting packets by transmitting additional information bits. We have implemented and tested different error-correction codes on the Chipcon CC1000 radio. Due to the constraints of low power consumption and small form factor, the error-correction codes have been designed to be simple and can correct single-bit or double-bit errors. Our ECC implementations are effective when the bit error rate is not high and most errors are single bit. When most of the errors are bursty, our codes are not as effective in reducing packet losses. We expect that we can

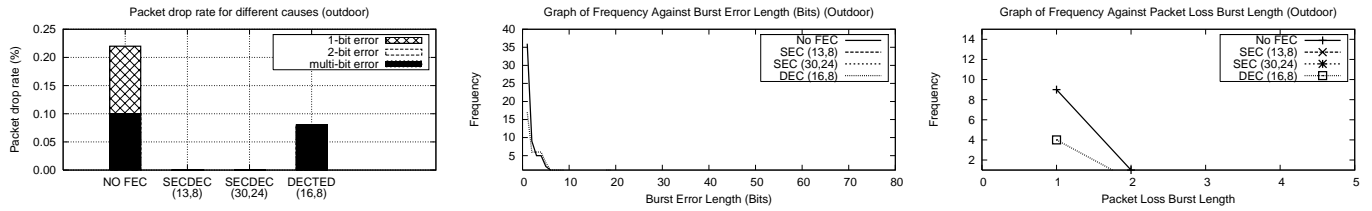


Fig. 8. Outdoor Experiment Results (600ft). (a) Packet drop cause, (b) Distribution of burst bit errors, (c) Distribution of burst packet errors

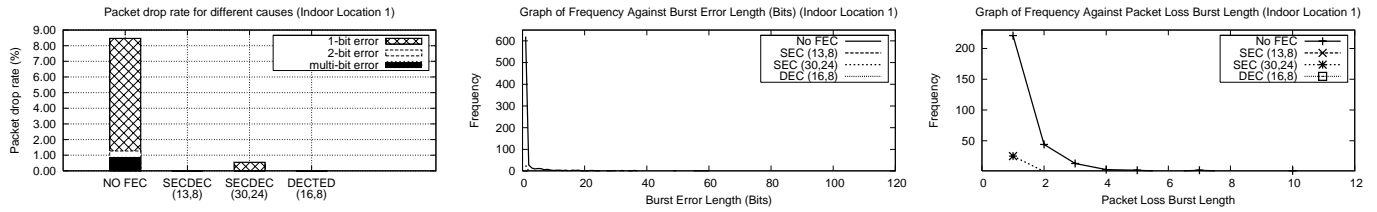


Fig. 9. Indoor Experiment Results (Location 1). (a) Packet drop cause, (b) Distribution of burst bit errors, (c) Distribution of burst packet errors

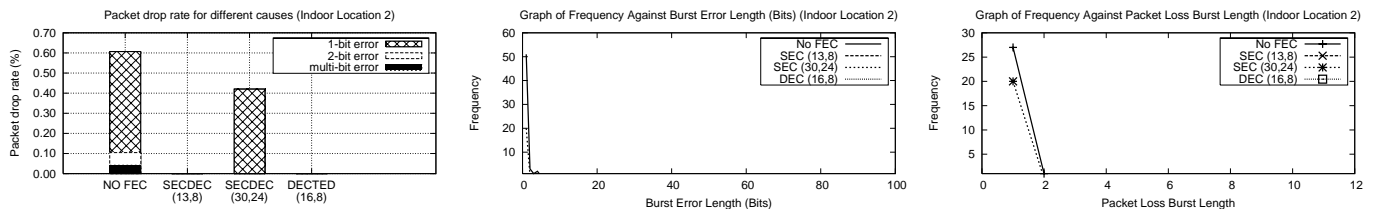


Fig. 10. Indoor Experiment Results (Location 2). (a) Packet drop cause, (b) Distribution of burst bit errors, (c) Distribution of burst packet errors

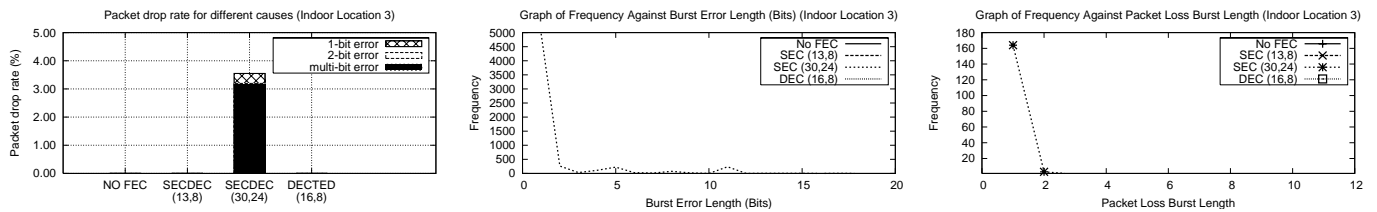


Fig. 11. Indoor Experiment Results (Location 3). (a) Packet drop cause, (b) Distribution of burst bit errors, (c) Distribution of burst packet errors

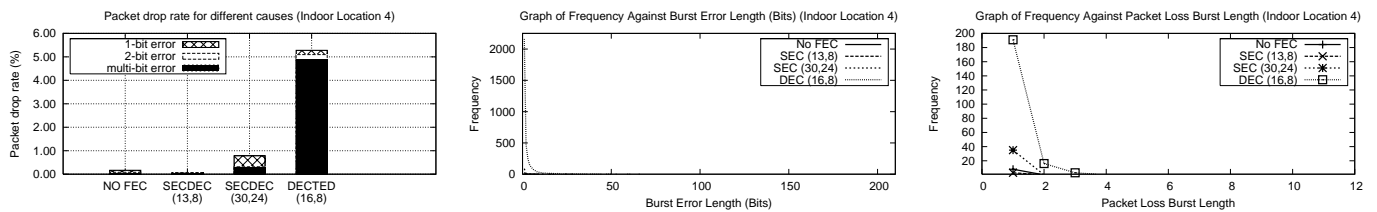


Fig. 12. Indoor Experiment Results (Location 4). (a) Packet drop cause, (b) Distribution of burst bit errors, (c) Distribution of burst packet errors

achieve lower error rates in such cases by using correction schemes that can correct burst errors. However, such schemes are likely to be complex and require high processing power as well as large amounts of storage memory.

REFERENCES

[1] T. A. Gulliver and V. K. Bhargava. A systematic (16,8) code for correcting double errors and detecting triple-adjacent errors. *IEEE Transactions on*

Computers, 42(1):109–112, 1993.

- [2] M. Y. Hsiao. A class of optimal minimum odd-weight-column sec-ded codes. *IBM J. Res Develop*, 14(4), July 1970.
- [3] N. Lee, P. Levis, and J. Hill. Mica high speed radio stack. *UC Berkeley Tech Report UCB/ERL M02/34*, 2002. <http://www.tinyos.net/tinyos-1.x/doc/stack.pdf>.
- [4] M. Luby. Lt codes. In *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*, page 271, 2002.
- [5] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *J. Soc. Indust. Appl. Math*, 8:300–304, 1960.