Selected CRC Polynomials Can Correct Errors and Thus Reduce Retransmission

Travis Mandel, Jens Mache

***Abstract*- For wireless sensor networks, minimizing communication is crucial to improve energy consumption and thus lifetime. Whereas the standard way to deal with transmission errors is retransmission (automatic repeat request ARQ), in this paper we investigate an alternative: correcting bit errors using Cyclic Redundancy Checks CRCs (which are already used for error detection). Selected CRC polynomials - including CCITT-16 which is used by IEEE 802.15.4 and TinyOS - can correct 1-bit errors in up to 240 bits of data. We present our send-check-confirm (SCC) protocol that reduces retransmission without sacrificing reliability since corrections are validated. In addition, we list 64 16-bit candidate CRC polynomials that can correct for 1- and 2-bit errors in less than 240 bits of data.**

Index Terms*- Wireless Sensor Networks, Cyclic Redundancy Check (CRC), Error Correction, Reliability, Network Protocol, Low Power Comsumption

I. INTRODUCTION

Error detection using Cyclic Redundancy Checks(CRCs) is implemented in most communication protocols. However, relatively little work has been done in using these CRCs for the *correction* of bit errors. Correcting bit errors - instead of retransmitting the whole packet - improves energy consumption and thus lifetime of wireless sensor networks, given that communication is extremely expensive when compared to computation [15]. Correcting 1- and 2-bit errors is very worthwhile, considering that in a study of IEEE 802.15.4, around 50% of errors are isolated to a single bit, and 60% of burst errors are only two bits long [2].

The rest of the paper is organized as follows: We will present background and related work in section 2, Section 3 will discuss single-bit error correction, Section 4 will examine error correction protocols, Section 5 will analyze the performance of our protocol, Section 6 will introduce the motivation and implementation of multiple-bit error correction, followed by discussion and conclusions.

This material is based upon work supported by the National Science Foundation under Grant CNS-0720914. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

T. Mandel is with the School of Computer Science, Carnegie Mellon University Pittsburgh, PA (e-mail: tmandel@andrew.cmu.edu).

J. Mache is with the Computer Science Department, Lewis & Clark College, Portland, OR (e-mail: jmache@lclark.edu)

II. BACKGROUND AND RELATED WORK

A. Cyclic Redundancy Checks

Cyclic redundancy checks use binary polynomial division to detect errors. To compute a CRC one must first pick a generator polynomial G. In practice, there are certain recommended choices of G that increase the error detecting ability of our CRC algorithm. [16] As an example, let G = (x^3+1). We represent the coefficients of the polynomial as bits, so G=1001. We also know that the first bit of G will always be a 1, so we only need to store n=3 bits, 001. Now assume our original data M = 101110. To compute the checksum of M, we must first append n bits to the end to create D=101110000 and take $CRC_G(D)$. This is simply binary polynomial division of G into D.

```
          101011
    1001)101110000
         1001
          101
          000
          1010
          1001
           110
           000
           1100
           1001
            1010
            1001
             011
```

So the $CRC_G(101110000)$ is 011 in binary (3 in decimal). Then our transmitted message M' consists of M concatenated with its CRC, or 101110011.

The receiver takes the CRC of what it receives. If there is no corruption, this CRC will be zero, otherwise, an error is detected. In our example, we take CRC_G (101110011)=0, but if the last bit is corrupted CRC_G (101110010)=1.

B. Related Work

Relatively little work has been done in using CRCs for the correction of bit errors.

Much of the previous work has been concerned with only a single generator polynomial , such as a paper titled *Single Bit Error Correction Implementation in CRC-16 on FPGA* [5], which offers optimizations for single-bit correction that are only applicable for CRC-16-X25. They mention that their results could be easily modified for another 16-bit polynomial, but their method relies heavily on the fact that the generator polynomial has exactly 16 bits.

[4] discusses further hardware-level optimizations for the table used in single-bit error correction (this table is described in section III). They claim their methods work for any received message length and any formal CRC generator polynomial, but do not offer a thorough discussion of single-bit error correction over generalized polynomials and messages.

McDaniel's paper on single-bit error correction [1] gives a much more thorough analysis and is discussed in detail in section III.

Related work regarding reduced retransmission includes [10, 11, 9, 8] and is discussed further in Section VII.

III. Single Bit Error Correction

In [1] McDaniel describes a tabular method for correcting single bit errors given a n+1-bit generator polynomial G. The first step is to precompute an error correction table T for the particular choice of G. To do this, one creates a message Z of length 2^n -1 that is composed entirely of zeros. Then one changes each consecutive bit i of Z to 1 creating z_i, and fills table T such that $T[CRC_G(z_i)]=i$. For example, n=3 and $G=x^3+x+1$ yields z_1=1000000 and $CRC_G(z_1)=5$, z_2=0100000 and $CRC_G(z_2)=7$, $CRC_G(z_3)=6$, $CRC_G(z_4)=3$, $CRC_G(z_5)=4$, $CRC_G(z_6)=2$, $CRC_G(z_7)=1$. Thus T={correct, 7, 6, 4, 5, 1, 3, 2}. Note that the zeroth entry in the table will not be used since a CRC of zero indicates correctness, thus there is no bit to correct. If no two mutations of Z have the same CRC, meaning CRC_G is injective for every z_i, then G is a suitable polynomial for error correction. This means given any message M' (consisting of an original message M of size $L=2^n-1-n$ with an n-bit checksum C_1 appended), one can correct any single bit error.

To correct single-bit errors, one takes the $CRC_G(M')=C_2$. If C_2 is zero, the initial message is correct. Otherwise, $T[C_2]$ is the index at which the single bit error occurred. Continuing the above example, M=1100 yields $C_1=CRC_G(1100)=010$ and M'=M concatenated with C_1=1100010. Incorrect reception of 1101010 yields $C_2= CRC_G(1101010)=3$. Since this CRC is not zero, we examine our table. T[3]=4, which indicates that bit 4 was corrupted. Thus flipping bit four gives us 1100010, the correct message.

A. Generator Polynomials

McDaniel's paper [1] claimed that no commonly used generator polynomials could be used for error correction. This would present some obstacles to the widespread use of error correcting CRCs, as there are certain error detecting properties of most commonly-used generator polynomials [16] that make them more resilient than others to common forms of corruption, such as burst errors. However, McDaniel's claim is untrue. We have verified that the CRC-8-CCITT ($x^8 + x^7 + x^3 + x^2 + 1$) standard does indeed have this error-correcting property. Other polynomials also have this capability with smaller message sizes, see section III.B.2, below.

B. Dealing with Different Message Sizes

McDaniel [1] also claimed that the table T (described above) can handle any message size of $\leq 2^n$-1-n. This is not the case. Continuing the above example, M=11 yields $C_1=CRC_G(11)=101$ and M'=M concatenated with C_1=11101. Incorrect reception of 01101 yields $C_2= CRC_G(01101)=6$. Since this CRC is not zero, we examine our table. T[6]=3, which indicates that bit 3 was corrupted. But flipping bit 3 gives us 01001 which is incorrect.

McDaniel's original method can only handle messages of exactly 2^n-1-n bits. Thus messages using 16 bit error-correcting CRCs must contain $2^{16}-1-16 \approx 8$ KiloBytes of data (whereas TinyOS [7] allows at most 29 data Bytes). Being forced to send messages of exactly this length across the medium would require an unreasonable amount of bandwidth and energy, as increasing the amount of data in each packet requires more of these resources. Fixing message length in this way would also increase the probability that several bits of our message were corrupted. We have devised two solutions to this problem:

1) Bit "Padding"

By padding the message only while calculating the CRC, not transmitting the padded bits, we can avoid such problems. When calculating C_1, we simply iterate 2^n-1-n times through the bits of the message, considering the bit to be zero if we have run off the end of the message. Then when calculating C_2, we iterate through all but the last n bits of the message, add zeros until we reach 2^n-1-n, and then iterate through the last n message bits. For example, if n=3 and M' = 11101, we take $CRC_G(1100101)$ In such a manner we still have to send only L+n bits across the medium, using a minimum of bandwidth, but can accommodate any message of length $L \leq 2^n-1-n$.

2) Initialization

If we know a maximum data length $X<2^n-1-n$, we can make Z of length X+n when we calculating the correction table. Then we only have to bit pad smaller messages to length X. In addition to potentially saving bandwidth, this also saves time, as the CRC method has to consider fewer padding bits. For example, n=3, $G=x^3+x+1$ and X=2 yields $CRC_G(00001)=6$, $CRC_G(00010)=3$, $CRC_G(00100)=4$, $CRC_G(01000)=2$, $CRC_G(10000)=1$, thus T={correct, 5, 4, 2, 3, --, 1, --}, and we can correct any message up to length X=2. Given a maximum data length $X<2^n-1-n$, the table is not completely filled and more generator polynomials are suitable for error correction.

IV. Protocols

The major problem with replacing the original automatic repeat request (ARQ) protocol with a single-bit send-correct (SC) protocol is that it will almost certainly lead to wrong corrections. To see this, assume the message M' has two errors. Then when M' is received, C_2 will be computed and found to be non-zero. Thus C_2 will be looked up in the table, indicating a bad bit R (assuming the table has a mapping for that CRC). When the checksum C is taken of M' with R flipped, C will be zero, since the table was constructed to have that property. This is unacceptable as now we perceive a message as correct that is in fact incorrect: we started with two errors and corrected only one bit. Thus one can only use the SC protocol if one can be certain that all messages have either 0 or 1-bit errors, and no multiple-bit errors.

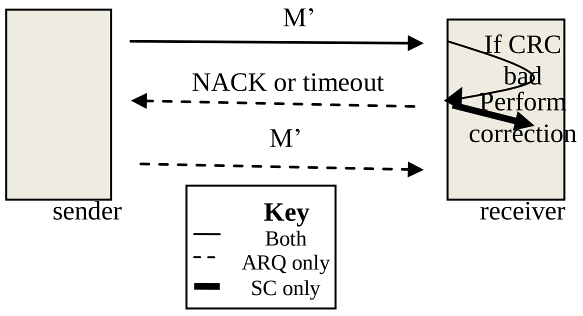

Figure 1. ARQ and SC protocols

A. *Proposed SCC protocol*

In order to resolve this issue, we need some sort of confirmation of the bit that we suspect of being faulty. We propose a single-bit send-check-confirm (SCC) protocol which includes this safeguard. The first step after hypothesizing an error bit R is to check whether or not R falls within the padding area, as if it is, it cannot possibly be right. Otherwise we send to the original sender R’s index concatenated with R’s value. Because this message is much shorter the original (~log n bits), the probability of corruption is much lower. To guard against corruption of this short message, one can either append a CRC for error detection only, or use some simple error correction scheme such as triple modular redundancy. When the original sender receives the message, they examine the original data. If the received bit is indeed correct, the sender sends a very short message indicating this fact. The mechanism is very similar to the HTTP protocol’s “304 Not Modified” message, which is a short message sent in place of a resend of a web document if there is no need to resend the object (i.e. the document has not been modified since it was last accessed) [3].

If the bit is not correct, the sender resends the original message. Thus, if there is a correctable error, we have dramatically shortened the message size, and if not, the resend penalty remains the same as in the original ARQ protocol. Fig. 1 depicts the ARQ and SC protocols, and Fig. 2 depicts our proposed SCC protocol.

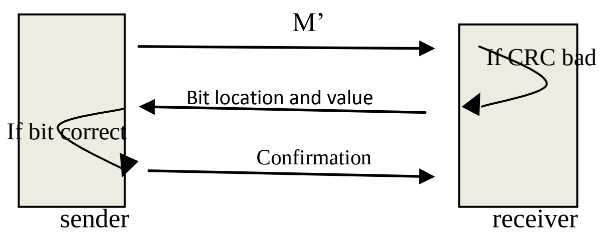

Figure 2. Proposed SCC protocol, in the case of a single bit error. Note that message size is severely decreased, as M’ does not have to be sent a second time.

We implemented the above algorithms, see class diagram in Fig. 3.

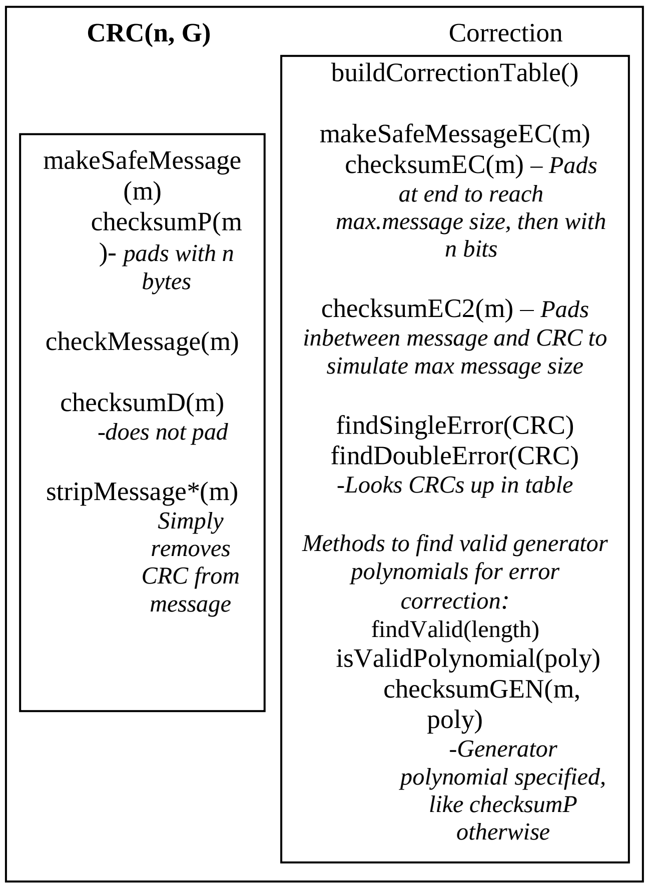
CRC(n, G)
Correction
makeSafeMessage (m)
checksumP(m)- pads with n bytes
checkMessage(m)
checksumD(m) -does not pad
stripMessage*(m) Simply removes CRC from message
buildCorrectionTable()
makeSafeMessageEC(m)
checksumEC(m) – Pads at end to reach max.message size, then with n bits
checksumEC2(m) – Pads inbetween message and CRC to simulate max message size
findSingleError(CRC)
findDoubleError(CRC)
-Looks CRCs up in table
Methods to find valid generator polynomials for error correction:
findValid(length)
isValidPolynomial(poly)
checksumGEN(m, poly)
-Generator polynomial specified, like checksumP otherwise

Figure 3. Class Diagram

V. Performance Analysis:

There is a potential for error in our SCC protocol, in that it is possible to have the server reply that we have corrected all errors when in fact we have not. We call this “undercorrecting” because some errors remain after the initial correction. We will now derive the probability that such an event occurs.

We can model bit errors as a random process in which with some probability p we flip each bit in our data. [14] This is equivalent to X=Binomial(n,p), where n is the number of corruptible bits. In our case, there are n=248 bits (The 232 message bits plus 16 CRC bits) . E[x]=np is between 1 and 1.5 since [2] tells us that around 50% of bit errors are single-bit errors. We would like to know the probability that the server responds that our guess is correct(event A). This is the probability that the table has an entry and the bit in the table is one of the bits that was flipped originally. We do not care about the case where X=0, since we are assuming for argument’s sake an infalliable CRC. If X is 1, we know that the table has an entry (because of how it is constructed) and that entry is correct (since we have only a 1-bit error). happen, If x is greater than 1, we know that the probability that the table is full is independent of the probability that the bit chosen is correct. The probability the table entry chosen has a value is the number of entries in the table over the table size (we have assumed a uniform distribution of errors). The probability the bit is flipped is the number of bits chosen, k, over the total number of possible bits, 248. Hence:

$$\Pr[A] = \Pr[x=1] + \sum_{k=2}^{248} \frac{248}{2^{16}} \times \Pr[x=k] \times \frac{k}{248}$$
$$= \Pr[x=1] + \frac{1}{2^{16}} \sum_{k=2}^{248} k \Pr[x=k]$$
$$= \Pr[x=1] + \frac{1}{2^{16}} (E[k] - 1\Pr[x=1] - 0\Pr[x=0])$$
$$= \Pr[x=1] + \frac{1}{2^{16}} (np - \Pr[x=1])$$

Notice that either x is one and we have corrected our data correctly, or we undercorrect our data. The probability of under-correcting our data is $\frac{1}{2^{16}}(248p - \Pr[x=1])$, which is around $\frac{1}{2^{16}}$ since np is about 1.5 and Pr[x=1] is around .5 . Thus the probability of under-correcting our data is only $\frac{1}{2^{16}}$ with our protocol, which is acceptably small compared to the probability that the CRC fails to detect the error in the first place.

VI. MULIPLE-BIT ERROR CORRECTION

A. Motivation

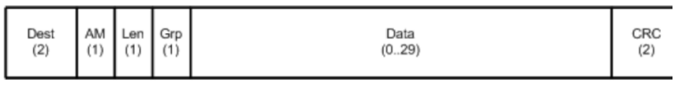

Figure 4. TinyOS Packet Format (sizes in bytes)

As shown in Fig. 4, TinyOS [7] uses a message format that allows at most 29 data Bytes (232 data bits) but 16 CRC bits. We found that the generator polynomial used in the IEEE 802.15.4 standard, $x^{16}+x^{12}+x^5+1$, is usable for single-bit-error correction over this message length. However an 8-bit CRC would already be sufficient to correct single bit errors in up to 2^8-1-8=247 bits of data. For formats such as TinyOS' which use a longer CRC, we can use the extra information provided by the CRC to correct multiple erroneous bits.

B. Implemenation and Analysis

In order to perform multiple bit error correction, one needs a CRC of length n such that 2^n is much larger than the maximum message length L. In fact, to perform k-bit error correction, $2^n \gg \binom{L}{1} + \binom{L}{2} + \cdots + \binom{L}{k}$. The reason for this is that the CRCs of 1,2,..k-bit corrupted z_i's must be unique, so that our table mapping is injective. Once we have a table T with such a unique mapping, we may proceed with the SCC protocol, with the difference being that our table can indicate that several bits are corrupted, and we must send k locations and k values to be verified. Note that we will still receive only a confirmation or a resend, even if only one hypothesized error bit is incorrect.

We tested all 16-bit CRC polynomials such that all one and two bit errors in 232 bits of data mapped to unique CRC values. Out of the 2^{16} possibilities, we found 64 polynomials that can correct for one and two bit errors, see Table 1. Fig. 5 shows that as the maximum message (data) size increases, at first a very large number of possible polynomials exist, but this decreases quite abruptly after 113 bits of message (data) length, and remains fairly constant until 240 bits, at which point it is not possible to perform two bit error correction with a 16-bit CRC. The exact reason for such abrupt decreases in polynomial availability is left for future work. We also noticed that, in general, the polynomials that work for a given message length work for all lengths less than that, which explains why the graph is monotonically decreasing.

We also modified our protocol such that the receiver may detect 1 or 2 bit errors and send the locations and values to the original sender. This causes only a minor increase in communication, and the runtime is not impacted. However, the size of the correction table is now 128 KB, as compared with 0.25KB for single-bit correction[1]. The advantage of multiple-bit error correction is further reduced retransmissions.

TABLE I. LIST OF 64 CRC16 GENERATOR POLYNOMIALS THAT CAN CORRECT 1- AND 2-BIT ERRORS IN 232 BITS OF DATA.

0000111000000111[a]
0001100100110001
0001110001101011
0010000100001001
0010010010101001
0010010111010011
0010011111111001
0010101001001001
0011001100111111
0011010101011001
0011011010010011
0011111101010011
0011111111001001
0100010111101111
0101011011110011
0101100100110101
0101101100101111
0110111101100011
0110111111101101
0111000111100111
0111001000110011
0111010101011101
0111100010110111
0111100111001011
0111101110111101
0111110011111011
1000000100000011
1000110111101101
1000111110100111
1001001011011001
1001001011110111
1001001110000111
1001010111111001
1001011101001001
1001100010011101
1001110101010111
1001111011010101
1010001110001011
1010011100111101
1010110001110001
1010110101101011
1011011111011011
1011111001111101
1100000011100001
1100000100000111
1100001110010011
1100100111001111

[1] Here is how these numbers were computed: For single-bit correction, we have 232 data bits and 8 CRC bits, for a total of 240 bits. So there are 240 potential bad indices, and we can represent the index number with 8 bits. A CRC-8 correction table has 2^8 rows of a byte each =.25KB.
For double-bit correction, we have 232 data bits and 16 CRC bits, for a total of 248 bits. So there are 248 potential bad indices, and we can represent the index number with 8 bits. A CRC-16 correction table has 2^16 rows of a byte each =64KB. But each entry in the table has two possible indices, so we have 128KB of space for the double-bit error correction table

1100101111100011
1100111100011101
1101010101010111
1101010101110011
1101101000111101
1101111010010011
1110001000111111
1110001110001111
1110011100100111
1110100010101111
1110100110110101
1110101000101111
1110110101101111
1110111101000101

a. Coefficients of $x^{15}, x^{14}, \ldots, x^{0}$ are listed left to right, coefficient of x^{16} is always 1

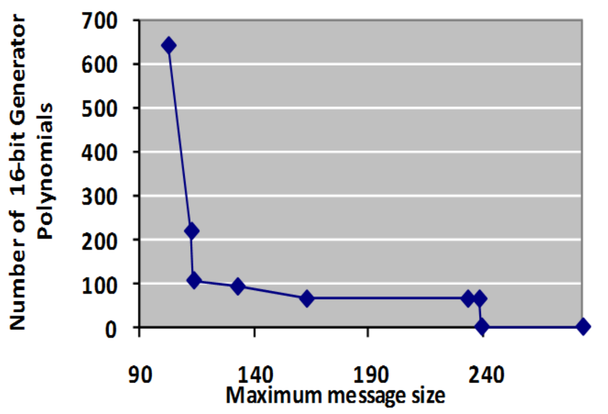

Figure 5. As the maximum message (data) size increases, the number of CRC polynomials for 1- and 2-bit error correction decreases.

VII. DISCUSSION

Related work about reducing the retransmission overhead by utilizing the corrupted packet includes [10, 11, 9]. In Partial Packet Recovery (PPR) [11], the receiver measures confidence values for the correctness of each bit in decoding, and requests for retransmission only those bits that are likely in error. In SOFT [10], the receiver stores the confidence values of the corrupted packet and combines it with the retransmitted packet, thereby improving packet reception probability. In the ZERO retransmission scheme [9], baseband level network coding is used to mix the retransmitted packet with the next packet. Our work is different since (1) our schema does not depend on a next packet and (2) our error-correcting CRC need less computational resources and energy.

ZigZag decoding [8] avoids retransmissons - that are caused by collisions - by decoding the non-overlapping pieces. Our work is different since (1) our errors do not have to be caused by collisions and (2) our error-correcting CRC needs less computational resources and energy.

Reed-Solomon Error Correction is a widely used correction scheme, however one of its faults is that it fails to efficiently correct or detect errors above a certain corruption level. [12] Thus many network architectures Hybrid ARQ (HARQ) protocol, which employs Reed-Solomon error correction in conjunction with a detecting CRC. [13] This allows the network to be able to correct small errors, but also detect high-error situations that Reed-Solomon cannot correct. However, our protocol uses only the CRC, thus avoiding the extra Reed-Solomon repetition needed by the HARQ detection-correction scheme.

VIII. CONCLUSION

Wireless communication faces transmission errors, but reducing retransmission can extend the lifetime of energy-constrained sensor networks. Our main contributions are summarized as follows:

- It is possible to *correct* errors using CRCs already present in sensor networks. Error correction can drastically reduce communication as resending information becomes necessary less frequently.

- We presented a simple and effective send-check-confirm (SCC) protocol that reduces retransmission, but reliability is not sacrificed since corrections are validated. The mechanism is very similar to the HTTP protocols's "304 Not Modified" message (indicating that resending the document is not necessary).

- Many 16-bit CRC polynomials, including $x^{16}+x^{12}+x^{5}+1$ which is used by IEEE 802.15.4 and TinyOS, can correct 1-bit errors in up to 240 bits of data. Correcting bit errors is worthwhile, considering that in a study of IEEE 802.15.4 "around 50% of errors are isolated to a single bit, and 60% of burst errors are only two bits long" [2].

- We found 64 candidate polynomials for 16-bit CRC that can correct for 1- and 2-bit errors in less than 240 bits of data. Since TinyOS' message format allows up to 232 bits of data, we recommend the use of one of these CRC polynomials.

- The number of generator polynomials that work for multiple-bit error correction is rather sparse as the message (data) size increases.

Future research can be pursued in the following directions: a thorough experimental evaluation of the SCC protocol, optimization of the tabular method such as that proposed in [4], modifying existing protocol stacks (e.g. TinyOS [7] or SunSPOT [6]) and investigating table space/ time tradeoffs.

REFERENCES

[1] B. McDaniel, "An Algorithm for Error Correcting Cyclic Redundance Checks". Dr.Dobb's Journal, 2003

[2] A. Vedral, J. Wollert, "Analysis of Error and Time Behavior of the IEEE 802.15.4 PHY-Layer in an Industrial Environment". IEEE International Workshop on Factory Communication Systems, 2006

[3] "RFC 2616 Hypertext Transfer Protocol", http://www.faqs.org/rfcs/rfc2616.html

[4] D. Yun, G. Ning, and D. Zaiwang, "CRC Look-up Table Optimization for Single-Bit Error Correction". Tsinghua Science & Technology, Volume 12, Number 5, October 2007

[5] S. Shukla, and N. Bergmann, "Single Bit Error Correction Implementation in CRC-16 on FPGA". In International Conference on Field Programmable Technology, 2004

[6] SunSPOT device, http://www.sunspotworld.com

[7] TinyOS, http://tinyos.net

[8] S. Gollakota and D. Katab, "ZigZag Decoding: Combating Hidden Terminals in Wireless Networks", In Proc. ACM SIGCOMM, 2008

[9] S. Yun and H. Kim, "Towards Zero-Cost Retransmission throughPhysical-Layer Network Coding in Wireless Networks", In Proc. ACM SIGCOMM, 2008

[10] G. R. Woo et al., "Beyond the Bits: Cooperative Packet Recovery Using Physical Layer Information", In Proc. ACM MOBICOM, 2007.

[11] K. Jamieson, and H. Balakrishnan, "PPR: Partial Packet Recovery for Wireless Networks". In Proc. ACM SIGCOMM, 2007.

[12] A. Dagnelies, "Algebraic soft-decoding of Reed-Solomon codes" Universite Catholique de Louvain Master's Thesis, 2007. P.57

[13] "Packet Front-End Link Protocol" Polygon Systems. http://www.polygon-control-systems.com/interfaces_pflp.htm

[14] G. R. Grimmett, and D. Stirzaker, Probability and Random Processes. New York: Oxford University Press, USA, 2001. P.33

[15] N. Bulusu, "CSE 410/510 Sensor Networks Winter 2009 Lecture 1": http://www.cs.pdx.edu/~nbulusu/courses/cs410-win09/lectures/Lecture1-win09.ppt. Slide 14.

[16] P. Koopman and T. Chakravarty, "Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks" The International Conference on Dependable Systems and Networks, DSN-2004, 2004.