Investigating CRC Polynomials that Correct Burst Errors

Travis Mandel[1], Jens Mache

***Abstract*- Error correction (instead of retransmission) can improve energy efficiency and lifetime of wireless sensor networks. In this paper, we concentrate on burst errors and error-correcting Cyclic Redundancy Checks (CRCs). We describe an optimized method for finding error-correcting CRC polynomials as maximum data length per frame increases. Our results show that the CCITT-16 polynomial (which is used by the IEEE 802.15.4 standard and by TinyOS) can correct 5-bit bursts in up to 156 bits of data as well as 4-bit bursts in as high as 1500 bits of data. For maximum error-correction capability, particular 16-bit CRC polynomials can correct 7-bit bursts in up to 184 bits of data, whereas other 16-bit CRC polynomials can correct 6-bit bursts in up to 803 bits of data.**

Index Terms- Wireless Sensor Networks, Cyclic Redundancy Check (CRC), Error Correction, Burst Error, Energy Efficiency

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 burst errors is even more worthwhile, considering that burst errors are commonly encountered in mobile communication systems. [18]

The rest of the paper is organized as follows: We will present background and related work in section II, Section III will discuss single-bit error correction, Section IV will examine burst error correction, Section V will explain optimizations when searching for valid generator polynomials, Section VI will present and analyze the results of our investigation of generator polynomials, followed by our 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 error detecting ability. [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.

McDaniel's paper on single-bit error correction [1] gives a preliminary analysis of using CRCs to correct single bits. We discuss some of this work in section III.

Our previous paper [19] forms the basis of the work we present in this paper. In that paper we carefully examine McDaniel's article, correcting some misleading statements that suggested error correcting CRCs were not plausible in practical applications. We developed the SCC protocol, which reduces data retransmission without sacrificing reliability. We also investigated the use of CRCs for the correction of multiple-bit errors. We explain some of our findings in III.

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 *valid 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 in [19] 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 ≈ 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 the message length in this way would also increase the probability that several bits of our message were corrupted. In [19] we present 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 L=2, n=3 and M' = 11101, we take c_2=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. Burst Error Correction

A. Bursts

A *burst error* of length b is "a contiguous sequence of b bits in which the first and last bit, and any number of intermediate bits, are received in error" [17]. These errors are more likely to occur in noisy transmissions than other patterns of errors. [18]. The traditional method of correcting burst errors focuses on interleaving [1], which requires several messages to be sent in a short period of time in order to receive meaningful data. Sending a large number of messages in a short timeframe is inconvenient and often not feasible in power-constrained sensor networks.

In [19] we examine error correction for 1,2,…m-bit errors. However, since burst errors are a small subset of these errors, we can correct more bits over longer maximum message sizes if we focus on correcting burst errors. This fact, coupled with the likelihood of burst errors, makes them an attractive object of study.

B. *Building the table*

In single-bit error correction, we created each z_i by changing the ith bit of Z to 1. However, we must have a more complicated construction of the z_is if we wish to have every 1,2,3…b-bit burst error represented by a single z_i. To do this, we created a b-bit burst register and moved it along Z.

To ensure uniqueness, we required that the last bit of the burst register always be one, and created all permutations of the other b-1 bits. Hence, no two z_{is} can be the same, as each has a different final bit or a different b-1 bit permutation. However this construction leaves out the case where there is a short burst at the beginning of the data. So this case must be handled separately in the code.

Here's an example of the z_is generated with a burst size of 3 and a maximum data size of 6:
Initial generation: (before we have room for a full register)
100000
010000
110000
Now, we generate successive registers and move them along the string:
001000
000100
000010
000001
011000
001100
000110
000011
101000
010100
001010
000101
111000
011100
001110
000111

Hence we cover all 1-bit, 2-bit-burst, and 3-bit-burst errors. Once we have constructed the proper z_is, we construct the table in the usual manner, except for the fact that each CRC maps to several bit locations. Then error correction over these bursts proceeds as described in III.

V. Optimizing the Search for Valid Polynomials

Note that the optimizations we describe in this section do not pertain to the process of error correction itself, but rather to the process of finding valid polynomials over a certain error *pattern* (e.g. 3-bit-bursts) and a range of message sizes.

A. *Using Past Information*

We wished to examine polynomial validity at various message sizes and burst lengths. However, searching through all 2^{16} possible generator polynomials proved very inefficient, resulting in only a few points on our graph being computed given a reasonable amount of time. To optimize the process, we relied on an observation we had made concerning the nature of the valid polynomials for successively larger message lengths. It seems that the polynomials that were valid for a longer message are also valid for a shorter one, in other words that the polynomials for length m+1 are a subset of the polynomials valid for length m. We will now present a proof of this fact.

1) *Proof*

All we need to show is that if a polynomial does not work for length m over some error pattern P[1], it cannot work for length m+1. Assume, for a contradiction, a generator polynomial p is invalid for length m, but valid for length m+1. Then, since p is not valid for length m, there must be two mutations of Z, z_1 and z_2, such that $CRC(z_1)=c=CRC(z_2)$. In other words $z_1 \bmod p = c \bmod p$ and $z_2 \bmod p = c \bmod p$, since the CRC is the remainder of the binary division. Hence, by modular arithmetic, $2z_1 \bmod p = 2c \bmod p$ and $2z_2 \bmod p = 2c \bmod p$, so $CRC(2z_1)= CRC(2z_2)$.

Equivalently, using polynomial notation, $z_1(x)= Q_1(x)*p(x) + c(x)$ and $z_2(x)= Q_2(x)*p(x) +c(x)$, since the CRC is the remainder of $z_i(x)/p(x)$. But then $2*z_1(x)= (2*Q_1(x))*p(x) + 2*c(x)$ and $2*z_2(x)= (2*Q_2(x))*p(x) +2*c(x)$. Hence the remainder of $2z_i(x)/p(x)$ must be the remainder of $2c(x)/p(x)$ for i=1,2, so $CRC(2z_1)= CRC(2z_2)$.

Now $2z_1$ and $2z_2$ are both of length m+1. And they still have pattern P, since it is present in the first m bits, so $2z_1$ and $2z_2$ must be encountered when computing all z_is with pattern P for length m+1. Since $CRC(2z_1)= CRC(2z_2)$, there is a collision in m+1, and p is invalid, which contradicts our assumption. □

2) *Implementation*

Given this fact, we can take a minimum length of interest, say 100, and compute the valid polynomials using the standard method, and then only try those polynomials for the next larger length. In this way, we compute the same result using much fewer choices.

Psuedocode:
Open file containing valid polynomials for length m
Read from file into a list
------Loop until the list is empty----------
    Increment m
    Check each element of list for validity over message length m
    Output size to a file
    Remove invalid elements

If one wishes to examine a range of message sizes at which there are a large number of possibilities, this method may still take too long. However, this algorithm parallelizes very well. One first has each processor compute several points spaced across the range one wishes to investigate. Then, one subtracts the polynomials found at measurement r+1 from those found at measurement r. We do this

[1] Note that the pattern must be resilient to adding an extra zero to the end of the string. For example, this proof does not apply to the pattern of all ones.

because if a polynomial is still valid for length r+1, there is no reason to continually test it when computing the r-r+1 interval. This also ensures that the program will terminate before r+1, avoiding unnecessary computations. Each processor then computes a segment of the curve with the remaining polynomials.

VI. Results and Analysis

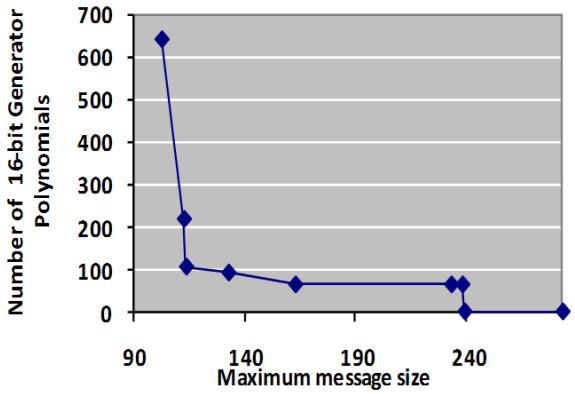

Figure 1. Original sparsely sampled graph shown in [19] of valid generator polynomials for double bit errors.

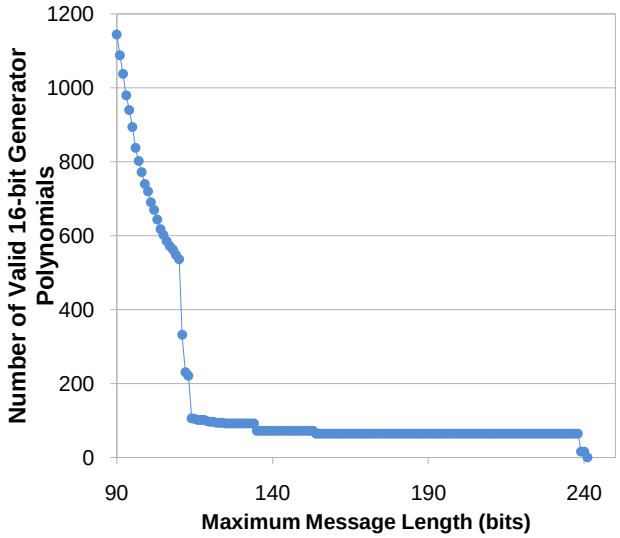

Figure 2. Valid generator polynomials for double bit errors, now with every integer point sampled, made possible by optimizations show in V. We do not show results past 240, as no generator polynomials exist past that point. Note that the smoother graph allows us to examine more closely the parabolic nature of the points before 110, and the sharp drop from 110 to 114 is now evident.

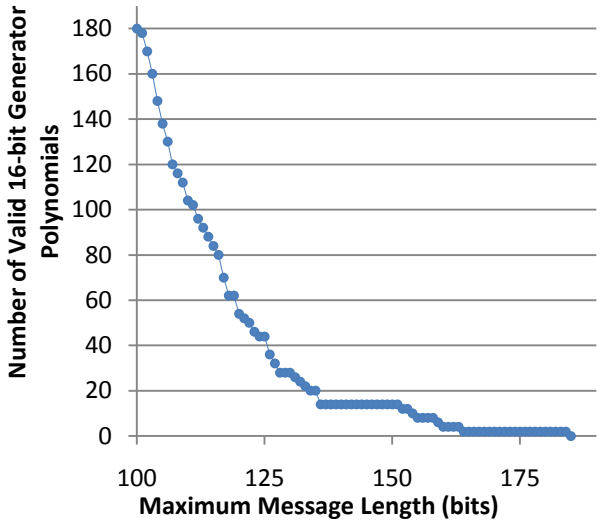

Figure 3. Valid Generator Polynomials for 7 Bit Burst Errors. There are no more valid polynomials after the maximum message length exceeds 184 bits.

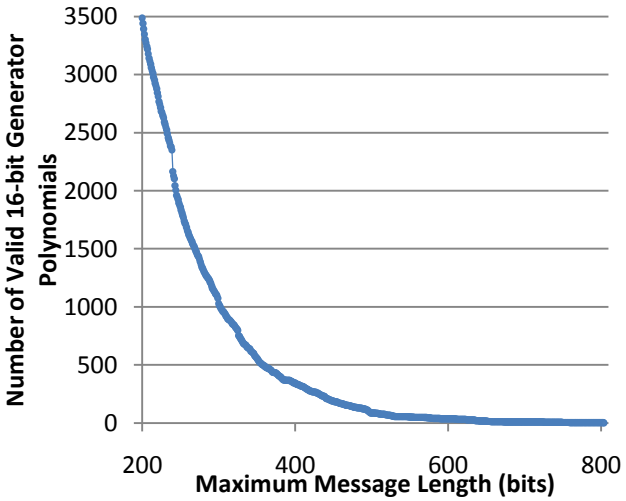

Figure 4. Valid Generator Polynomials for 6 Bit Burst Errors. For Tiny OS' maximum message size (232 bits), there are 2524 valid generator polynomials. There are no more valid polynomials after the maximum message length exceeds 803 bits.

There are several very interesting things to note about these three graphs. One is that the burst graphs are fairly clearly exponential decay, while the double bit graph is much sharper-cornered. To see this, after performing an exponential regression on the three sets of data, Figure 4 had a square error coefficient(R^2) of .96, Figure 3 has an R^2 of .73, but Figure 2 has a an R^2 of .31. Another feature of note is that the 6-bit-graph seems fairly smooth until the large gap between 239 bits and 240 bits. This is interesting because 240-241 is the point at which there become no more valid double-bit polynomials, so perhaps there is a connection there.

A simple observation is that there are many more polynomials for 6-bit-bursts than there are for seven bit-bursts at the same message size. And lower burst lengths have many more generator polynomials available, as shown below:

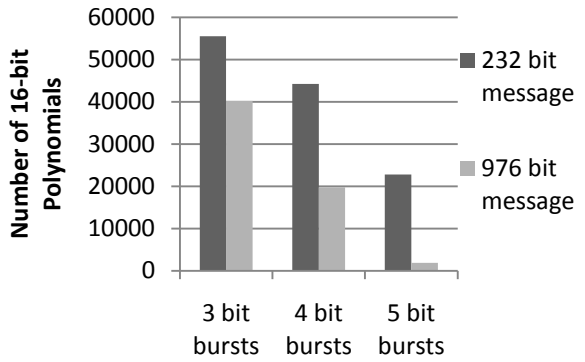

Figure 5. Valid Generator Polynomials for 3,4, and 5 bit burst errors. We compare the number of polynomials between TinyOS's 232 bit and 802.15.4's 976 bit maximum message size. Note that as the burst size decreases, the number of polynomials approaches 2^{16}-1 (65535).

We also tested the standard CRC-16-CCITT polynomial at different burst lengths. There are no significant results until 5-bit bursts, at which point the CCITT polynomial works for messages up to 156 bits. And for 4-bit-bursts, the standard polynomial is still valid even up to messages of size 1500, which means that standards such as 802.15.4, which have a maximum message size of 127 bytes (976 bits), could be able to correct single bit, 2, 3, and 4-bit-burst errors while using the currently used generator polynomial.

VII. CONCLUSIONS

Error correction can improve energy efficiency and lifetime of wireless networks. In this paper, we concentrated on burst errors and error-correcting CRCs. Our main contributions are summarized as follows:

- Error Correction with CRCs is not limited to single-bit correction, even at longer message sizes.
- The standard CRC-16-CCITT generator polynomial is valid for 4-bit-burst correction even up to 802.15.4's maximum message length of 127 bytes.
- Generator polynomials that are not valid for a given maximum message size cannot be valid for a larger size.
- Burst errors of 1, 2 ,3, 4 ,5, and 6 bits, which are common in most networks, can be corrected with the maximum Tiny OS message size.
- For maximum error-correction capability, particular 16-bit CRC polynomials can correct 7-bit bursts in up to 184 bits of data, whereas other 16-bit CRC polynomials can correct 6-bit bursts in up to 803 bits of data.

Future research includes 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.

[17] W. Stallings, "Wireless Communications and Networks: Second Edition" Prentice Hall, 2005. p.198

[18] W. Stallings, "Data and Computer Communications" Prentice Hall, 2007. P.186

[19] T. Mandel and J. Mache. "Selected CRC Polynomials Can Correct Errors and Thus Reduce Retransmission". WITS (DCOSS) 2009.