CRC Performance Calculations Brute Forcing the Dual Code –the Forgotten Backdoor

Technical Report · October 2014

DOI: 10.13140/2.1.4475.1042

2 authors:

Jan Endresen
Sentri AS
25 PUBLICATIONS **397** CITATIONS
SEE PROFILE

Anne Elisabeth Vallestad
AEVLOGIC AS
5 PUBLICATIONS **127** CITATIONS
SEE PROFILE

CRC Performance Calculations

Brute Forcing the Dual Code - the Forgotten Backdoor

Jan Endresen
ABB Corporate Research
Bergerveien 12
1375 Billingstad, Norway
jan.endresen@no.abb.com

Anne Elisabeth Vallestad
ABB Corporate Research
Bergerveien 12
1375 Billingstad, Norway
anne.e.vallestad@no.abb.com

***Abstract*—The use of a Cyclic Redundancy Check (CRC) is a common way of protecting information in communication and storage systems. To calculate the peformance of the code is not trivial and in particular, when the code is shortened. We show that there is a direct way to calculate the probability of undetected errors for any shortened polynomial code. A software implementation can handle CRCs of length 32 and a Field Programmable Gate Array (FPGA) based solution can handle CRCs of length 40. It is no longer necessary to rely on approximate calculations.**

Keywords—weight structure; polynomial codes; dual code; shortened codes

I. INTRODUCTION

Communication over a noisy channel is normally protected by a CRC (Cyclic Redundancy Check). The key question is: What is the likelihood that a received code-word is corrupted, but the CRC is unable to detect it? This is called the probability of undetected errors (Pude) or the residual error rate. It can be calculated directly if the weight structure (A_i) of the code is known. However, it is only known in closed form for some codes, e.g. Hamming codes and a subset of the BCH (Bose-Chaudhuri-Hocquenghem) codes. Often the application is not using the full length of the code words. By shortening the code-word the probability of undetected errors is normally improved partly by increasing the minimum Hamming Distance. However, in this case even less is known in closed form. This paper proposes an alternative approach for calculating the weight structure exploiting the use of the dual code.

II. GENERAL CODING THEORY

Most CRCs in use today are linear codes, i.e. for a binary code the modulo-2 sum of two code-words is also a code-word. In the analysis of the Pude it is normal to make two assumptions: the channel is binary symmetric and the bit errors are independent. Binary symmetric channel (BSC) means that the probability of corrupting a 1 into a 0 is the same as corrupting a 0 into a 1. The two assumptions may or may not be true for a particular communication system. Some communication systems have the property that errors tend to come in bursts which would be a violation of the independence assumption and some modulation schemes violate the BSC property.

If the two assumptions are fulfilled the Pude can be calculated exactly by the following formula [1]:

$$P_{ude} = \sum_{i=d_{min}}^{n} A_i \, p^i \, (1-p)^{n-i} \qquad (1)$$

A_i is the weight structure for the code, p is the bit error rate (the probability that a bit is corrupted), d_{min} is the minimum Hamming Distance and n is the total length of the code-word including the CRC. In principle the weight structure can be obtained by generating all possible code words and count how many code words have one bit equal to 1, how many code words have two bits equal to 1 and so on all the way to n. This approach is not practical when k (the length of the information part) becomes large, typically k > 32. The total number of different code words in a binary code is 2^k. A_1 to A_{dmin-1} are equal to zero. n= k+r where r is the number of CRC bits and the code is denoted as an (n,k) code.

Often the application of the code is not using the full length of the code-words. This is called shortening the code. By shortening the code, the P_{ude} is normally improved. If the curve of P_{ude} is monotonic, i.e. it increases monotonically from small bit error rates to p=0.5 then it is called proper. For some shortened codes this is no longer true. Such codes are called improper and the significance of that is that the upper bound for the $P_{ude} = 2^{-r}$ is no longer valid.

Other researchers have also investigated the performance of shortened codes like Fujiwara [3] but he limited the investigation to Hamming codes. Schiller [5] has developed a very generic approach, which is especially promising for systems with nested (concatenated) CRCs.

III. AN APPROXIMATION APPROACH

A common approximation equation for (1) is:

$$P_{ude} = 2^{-r} \sum_{i=d_{min}}^{n} \binom{n}{i} \, p^i \, (1-p)^{n-i} \qquad (2)$$

where $\binom{n}{i}$ is the binomial coefficient. This equation could be based on:

$$P_{ude} \leq 2^{-r} \sum_{i=1}^{n} \binom{n}{i} p^i (1-p)^{n-i} \quad (3)$$

which is an upper bound for the average probability of undetected error for all (n,k) linear systematic codes [1]. It is important to understand that some codes will be better than this bound and others will be worse. It is therefore a real possibility that a particular code is significantly worse than indicated by this bound (3). Furthermore the terms from 1 to d_{min-1} have been removed in the common approximation (2). Since all those terms are non-negative the new "bound" is even more likely to give an optimistic unsafe result.

There are additional problems with the above approximation (2). How is the d_{min} determined? If the code is defined by a generator polynomial then an analysis of the polynomial can reveal some of the properties of the code. It will reveal the minimum d_{min} of the code, but not necessarily the true value. It could be higher.

Equation (2) must never be used to ascertain if the code is proper. The properness of the code is critical to know since the maximum P_{ude} could be significantly affected. To evaluate the code with very short information part is also an indication if the code is proper or not and it will reveal the maximum d_{min}. Wolf [2] has shown that a code can be improper for certain information lengths only. This makes it important to evaluate P_{ude} for all the allowed information lengths.

IV. The Dual Code

Assume that the information part of the code is 124 bytes long and the CRC is 4 bytes. Brute forcing this code (generating all possible code-words) would require $2^{8*124} \approx 4*10^{298}$ code words to be generated which is an inconceivable task. However, there is another alternative. There exists a related code where the length of the information part is equal to the length of the CRC part of the original code. Furthermore, the length of the CRC is equal to the length of the information part of the original code. The dual code has the same total length as the original code n= r+k and is denoted as a (n,r) code.

There is a remarkable result from Florence Jessie McWilliams [4] that enables us to calculate the P_{ude} for the original code from the weight structure for the dual code (B_i) [1].

$$P_{ude} = 2^{-r} \sum_{i=0}^{n} B_i (1-2p)^i - (1-p)^n \quad (4)$$

B_i is the weight structure for the dual code, p is again the bit error rate and n is the total length of the dual code word including the CRC.

In the above example it is possible to brute force the 2^{32} code words (about 4 billion) of the dual code. Obtaining the weight structure is then just a matter of counting. The method is the same for the original and the dual code, the difference is the number of code words.

McWilliams [4] published her "Identity" relating the weight structure of the original and the dual code. However this is normally not a feasible approach due to the enormously big numbers of code words for the original code.

V. Generating the Dual Code

From the generator polynomial, the so-called generator matrix can be obtained. If the generator matrix is manipulated into the form:

$$G = [\, P \mid I_k \,] \quad (5)$$

where the generator matrix has two parts P and the Identity matrix of size k, then the generator matrix for the dual code is:

$$G_{dual} = [\, I_r \mid P^T \,] \quad (6)$$

where P^T is the transposed of P and the Identity matrix is of size r [1].

G can easily be produced for any polynomial code. Each row in the matrix P is the corresponding CRC for an information part consisting of a single 1 bit placed in the appropriate place.

VI. Practical Implementations

As a test case we are using a BCH code with triple error correction and r = 24. The maximum code-word length of this code is 255 bits and the weight structure is known for the dual code at the maximum length which was used as check [1]. We have developed both an FPGA design and a software program to calculate the weight structure for the dual code. The Mathematica code is shown next. This is the complete code to generate the complete weight structures for the dual code for all the different information lengths from k=1 to k = 231 bits. The Mathematica code is not very efficient when it comes to execution time and takes about one day to produce the results for this 24 bits CRC and k = 231. We have also converted the Mathematica code to the programming language C and use look-up tables extensively rather than the function calls. In this way we can produce the weight structures for a 32 bits CRCs in a few hours on a standard PC. The FPGA (Xilinx Virtex-7-family, device VX485T) can do the job for the CRC32 in 40 seconds. The FPGA design can be modified to handle a 40 bits CRCs.

The Mathematica code for the BCH(255,231,7) is:

```
(*The BCH Polynomial*)
g = x^24 + x^23 + x^21 + x^20 + x^19 + x^17 + x^16 + x^15 + x^13 + x^8 + x^7 +
x^5 + x^4 + x^2 + 1;

(* 1) The Generator Matrix in vector form represented as integers*)
For[j = 1, j <= k, j++,
  gv[j] = FromDigits[Reverse[ CoefficientList[
PolynomialRemainder[ x^(r - 1 + j), g, x, Modulus -> 2], x]], 2]];

(* 2) Initialize the dual weights b to 0*)
For[i = 0, i <= n, i++,  For[j = 1, j <= k, j++, b[i, j] = 0]];

(* 3) Brute force all possible dual code code-words*)
For[i = 0, i <= 2^r – 1, i++,
  w = DigitCount[ i, 2, 1];
  For[j = 1, j <= k, j++,
    If[ OddQ[ DigitCount[ BitAnd[ i, gv[j]], 2, 1]], w++];
    b[w, j]++]];
```

A. A short explanation of the code.

1) The generator matrix is obtained by producing k independent rows. Each row consists of a single "1" bit in the information part, all at different places. This is the term x^{r-1+j}. The function PolynomialRemainder produces in a polynomial form, the CRC for the given information part. CoefficientList takes this polynomial and turn it into a list of 1 and 0. The Reverse call is needed since Mathematica lists polynomial with the lowest exponent to the left. This list is converted into an integer by FromDigits

2) The weights are stored in a matrix b[i,j]. J is the index representing the length of the information part and the length of the CRC for the dual code. The algorithm is iterative in the sense that b[i,j] = b[i,j] + 1 for some conditions. Mathematica requires b[i,j] to be given an initial value.

3) The brute forcing requires 2^r code-words to be generated. The information part is represented by i. First the weight of the information part is found by DigitCount and then weight for the CRC part for each different lengths. In the final loop j represents the different length CRCs. BitAnd produces the bitwise AND of the information part and the j^{th} column of the generator matrix for the dual code. DigitCount counts the number of "1" in the binary result from the BitAnd call. If the number of "1" is odd (OddQ) then the weight is incremented by 1. This weight together with column index j determine which element in the weight matrix to increment.

If one would like to speed up this code the DigitCount function is the prime target. It takes about 90% of the execution time of the inner loop. It can quite easily be replaced by table look-up which needs to be generated prior to the brute forcing.

After the weight structure has been obtained, equation (4) is used to calculate the probability of undetected errors. A note of caution when using (4) is that it consists of two terms both close in value to one (at least for small p and large n). The difference between the two terms produces a very small value typically 10^{-10} to 10^{-20} is common. If floating point arithmetic is used non sensible results can be obtained. The simple solution is to give p=1/1000 rather than p=0.001 and let Mathematica do the calculations in integer arithmetic. Mathematica automatically adjusts the precision to whatever is required at the cost of execution time.

B. Graphical results

Figure 1 shows the probability of undetected errors for the BCH test case for different information lengths. It should be noted that for k=16 the code is significantly improper. The minimum Hamming Distance changes from 7 to 8 between k=16 and k=8.

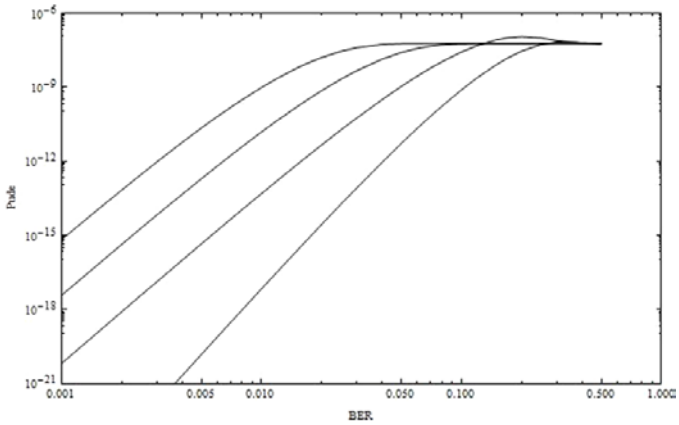

Fig. 1. The P_{ude} for the BCH code, for k=8, 16, 96 and 231

Figure 2 shows the same as figure 1 with the added red dashed curves, using the approximation given by (2). For the long code-words they are almost identical to the correct curves. However at the shorter lengths k = 8 and 16 they are significantly different. If evaluated at 1% bit error they differ by a factor above 5 and 6 respectively.

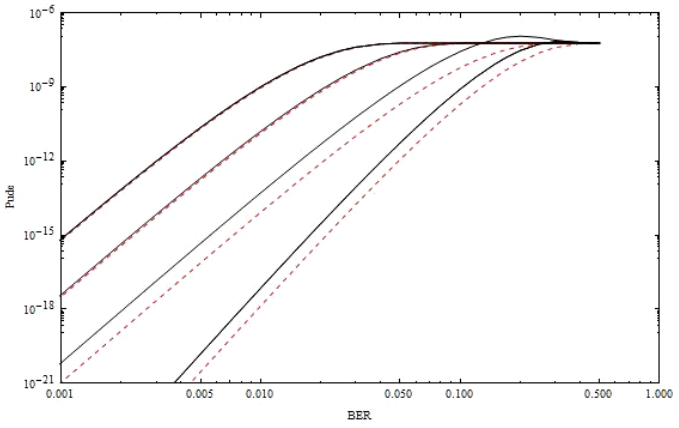

Fig. 2. The P_{ude} for the BCH code, for k=8, 16, 96 and 231 with the approximations from (2) added in red dashed.

VII. CONCLUSIONS

We have given a practical solution to the old problem of calculating the probability of undetected error exactly. Our approach is generic in the sence that any polynomial code and any degree of shortening can be handled.

Using brute force on the dual code together with (4) is an alternative for most practical codes. We do not need to use the approximation given by (2), which must not be used for deciding if the code is proper or not. Equation (2) also has the added problem of the often unknown minimum Hamming Distance. Our approach does not require this parameter but will reveal the value d_{min} as the slope of the curve of Pude at low bit error rates.

A software version of the algorithm can be used for CRCs up to about 32 bits, depending upon the number of information bits required. The use of an FPGA (Xilinx Virtex-7-family, device VX485T) can brute force the dual code up to 40 bit CRCs at least.

REFERENCES

[1] Lin and Costello (2004), Error Control Coding, Prentice Hall; 2 edition

[2] Wolf, J.K. and Blankeney, R.D. An exact evaluation of the propability of undetected error for certain shortened binary CRC codes,, IEEE Transactions on Communications (Volume:33, Issue: 6) Jun 1985, pp 570 - 574

[3] T. Fujiwara, T. Kasami, A. Kitai, and S. Lin, "On the Undetected Error Probability for Shortened Hamming Codes", IEEE Trans on Communications vo!. COM-33, pp. 570-574, June, 1985.

[4] MacWilliams, F.J. and N.J.A. Sloane (1981). The Theory of Error-Correcting Codes, North-Holland

[5] Schiller, F., Mattes, T.: An Efficient Method to Evaluate CRC-Polynomials for Safety- Critical Industrial Communication. Journal of Applied Computer Science 14, 57–80 (2006)