Network Working Group
Request for Comments: 3385
Category: Informational
D. Sheinwald
J. Satran
IBM
P. Thaler
V. Cavanna
Agilent
September 2002

Internet Protocol Small Computer System Interface (iSCSI) Cyclic Redundancy Check (CRC)/Checksum Considerations

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

In this memo, we attempt to give some estimates for the probability of undetected errors to facilitate the selection of an error detection code for the Internet Protocol Small Computer System Interface (iSCSI).

We will also attempt to compare Cyclic Redundancy Checks (CRCs) with other checksum forms (e.g., Fletcher, Adler, weighted checksums), as permitted by available data.

1. Introduction

Cyclic Redundancy Check (CRC) codes [Peterson] are shortened cyclic codes used for error detection. A number of CRC codes have been adopted in standards: ATM, IEC, IEEE, CCITT, IBM-SDLC, and more [Baicheva]. The most important expectation from this kind of code is a very low probability for undetected errors. The probability of undetected errors in such codes has been, and still is, subject to extensive studies that have included both analytical models and simulations. Those codes have been used extensively in communications and magnetic recording as they demonstrate good "burst error" detection capabilities, but are also good at detecting several independent bit errors. Hardware implementations are very simple and well known; their simplicity has made them popular with hardware developers for many years. However, algorithms and software for effective implementations of CRC are now also widely available [Williams].

The probability of undetected errors depends on the polynomial selected to generate the code, the error distribution (error model), and the data length.

2. Error Models and Goals

We will analyze the code behavior under two conditions:

- noisy channel - burst errors with an average length of n bits
- low noise channel - independent single bit errors

Burst errors are the prevalent natural phenomenon on communication lines and recording media. The numbers quoted for them revolve around the BER (bit error rate). However, those numbers are frequently nothing more than a reflection of the Burst Error Rate multiplied by the average burst length. In field engineering tests, three numbers are usually quoted together -- BER, error-free-seconds and severely-error-seconds; this illustrates our point.

Even beyond communication and recording media, the effects of errors will be bursty. An example of this is a memory error that will affect more than a single bit and the total effect will not be very different from the communication error, or software errors that occur while manipulating packets will have a burst effect. Software errors also result in burst errors. In addition, serial internal interconnects will make this type of error the most common within machines as well.

We also analyze the effects of single independent bit errors, since these may be caused by certain defects.

On burst, we assume an average burst error duration of bd, which at a given transmission rate s, will result in an average burst of a = bd*s bits. (E.g., an average burst duration of 3 ns at 1Gbs gives an average burst of 3 bits.)

For the burst error rate, we will take 10^-10. The numbers quoted for BER on wired communication channels are between 10^-10 to 10^-12 and we consider the BER as burst-error-rate*average-burst-length. Nevertheless, please keep in mind that if the channel includes wireless links, the error rates may be substantially higher.

For independent single bit errors, we assume a 10^-11 error rate.

Because the error detection mechanisms will have to transport large amounts of data (petabytes=10^16 bits) without errors, we will target very low probabilities for undetected errors for all block lengths (at 10Gb/s that much data can be sent in less than 2 weeks on a single link).

Alternatively, as iSCSI has to perform efficiently, we will require that the error detection capability of a selected protection mechanism be very good, at least up to block lengths of 8k bytes (64kbits).

The error detection capability should keep the probability of undetected errors at values that would be "next-to-impossible". We recognize, however, that such attributes are hard to quantify and we resorted to physics. The value 10^23 is the Avogadro number while 10^45 is the number of atoms in the known Universe (or it was many years ago when we read about it) and those are the bounds of incertitude we could live with. (10^-23 at worst and 10^-45 if we can afford it.) For 8k blocks, the per/bit equivalent would be (10^-28 to 10^-50).

3. Background and Literature Survey

Each codeword of a binary (n,k) CRC code C consists of n = k+r bits. The block of r parity bits is computed from the block of k information bits. The code has a degree r generator polynomial g(x).

The code is linear in the sense that the bitwise addition of any two codewords yields a codeword.

For the minimal m such that g(x) divides (x^m)-1, either n=m, and the code C comprises the set D of all the multiplications of g(x) modulo (x^m)-1, or n<m, and C is obtained from D by shortening each word in the latter in m-n specific positions. (This also reduces the number of words since all zero words are then discarded and duplicates are not maintained.)

Error detection at the receiving end is made by computing the parity bits from the received information block, and comparing them with the received parity bits.

An undetected error occurs when the received word c’ is a codeword, but is different from the c that is transmitted.

This is only possible when the error pattern e=c’-c is a codeword by itself (because of the linearity of the code). The performance of a CRC code is measured by the probability Pud of undetected channel errors.

Let Ai denote the number of codewords of weight i, (i.e., with i 1-bits). For a binary symmetric channel (BSC), with sporadic, independent bit error ratio of probability 0<=epsilon<=0.5, the probability of undetected errors for the code C is thus given by:

Pud(C,epsilon) = Sigma[for i=d to n] (Ai*(epsilon^i)*(1-epsilon)^(n-i))

where d is the distance of the code: the minimal weight difference between two codewords in C which, by the linearity of the code, is also the minimal weight of any codeword in the code. Pud can also be expressed by the weight distribution of the dual code: the set of words each of which is orthogonal (bitwise AND yields an even number of 1-bits) to every word of C. The fact that Pud can be computed using the dual code is extremely important; while the number of codewords in the code is 2^k, the number of codewords in the dual code is 2^r. k is in the orders of thousands, and r in the order of 16 or 24 or 32. If we use Bi to denote the number of codewords in the dual code which are of weight i, then ([LinCostello]):

Pud (C,epsilon) = 2^-r Sigma [for i=0 to n] Bi*(1-2*epsilon)^i -
(1-epsilon)^n

Wolf [Wolf94o] introduced an efficient algorithm for enumerating all the codewords of a code and finding their weight distribution.

Wolf [Wolf82] found that, counter to what was assumed, (1) there exist codes for which Pud(C,epsilon)>Pud(C,0.5) for some epsilon not=0.5 and (2) Pud is not always increasing for 0<=epsilon<=0.5. The value of what was assumed to be the worst Pud is Pud(C,0.5)=(2^-r) - (2^-n). This stems from the fact that with epsilon=0.5, all 2^n received words are equally likely and out of them 2^(n-r)-1 will be accepted as codewords of no errors, although they are different from the codeword transmitted. Previously Pud had been assumed to equal [2^(n-r)-1]/(2^n-1) or the ratio of the number of non-zero multiples of the polynomial of degree less than n (each such multiple is undetected) and the number of possible error polynomials. With either formula Pud approaches 1/2^r as n approaches infinity, but Wolf’s formula is more accurate.

Wolf [Wolf94j] investigated the CCITT code of r=16 parity bits. This code is a member of the family of (shortened codes of) BCH codes of length 2^(r-1) -1 (r=16 in the CCITT 16-bit case) generated by a polynomial of the form g(x) =(x+1)p(x) with p(x) being a primitive polynomial of degree r-1 (=15 in this case). These codes have a BCH design distance of 4. That is, the minimal distance between any two codewords in the code is at least 4 bits (which is earned by the fact that the sequence of powers of alpha, the root of p(x), which are roots of g(x), includes three consecutive powers -- alpha^0, alpha^1, alpha^2). Hence, every 3 single bit errors are detectable.

Wolf found that different shortened versions of a given code, of the same codeword length, perform the same (independent of which specific indexes are omitted from the original code). He also found that for the unshortened codes, all primitive polynomials yield codes of the same performance. But for the shortened versions, the choice of the primitive polynomial does make a difference. Wolf [Wolf94j] found a primitive polynomial which (when multiplied by x+1) yields a generating polynomial that outperforms the CCITT one by an order of magnitude. For 32-bit redundancy bits, he found an example of two polynomials that differ in their probability of undetected burst of length 33 by 4 orders of magnitude.

It so happens, that for some shortened codes, the minimum distance, or the distribution of the weights, is better than for others derived from different unshortened codes.

Baicheva, et. al. [Baicheva] made a comprehensive comparison of different generating polynomials of degree 16 of the form g(x) = (x+1)p(x), and of other forms. They computed their Pud for code lengths up to 1024 bits. They measured their "goodness" -- if Pud(C,epsilon) <= Pud(C,0.5) and being "well-behaved" -- if Pud(C,epsilon) increases with epsilon in the range (0,0.5). The paper gives a comprehensive table that lists which of the polynomials is good and which is well-behaved for different length ranges.

For a single burst error, Wolf [Wolf94J] suggested the model of (b:p) burst -- the errors only occur within a span of b bits, and within that span, the errors occur randomly, with a bit error probability 0 <= p <= 1.

For p=0.5, which used to be considered the worst case, it is well known [Wolf94J] that the probability of undetected one burst error of length b <= r is 0, of length b=r+1 is 2^-(r-1), and of b > r+1, is 2^-r, independently of the choice of the primitive polynomial.

With Wolf's definition, where p can be different from 0.5, indeed it was found that for a given b there are values of p, different from 0.5 which maximize the probability of undetected (b:p) burst error.

Wolf proved that for a given code, for all b in the range r < b < n, the conditional probability of undetected error for the (n, n-r) code, given that a (b:p) burst occurred, is equal to the probability of undetected errors for the same code (the same generating polynomial), shortened to block length b, when this shortened code is used with a binary symmetric channel with channel (sporadic, independent) bit error probability p.

For the IEEE-802.3 used CRC32, Fujiwara et al. [Fujiwara89] measured the weights of all words of all shortened versions of the IEEE 802.3 code of 32 check bits. This code is generated by a primitive polynomial of degree 32:

g(x) = x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 and hence the designed distance of it is only 3. This distance holds for codes as long as 2^32-1. However, the frame format of the MAC (Media Access Control) of the data link layer in IEEE 802.3, as well as that of the data link layer for the Ethernet (1980) forbid lengths exceeding 12,144 bits. Thus, only such bounded lengths are investigated in [Fujiwara89]. For shortened versions, the minimum distance was found to be 4 for lengths 4096 to 12,144; 5 for lengths 512 to 2048; and even 15 for lengths 33 through 42. A chart of results of calculations of Pud is presented in [Fujiwara89] from which we can see that for codes of length 12,144 and BSC of epsilon = 10^-5 - 10^-4, Pud(12,144,epsilon)= 10^-14 - 10^-13 and for epsilon = 10^-4 - 10^-3, Pud(512,epsilon) = 10^-15, Pud(1024,epsilon) = 10^-14, Pud(2048,epsilon) = 10^-13, Pud(4096,epsilon) = 10^-12 - 10^-11, and Pud(8192,epsilon) = 10^-10 which is rather close to 2^-32.

Castagnoli, et. al. [Castagnoli93] extended Fujiwara’s technique for efficiently calculating the minimum distance through the weight distribution of the dual code and explored a large number of CRC codes with 24 and 32 redundancy bit. They explored several codes built as a multiplication of several lower degree irreducible polynomials.

In the popular class of (x+1)*deg31-irreducible-polynomial they explored 47000 polynomials (not all the possible ones). The best they found has d=6 up to block lengths of 5275 and d=4 up to 2^31-1 (bits).

The investigation was done in 1993 with a special purpose processor.

By comparison, the IEEE-802 code has d=4 up to at least 64,000 bits (Fujikura stopped looking at 12,144) and d=3 up to 2^32-1 bits.

CRC32/4 (we will refer to it as CRC32C for the remainder of this memo) is 11EDC6F41; IEEE-802 CRC is 104C11DB7, denoting the coefficients as a bit vector.

[Stone98] evaluated the performance of CRC (the AAL5 CRC that is the same as IEEE802) and the TCP and Fletcher checksums on large amounts of data. The results of this experiment indicate a serious weakness of the checksums on real-data that stems from the fact that checksums do not spread the "hot spots" in input data. However, the results show that Fletcher behaves by a factor of 2 better than the regular TCP checksum.

4. Probability of Undetected Errors - Burst Error

4.1 CRC32C (Derivations from [Wolf94j])

Wolf [Wolf94j] found a 32-bit polynomial of the form g(x) = (1+x)p(x) for which the conditional probability of undetected error, given that a burst of length 33 occurred, is at most (i.e., maximized over all possible channel bit error probabilities within the burst) 4 * 10^-10.

We will now figure the probability of undetected error, given that a burst of length 34 occurred, using the result derived in this paper, namely that for a given code, for all b in the range 32 < b < n, the conditional probability of undetected error for the (n, n-32) code, given that a (b:p) burst occurred, is equal to the probability of undetected errors for the same code (the same generating polynomial), shortened to block length b, when this shortened code is used with a binary symmetric channel with channel (sporadic, independent) bit error probability p.

The approximation formula for Pud of sporadic errors, if the weights Ai are distributed binomially, is:

Pud(C, epsilon) =~= Sigma[for i=d to n] ((n choose i) / 2^r)*(1-epsilon)^(n-i) * epsilon^i .

Assuming a very small epsilon, this expression is dominated by i=d. From [Fujiwara89] we know that for 32-bit CRC, for such small n, d=15. Thus, when n grows from 33 to 34, we find that the approximation of Pud grows by (34 choose 15) / (33 choose 15) = 34/19; when n grows further to 35, Pud grows by another 35/20.

Taking, from Wolf [Wolf94j], the most generous conditional probability, computed with the bit error probability p* that maximizes Pub(p|b), we derive: Pud(p*|33) = 4 x 10^{-10}, yielding Pud(p*|34) = 7.15 x 10^{-10} and Pud(p*|35) = 1.25 x 10^{-9}.

For the density function of the burst length, we assume the Rayleigh density function (the discretization thereof to integers), which is the density of the absolute values of complex numbers of Gauss distribution:

f(x) = x / a^2 exp {-x^2 / 2a^2 }   , x>0 .

This density function has a peak at the parameter a and it decreases smoothly as x increases.

We take three consecutive bits as the most common burst event once an error does occur, and thus a=3.

Now, the probability that a burst of length b occurs in a specific position is the burst error rate, which we estimate as 10^{-10}, times f(b). Calculating for b=33 we find f(33) = 1.94 x 10^{-26}. Together, we found that the probability that a burst of length 33 occurred, starting at a specific position, is 1.94 x 10^{-36}.

Multiplying this by the generous upper bound on the probability that this burst error is not detected, Pud(p*|33), we get that the probability that a burst occurred at a specific position, and is not detected, is 7.79 x 10 ^{-46}.

Going again along this path of calculations, this time for b=34 we find that f(34) = 4.85*10^{-28}. Multiplying by 10^{-10} and by Pud(p*|34) = 7.15*10^{-10} we find that the probability that a burst of length 34 occurred at a specific position, and is not detected, is 3.46*10^{-47}.

Last, computing for b=35, we get 1*10^{-29} * 10^{-10} * 1.25*10^{-9} = 1.25*10^{-48}.

It looks like the total can be approximated at 10^-45 which is within the bounds of what we are looking for.

When we multiply this by the length of the code (because thus far we calculated for a specific position) we have 10^-45 * 6.5*10^4 = 6.5*10^-41 as an upper bound on the probability of undetected burst error for a code of length 8K Bytes.

We can also apply this overestimation for IEEE 802.3.

Comment: 2^{-32} = 2.33*10^{-10}.

5. Probability of Undetected Errors - Independent Errors

5.1 CRC (Derivations from [Castagnoli93])

It is reported in [Castagnoli93] that for BER = epsilon=10^-6, Pud for a single bit error, for a code of length 8KB, for both cases, IEEE-802.3 and CRC32C is 10^{-20}. They also report that CRC32C has distance 4, and IEEE either 3 or 4 for this code length. From this, and the minimum distance of the code of this length, we conclude that with our estimation of epsilon, namely 10^{-11}, we should multiply the reported result by {10^{-5}}^4 = 10^{-20} for CRC32C, and either 10^{-15} or 10^{-20} for IEEE802.3.

5.2 Checksums

For independent bit errors, Pud of CRC is approximately 12,000 better than Fletcher, and 22,000 better than Adler. For burst errors, by the simple examples that exist for three consecutive values that can produce an undetected burst, we take the factor to be at least the same.

If in three consecutive bytes, the error values are x, -2x, x then the error is undetected. Even for this error pattern alone, the conditional probability of undetected error, assuming a uniform distribution of data, is 2^-16 = 1.5 * 10^-5. The probability that a burst of length 3 bytes occurs, is f(24) = 3*10^-14. Together: 4.5*10^-19. Multiplying this by the length of the code, we get close to 4.5*10^-16, way worse than the vicinity of 10^-40.

The numbers in the table in Section 7 below reflect a more "tolerant" difference (10*4).

6. Incremental CRC Updates

In some protocols the packet header changes frequently. If the CRC includes the changing part, the CRC will have to be recomputed. This raises two issues:

- the complete computation is expensive
- the packet is not protected against unwanted changes between the last check and the recomputation

Fortunately, changes in the header do not imply a need for completed CRC computation. The reason is the linearity of the CRC function. Namely, with I1 and I2 denoting two equal-length blocks of information bits, CRC(I) denoting the CRC check bits calculated for I, and + denoting bitwise modulo-2 addition, we have CRC(I1+I2) = CRC(I1)+CRC(I2).

Hence, for an IP packet, made of a header h followed by data d followed by CRC bits c = CRC(h d), arriving at a node, which updates header h to become h', the implied update of c is an addition of CRC(h'-h 0), where 0 is an all 0 block of the length of the data block d, and addition and subtraction are bitwise modulo 2.

We know that a predetermined permutation of bits does not change distance and weight statistics of the codewords. It follows that such a transformation does not change the probability of undetected errors.

We can then conceive the packet as if it was built from data d followed by header h, compute the CRC accordingly, c=CRC(d h), and update at the node with an addition of CRC(0 h'-h)=CRC(h'-h), but on transmission, send the header part before the data and the CRC bits. This will allow a faster computation of the CRC, while still letting the header part lead (no change to the protocol).

Error detection, i.e., computing the CRC bits by the data and header parts that arrive, and comparing them with the CRC part that arrives together with them, can be done at the final, end-target node only, and the detected errors will include unwanted changes introduced by the intermediate nodes.

The analysis of the undetected error probability remains valid according to the following rationale:

The packet started its way as a codeword. On its way, several codewords were added to it (any information followed by the corresponding CRC is a codeword). Let e denote the totality of errors added to the packet, on its long, multi-hop journey. Because the code is linear (i.e., the sum of two codewords is also a codeword) the packet arriving to the end-target node is some codeword + e, and hence, as in our preceding analysis, e is undetected if and only if it is a codeword by itself. This fact is the basis of our above analysis, and hence that analysis applies here too. (See a detailed discussion at [braun01].)

7. Complexity of Hardware Implementation

Comparing the cost of various CRC polynomials, we used a tool available at http://www.easics.com/webtools/crctool to implement CRC generators/checkers for various CRC polynomials. The program gives either Verilog or VHDL code after specifying a polynomial, as well as the number of data bits, k, to be handled in one clock cycle. For a serial implementation, k would be one.

The cost for either one generator or checker is shown in the following table.

The number of 2-input XOR gates, for an un-optimized implementation, required for various values of k:

Polynomial	k=32	k=64	k=128
CCITT-CRC32	488	740	1430
IEEE-802	872	1390	2518
CRC32Q(Wolf)	944	1444	2534
CRC32C	1036	1470	2490

After optimizing (sharing terms) and in terms of Cells (4 cells per 2 input AND, 7 cells per 2 input XOR, 3 cells per inverter) the cost for two candidate polynomials is shown in the following table.

Polynomial	k=32	k=64
CCITT-CRC32	1855	3572
CRC32C	4784	7111

For 32-bit datapath, CCITT-CRC32 requires 40% of the number of cells required by the CRC32C. For a 64-bit datapath, CCITT-CRC32 requires 50% of the number of cells.

The total size of one of our smaller chips is roughly 1 million cells. The fraction represented by the CRC circuit is less than 1%.

8. Implementation of CRC32C

8.1 A Serial Implementation in Hardware

A serial implementation that processes one data bit at a time and performs simultaneous multiplication of the data polynomial by x^32 and division by the CRC32C polynomial is described in the following Verilog [ieee1364] code.

```
///////////////////////////////////////////////////////////////////////
//File: CRC32_D1.v
//Date: Tue Feb 26 02:47:05 2002
//
//Copyright (C) 1999 Easics NV.
//This source file may be used and distributed without restriction
//provided that this copyright statement is not removed from the file
//and that any derivative work contains the original copyright notice
//and the associated disclaimer.
//
//THIS SOURCE FILE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS
//OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
//WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
//
//Purpose: Verilog module containing a synthesizable CRC function
//* polynomial: (0 1 2 4 5 7 8 10 11 12 16 22 23 26 32)
//* data width: 1
//
//Info: jand@easics.be (Jan Decaluwe)
//http://www.easics.com
///////////////////////////////////////////////////////////////////////
module CRC32_D1;
// polynomial: (0 1 2 4 5 7 8 10 11 12 16 22 23 26 32)
// data width: 1
function [31:0] nextCRC32_D1;
input Data;
input [31:0] CRC;
reg [0:0] D;
reg [31:0] C;
reg [31:0] NewCRC;
begin
D[0] = Data;
C = CRC;
NewCRC[0] = D[0] ^ C[31];
NewCRC[1] = D[0] ^ C[0] ^ C[31];
NewCRC[2] = D[0] ^ C[1] ^ C[31];
NewCRC[3] = C[2];
NewCRC[4] = D[0] ^ C[3] ^ C[31];
NewCRC[5] = D[0] ^ C[4] ^ C[31];
NewCRC[6] = C[5];
NewCRC[7] = D[0] ^ C[6] ^ C[31];
NewCRC[8] = D[0] ^ C[7] ^ C[31];
NewCRC[9] = C[8];
NewCRC[10] = D[0] ^ C[9] ^ C[31];
NewCRC[11] = D[0] ^ C[10] ^ C[31];
NewCRC[12] = D[0] ^ C[11] ^ C[31];
NewCRC[13] = C[12];
NewCRC[14] = C[13];
```

```
NewCRC[15] = C[14];
NewCRC[16] = D[0] ^ C[15] ^ C[31];
NewCRC[17] = C[16];
NewCRC[18] = C[17];
NewCRC[19] = C[18];
NewCRC[20] = C[19];
NewCRC[21] = C[20];
NewCRC[22] = D[0] ^ C[21] ^ C[31];
NewCRC[23] = D[0] ^ C[22] ^ C[31];
NewCRC[24] = C[23];
NewCRC[25] = C[24];
NewCRC[26] = D[0] ^ C[25] ^ C[31];
NewCRC[27] = C[26];
NewCRC[28] = C[27];
NewCRC[29] = C[28];
NewCRC[30] = C[29];
NewCRC[31] = C[30];
nextCRC32_D1 = NewCRC;
end
endfunction
endmodule
```

8.2 A Parallel Implementation in Hardware

A parallel implementation that processes 32 data bits at a time is described in the following Verilog [ieee1364] code. In software implementations, the next state logic is typically implemented by means of tables indexed by the input and the current state.

```
//////////////////////////////////////////////////////////////////////
//File: CRC32_D32.v
//Date: Tue Feb 26 02:50:08 2002
//
//Copyright (C) 1999 Easics NV.
//This source file may be used and distributed without restriction
//provided that this copyright statement is not removed from the file
//and that any derivative work contains the original copyright notice
//and the associated disclaimer.
//
//THIS SOURCE FILE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS
//OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
//WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
//
//Purpose: Verilog module containing a synthesizable CRC function
//* polynomial: p(0 to 32) := "100000101111011000111011011110001"
//* data width: 32
//
//Info: jand@easics.be (Jan Decaluwe)
```

```
//http://www.easics.com
////////////////////////////////////////////////////////////////////////
module CRC32_D32;
// polynomial: p(0 to 32) := "100000101111011000111011011110001"
// data width: 32
// convention: the first serial data bit is D[31]
function [31:0] nextCRC32_D32;
input [31:0] Data;
input [31:0] CRC;
reg [31:0] D;
reg [31:0] C;
reg [31:0] NewCRC;
begin
D = Data;
C = CRC;
NewCRC[0] = D[31] ^ D[30] ^ D[28] ^ D[27] ^ D[26] ^ D[25] ^ D[23]
^
D[21] ^ D[18] ^ D[17] ^ D[16] ^ D[12] ^ D[9] ^ D[8] ^
D[7] ^ D[6] ^ D[5] ^ D[4] ^ D[0] ^ C[0] ^ C[4] ^ C[5] ^
C[6] ^ C[7] ^ C[8] ^ C[9] ^ C[12] ^ C[16] ^ C[17] ^
C[18] ^ C[21] ^ C[23] ^ C[25] ^ C[26] ^ C[27] ^ C[28] ^
C[30] ^ C[31];
NewCRC[1] = D[31] ^ D[29] ^ D[28] ^ D[27] ^ D[26] ^ D[24] ^ D[22]
^
D[19] ^ D[18] ^ D[17] ^ D[13] ^ D[10] ^ D[9] ^ D[8] ^
D[7] ^ D[6] ^ D[5] ^ D[1] ^ C[1] ^ C[5] ^ C[6] ^ C[7] ^
C[8] ^ C[9] ^ C[10] ^ C[13] ^ C[17] ^ C[18] ^ C[19] ^
C[22] ^ C[24] ^ C[26] ^ C[27] ^ C[28] ^ C[29] ^ C[31];
NewCRC[2] = D[30] ^ D[29] ^ D[28] ^ D[27] ^ D[25] ^ D[23] ^ D[20]
^
D[19] ^ D[18] ^ D[14] ^ D[11] ^ D[10] ^ D[9] ^ D[8] ^
D[7] ^ D[6] ^ D[2] ^ C[2] ^ C[6] ^ C[7] ^ C[8] ^ C[9] ^
C[10] ^ C[11] ^ C[14] ^ C[18] ^ C[19] ^ C[20] ^ C[23] ^
C[25] ^ C[27] ^ C[28] ^ C[29] ^ C[30];
NewCRC[3] = D[31] ^ D[30] ^ D[29] ^ D[28] ^ D[26] ^ D[24] ^ D[21]
^
D[20] ^ D[19] ^ D[15] ^ D[12] ^ D[11] ^ D[10] ^ D[9] ^
D[8] ^ D[7] ^ D[3] ^ C[3] ^ C[7] ^ C[8] ^ C[9] ^ C[10] ^
C[11] ^ C[12] ^ C[15] ^ C[19] ^ C[20] ^ C[21] ^ C[24] ^
C[26] ^ C[28] ^ C[29] ^ C[30] ^ C[31];
NewCRC[4] = D[31] ^ D[30] ^ D[29] ^ D[27] ^ D[25] ^ D[22] ^ D[21]
^
D[20] ^ D[16] ^ D[13] ^ D[12] ^ D[11] ^ D[10] ^ D[9] ^
D[8] ^ D[4] ^ C[4] ^ C[8] ^ C[9] ^ C[10] ^ C[11] ^
C[12] ^ C[13] ^ C[16] ^ C[20] ^ C[21] ^ C[22] ^ C[25] ^
C[27] ^ C[29] ^ C[30] ^ C[31];
NewCRC[5] = D[31] ^ D[30] ^ D[28] ^ D[26] ^ D[23] ^ D[22] ^ D[21]
^
```

```
   D[17] ^ D[14] ^ D[13] ^ D[12] ^ D[11] ^ D[10] ^ D[9] ^
   D[5] ^ C[5] ^ C[9] ^ C[10] ^ C[11] ^ C[12] ^ C[13] ^
   C[14] ^ C[17] ^ C[21] ^ C[22] ^ C[23] ^ C[26] ^ C[28] ^
   C[30] ^ C[31];
   NewCRC[6] = D[30] ^ D[29] ^ D[28] ^ D[26] ^ D[25] ^ D[24] ^ D[22]
   ^
   D[21] ^ D[17] ^ D[16] ^ D[15] ^ D[14] ^ D[13] ^ D[11] ^
   D[10] ^ D[9] ^ D[8] ^ D[7] ^ D[5] ^ D[4] ^ D[0] ^ C[0] ^
   C[4] ^ C[5] ^ C[7] ^ C[8] ^ C[9] ^ C[10] ^ C[11] ^
   C[13] ^ C[14] ^ C[15] ^ C[16] ^ C[17] ^ C[21] ^ C[22] ^
   C[24] ^ C[25] ^ C[26] ^ C[28] ^ C[29] ^ C[30];
   NewCRC[7] = D[31] ^ D[30] ^ D[29] ^ D[27] ^ D[26] ^ D[25] ^ D[23]
   ^
   D[22] ^ D[18] ^ D[17] ^ D[16] ^ D[15] ^ D[14] ^ D[12] ^
   D[11] ^ D[10] ^ D[9] ^ D[8] ^ D[6] ^ D[5] ^ D[1] ^
   C[1] ^ C[5] ^ C[6] ^ C[8] ^ C[9] ^ C[10] ^ C[11] ^
   C[12] ^ C[14] ^ C[15] ^ C[16] ^ C[17] ^ C[18] ^ C[22] ^
   C[23] ^ C[25] ^ C[26] ^ C[27] ^ C[29] ^ C[30] ^ C[31];
   NewCRC[8] = D[25] ^ D[24] ^ D[21] ^ D[19] ^ D[15] ^ D[13] ^ D[11]
   ^
   D[10] ^ D[8] ^ D[5] ^ D[4] ^ D[2] ^ D[0] ^ C[0] ^ C[2] ^
   C[4] ^ C[5] ^ C[8] ^ C[10] ^ C[11] ^ C[13] ^ C[15] ^
   C[19] ^ C[21] ^ C[24] ^ C[25];
   NewCRC[9] = D[31] ^ D[30] ^ D[28] ^ D[27] ^ D[23] ^ D[22] ^ D[21]
   ^
   D[20] ^ D[18] ^ D[17] ^ D[14] ^ D[11] ^ D[8] ^ D[7] ^
   D[4] ^ D[3] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^ C[3] ^ C[4] ^
   C[7] ^ C[8] ^ C[11] ^ C[14] ^ C[17] ^ C[18] ^ C[20] ^
   C[21] ^ C[22] ^ C[23] ^ C[27] ^ C[28] ^ C[30] ^ C[31];
   NewCRC[10] = D[30] ^ D[29] ^ D[27] ^ D[26] ^ D[25] ^ D[24] ^
   D[22] ^
   D[19] ^ D[17] ^ D[16] ^ D[15] ^ D[7] ^ D[6] ^ D[2] ^
   D[1] ^ D[0] ^ C[0] ^ C[1] ^ C[2] ^ C[6] ^ C[7] ^ C[15] ^
   C[16] ^ C[17] ^ C[19] ^ C[22] ^ C[24] ^ C[25] ^ C[26] ^
   C[27] ^ C[29] ^ C[30];
   NewCRC[11] = D[21] ^ D[20] ^ D[12] ^ D[9] ^ D[6] ^ D[5] ^ D[4] ^
   D[3] ^ D[2] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^ C[2] ^ C[3] ^
   C[4] ^ C[5] ^ C[6] ^ C[9] ^ C[12] ^ C[20] ^ C[21];
   NewCRC[12] = D[22] ^ D[21] ^ D[13] ^ D[10] ^ D[7] ^ D[6] ^ D[5] ^
   D[4] ^ D[3] ^ D[2] ^ D[1] ^ C[1] ^ C[2] ^ C[3] ^ C[4] ^
   C[5] ^ C[6] ^ C[7] ^ C[10] ^ C[13] ^ C[21] ^ C[22];
   NewCRC[13] = D[31] ^ D[30] ^ D[28] ^ D[27] ^ D[26] ^ D[25] ^
   D[22] ^
   D[21] ^ D[18] ^ D[17] ^ D[16] ^ D[14] ^ D[12] ^ D[11] ^
   D[9] ^ D[3] ^ D[2] ^ D[0] ^ C[0] ^ C[2] ^ C[3] ^ C[9] ^
   C[11] ^ C[12] ^ C[14] ^ C[16] ^ C[17] ^ C[18] ^ C[21] ^
   C[22] ^ C[25] ^ C[26] ^ C[27] ^ C[28] ^ C[30] ^ C[31];
   NewCRC[14] = D[30] ^ D[29] ^ D[25] ^ D[22] ^ D[21] ^ D[19] ^
```

```
   D[16] ^
   D[15] ^ D[13] ^ D[10] ^ D[9] ^ D[8] ^ D[7] ^ D[6] ^
   D[5] ^ D[3] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^ C[3] ^ C[5] ^
   C[6] ^ C[7] ^ C[8] ^ C[9] ^ C[10] ^ C[13] ^ C[15] ^
   C[16] ^ C[19] ^ C[21] ^ C[22] ^ C[25] ^ C[29] ^ C[30];
   NewCRC[15] = D[31] ^ D[30] ^ D[26] ^ D[23] ^ D[22] ^ D[20] ^
   D[17] ^
   D[16] ^ D[14] ^ D[11] ^ D[10] ^ D[9] ^ D[8] ^ D[7] ^
   D[6] ^ D[4] ^ D[2] ^ D[1] ^ C[1] ^ C[2] ^ C[4] ^ C[6] ^
   C[7] ^ C[8] ^ C[9] ^ C[10] ^ C[11] ^ C[14] ^ C[16] ^
   C[17] ^ C[20] ^ C[22] ^ C[23] ^ C[26] ^ C[30] ^ C[31];
   NewCRC[16] = D[31] ^ D[27] ^ D[24] ^ D[23] ^ D[21] ^ D[18] ^
   D[17] ^
   D[15] ^ D[12] ^ D[11] ^ D[10] ^ D[9] ^ D[8] ^ D[7] ^
   D[5] ^ D[3] ^ D[2] ^ C[2] ^ C[3] ^ C[5] ^ C[7] ^ C[8] ^
   C[9] ^ C[10] ^ C[11] ^ C[12] ^ C[15] ^ C[17] ^ C[18] ^
   C[21] ^ C[23] ^ C[24] ^ C[27] ^ C[31];
   NewCRC[17] = D[28] ^ D[25] ^ D[24] ^ D[22] ^ D[19] ^ D[18] ^
   D[16] ^
   D[13] ^ D[12] ^ D[11] ^ D[10] ^ D[9] ^ D[8] ^ D[6] ^
   D[4] ^ D[3] ^ C[3] ^ C[4] ^ C[6] ^ C[8] ^ C[9] ^ C[10] ^
   C[11] ^ C[12] ^ C[13] ^ C[16] ^ C[18] ^ C[19] ^ C[22] ^
   C[24] ^ C[25] ^ C[28];
   NewCRC[18] = D[31] ^ D[30] ^ D[29] ^ D[28] ^ D[27] ^ D[21] ^
   D[20] ^
   D[19] ^ D[18] ^ D[16] ^ D[14] ^ D[13] ^ D[11] ^ D[10] ^
   D[8] ^ D[6] ^ D[0] ^ C[0] ^ C[6] ^ C[8] ^ C[10] ^ C[11] ^
   C[13] ^ C[14] ^ C[16] ^ C[18] ^ C[19] ^ C[20] ^ C[21] ^
   C[27] ^ C[28] ^ C[29] ^ C[30] ^ C[31];
   NewCRC[19] = D[29] ^ D[27] ^ D[26] ^ D[25] ^ D[23] ^ D[22] ^
   D[20] ^
   D[19] ^ D[18] ^ D[16] ^ D[15] ^ D[14] ^ D[11] ^ D[8] ^
   D[6] ^ D[5] ^ D[4] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^ C[4] ^
   C[5] ^ C[6] ^ C[8] ^ C[11] ^ C[14] ^ C[15] ^ C[16] ^
   C[18] ^ C[19] ^ C[20] ^ C[22] ^ C[23] ^ C[25] ^ C[26] ^
   C[27] ^ C[29];
   NewCRC[20] = D[31] ^ D[25] ^ D[24] ^ D[20] ^ D[19] ^ D[18] ^
   D[15] ^
   D[8] ^ D[4] ^ D[2] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^ C[2] ^
   C[4] ^ C[8] ^ C[15] ^ C[18] ^ C[19] ^ C[20] ^ C[24] ^
   C[25] ^ C[31];
   NewCRC[21] = D[26] ^ D[25] ^ D[21] ^ D[20] ^ D[19] ^ D[16] ^ D[9]
   ^
   D[5] ^ D[3] ^ D[2] ^ D[1] ^ C[1] ^ C[2] ^ C[3] ^ C[5] ^
   C[9] ^ C[16] ^ C[19] ^ C[20] ^ C[21] ^ C[25] ^ C[26];
   NewCRC[22] = D[31] ^ D[30] ^ D[28] ^ D[25] ^ D[23] ^ D[22] ^
   D[20] ^
   D[18] ^ D[16] ^ D[12] ^ D[10] ^ D[9] ^ D[8] ^ D[7] ^
```

```
D[5] ^ D[3] ^ D[2] ^ D[0] ^ C[0] ^ C[2] ^ C[3] ^ C[5] ^
C[7] ^ C[8] ^ C[9] ^ C[10] ^ C[12] ^ C[16] ^ C[18] ^
C[20] ^ C[22] ^ C[23] ^ C[25] ^ C[28] ^ C[30] ^ C[31];
NewCRC[23] = D[30] ^ D[29] ^ D[28] ^ D[27] ^ D[25] ^ D[24] ^
D[19] ^
D[18] ^ D[16] ^ D[13] ^ D[12] ^ D[11] ^ D[10] ^ D[7] ^
D[5] ^ D[3] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^ C[3] ^ C[5] ^
C[7] ^ C[10] ^ C[11] ^ C[12] ^ C[13] ^ C[16] ^ C[18] ^
C[19] ^ C[24] ^ C[25] ^ C[27] ^ C[28] ^ C[29] ^ C[30];
NewCRC[24] = D[31] ^ D[30] ^ D[29] ^ D[28] ^ D[26] ^ D[25] ^
D[20] ^
D[19] ^ D[17] ^ D[14] ^ D[13] ^ D[12] ^ D[11] ^ D[8] ^
D[6] ^ D[4] ^ D[2] ^ D[1] ^ C[1] ^ C[2] ^ C[4] ^ C[6] ^
C[8] ^ C[11] ^ C[12] ^ C[13] ^ C[14] ^ C[17] ^ C[19] ^
C[20] ^ C[25] ^ C[26] ^ C[28] ^ C[29] ^ C[30] ^ C[31];
NewCRC[25] = D[29] ^ D[28] ^ D[25] ^ D[23] ^ D[20] ^ D[17] ^
D[16] ^
D[15] ^ D[14] ^ D[13] ^ D[8] ^ D[6] ^ D[4] ^ D[3] ^
D[2] ^ D[0] ^ C[0] ^ C[2] ^ C[3] ^ C[4] ^ C[6] ^ C[8] ^
C[13] ^ C[14] ^ C[15] ^ C[16] ^ C[17] ^ C[20] ^ C[23] ^
C[25] ^ C[28] ^ C[29];
NewCRC[26] = D[31] ^ D[29] ^ D[28] ^ D[27] ^ D[25] ^ D[24] ^
D[23] ^
D[15] ^ D[14] ^ D[12] ^ D[8] ^ D[6] ^ D[3] ^ D[1] ^
D[0] ^ C[0] ^ C[1] ^ C[3] ^ C[6] ^ C[8] ^ C[12] ^ C[14] ^
C[15] ^ C[23] ^ C[24] ^ C[25] ^ C[27] ^ C[28] ^ C[29] ^
C[31];
NewCRC[27] = D[31] ^ D[29] ^ D[27] ^ D[24] ^ D[23] ^ D[21] ^
D[18] ^
D[17] ^ D[15] ^ D[13] ^ D[12] ^ D[8] ^ D[6] ^ D[5] ^
D[2] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^ C[2] ^ C[5] ^ C[6] ^
C[8] ^ C[12] ^ C[13] ^ C[15] ^ C[17] ^ C[18] ^ C[21] ^
C[23] ^ C[24] ^ C[27] ^ C[29] ^ C[31];
NewCRC[28] = D[31] ^ D[27] ^ D[26] ^ D[24] ^ D[23] ^ D[22] ^
D[21] ^
D[19] ^ D[17] ^ D[14] ^ D[13] ^ D[12] ^ D[8] ^ D[5] ^
D[4] ^ D[3] ^ D[2] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^ C[2] ^
C[3] ^ C[4] ^ C[5] ^ C[8] ^ C[12] ^ C[13] ^ C[14] ^
C[17] ^ C[19] ^ C[21] ^ C[22] ^ C[23] ^ C[24] ^ C[26] ^
C[27] ^ C[31];
NewCRC[29] = D[28] ^ D[27] ^ D[25] ^ D[24] ^ D[23] ^ D[22] ^
D[20] ^
D[18] ^ D[15] ^ D[14] ^ D[13] ^ D[9] ^ D[6] ^ D[5] ^
D[4] ^ D[3] ^ D[2] ^ D[1] ^ C[1] ^ C[2] ^ C[3] ^ C[4] ^
C[5] ^ C[6] ^ C[9] ^ C[13] ^ C[14] ^ C[15] ^ C[18] ^
C[20] ^ C[22] ^ C[23] ^ C[24] ^ C[25] ^ C[27] ^ C[28];
NewCRC[30] = D[29] ^ D[28] ^ D[26] ^ D[25] ^ D[24] ^ D[23] ^
D[21] ^
```

```
D[19] ^ D[16] ^ D[15] ^ D[14] ^ D[10] ^ D[7] ^ D[6] ^
D[5] ^ D[4] ^ D[3] ^ D[2] ^ C[2] ^ C[3] ^ C[4] ^ C[5] ^
C[6] ^ C[7] ^ C[10] ^ C[14] ^ C[15] ^ C[16] ^ C[19] ^
C[21] ^ C[23] ^ C[24] ^ C[25] ^ C[26] ^ C[28] ^ C[29];
NewCRC[31] = D[30] ^ D[29] ^ D[27] ^ D[26] ^ D[25] ^ D[24] ^
D[22] ^
D[20] ^ D[17] ^ D[16] ^ D[15] ^ D[11] ^ D[8] ^ D[7] ^
D[6] ^ D[5] ^ D[4] ^ D[3] ^ C[3] ^ C[4] ^ C[5] ^ C[6] ^
C[7] ^ C[8] ^ C[11] ^ C[15] ^ C[16] ^ C[17] ^ C[20] ^
C[22] ^ C[24] ^ C[25] ^ C[26] ^ C[27] ^ C[29] ^ C[30];
nextCRC32_D32 = NewCRC;
end
endfunction
```

8.3 Some Hardware Implementation Comments

The iSCSI spec specifies that the most significant 32 bits of the data be complemented prior to performing the CRC computation. For most implementations of the CRC algorithm, such as the ones described here, which perform simultaneous multiplication by x^32 and division by the CRC polynomial, this is equivalent to initializing the CRC register to ones regardless of the CRC polynomial. For other implementations, in particular one that only performs division by the CRC polynomial (and for which the prescribed multiplication by x^32 is performed externally) initializing the CRC register to ones does not have the same effect as complementing the most significant 32 bits of the message. With such implementations, for the CRC32c polynomial, initializing the CRC register to 0x2a26f826 has the same effect as complementing the most significant 32 bits of the data. See reference [Tuikov&Cavanna] for more details.

8.4 Fast Hardware Implementation References

Fast hardware implementations start from a canonic scheme (as the one presented in 7.2) and optimize it based on different criteria. Two classic papers on this subject are [Albertengo1990] and [Glaise1997]. A more modern (and systematic) approach can be found in [Shie2001] and [Sprachman2001].

9. Summary and Conclusions

The following table is a summary of the error detection capabilities of the different codes analyzed. In the table, d is the minimal distance at block length block (in bits), i/byte - software instructions/byte, Table size (if table lookup needed), T-look number of lookups/byte, Pudb - Pud burst and Puds - Pud sporadic:

Code	d	Block	i/Byte	Tsize	T-look	Pudb	Puds
Fletcher32	3	2^19	2	-	-	10^-37	10^-36
Adler32	3	2^19	3	-	-	10^-36	10^-35
IEEE-802	3	2^16	2.75	2^18	0.5/b	10^-41	10^-40
CRC32C	3	2^31-1	2.75	2^18	0.5/b	10^-41	10^-40

The probabilities for undetected errors in the above table are computed assuming uniformly distributed data. For real data - that can be biased - [Stone98], checksums behave substantially worse than CRCs.

Considering the protection level it offers, the lack of sensitivity for biased data and the large block it can protect, we think that CRC32C is a good choice as a basic error detection mechanism for iSCSI.

Please observe also that burst errors characterized by a fixed average time will have a higher impact on error detection capability as the speed of the channels (machines and networks) increases. The only way to keep the Pud within bounds for the long-term is to reduce the BER by using better coding of lower levels of the channel.

10. Security Considerations

These codes detect unintentional changes to data such as those caused by noise. In an environment where an attacker can change the data, it can also change the error-detection code to match the new data. Therefore, the error-detection codes overviewed here do not provide protection against attacks. Indeed, these codes are not intended for security purposes; they are meant to be used within some application, and the application’s threat model and security design control the security considerations for the use of the CRC.

11. References and Bibliography

[Albertengo1990] G. Albertengo, R. Sisto, "Parallel CRC Generation IEEE Micro", Vol. 10, No. 5, October 1990, pp. 63-71.

[Arazi] B Arazi, "A commonsense Approach to the Theory of Error Correcting codes".

[Baicheva] T Baicheva, S Dodunekov and P Kazakov, "Undetected error probability performance of cyclic redundancy-check codes of 16-bit redundancy", IEEE Proceedings on Communications, 147:253-256, October 2000.

[Black] "Fast CRC32 in Software" by Richard Black, 1994, at www.cl.cam.ac.uk/Research/SRG/bluebook/21/crc/crc.html.

[Castagnoli93] Guy Castagnoli, Stefan Braeuer and Martin Herrman "Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits", IEEE Transact. on Communications, Vol. 41, No. 6, June 1993.

[braun01] Florian Braun and Marcel Waldvogel, "Fast Incremental CRC Updates for IP over ATM Networks", IEEE, High Performance Switching and Routing, 2001, pp. 48-52.

[FITS] "NASA FITS documents" at http://heasarc.gsfc.nasa.gov/docs/heasarc/ofwg/docs/general/checksum/node26.html.

[Fujiwara89] Toru Fujiwara, Tadao Kasami, and Shu Lin, "Error detecting capabilities of the shortened hamming codes adopted forerror detection in IEEE standard 802.3", IEEE Transactions on Communications, COM-37:986989, September 1989.

[Glaise1997] Glaise, R. J., "A two-step computation of cyclic redundancy code CRC-32 for ATM networks", IBM Journal of Research and Development, Volume 41, Number 6, 1997.

[ieee1364] IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language, IEEE Standard 1364-1995, December 1995.

[LinCostello] S. Lin and D.J. Costello, Jr., "Error Control Coding: Fundamentals and Applications", Englewood Cliffs, NJ: Prentice Hall, 1983.

[Peterson] W Wesley Peterson & E J Weldon - Error Correcting Codes - First Edition 1961/Second Edition 1972.

[RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[Ritter] Ritter, T. 1986. The Great CRC Mystery. Dr. Dobb’s Journal of Software Tools. February. 11(2): 26-34, 76-83.

[Polynomials] "Information on Primitive and Irreducible Polynomials" at http://www.theory.csc.uvic.ca/~cos/inf/neck/PolyInfo.html.

[RFC1146] Zweig, J. and C. Partridge, "TCP Alternate Checksum Options", RFC 1146, March 1990.

[RFC1950] Deutsch, P. and J. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

[Shie2001] Ming-Der Shieh, et. al, "A Systematic Approach for Parallel CRC Computations", Journal of Information Science and Engineering, Vol.17 No.3, pp.445-461.

[Sprachman2001] Michael Sprachman, "Automatic Generation of Parallel CRC Circuits", IEEE Design & Test May-June 2001.

[Stone98] J. Stone et. al., "Performance of Checksums and CRC’s over Real Data", IEEE/ACM Transactions on Networking, Vol. 6, No. 5, October 1998.

[Williams] Ross Williams - A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS widely available on the net - (e.g., ftp.adelaide.edu.au/pub/rocksoft/crc_v3.txt)

[Wolf82] J.K. Wolf, Arnold Michelson and Allen Levesque, "On the probability of undetected error for linear block codes", IEEE Transactions on Communications, COM-30: 317-324, 1982.

[Wolf88] J.K. Wolf, R.D. Blackeney, "An Exact Evaluation of the Probability of Undetected Error for Certain Shortened Binary CRC Codes", Proc. MILCOM - IEEE 1988.

[Wolf94J] J.K. Wolf and Dexter Chun, "The single burst error detection performance of binary cyclic codes", IEEE Transactions on Communications COM-42:11-13, January 1994.

[Wolf94O] Dexter Chun and J.K. Wolf, "Special Hardware for computing the probability of undetected error for certain binary crc codes and test results", IEEE Transactions on Communications, COM-42:2769-2772.

[Tuikov&Cavanna] Luben Tuikov and Vicente Cavanna, "The iSCSI CRC32C Digest and the Simultaneous Multiply and Divide Algorithm", January 30, 2002. White paper distributed to the IETF ips iSCSI reflector.

12. Acknowledgements

We would like to thank Matt Wakeley for providing us with the motivation to co-author this paper and for helpful discussions on the subject matter, during his employment with Agilent.

13. Authors' Addresses

Julian Satran
IBM, Haifa Research Lab
MATAM - Advanced Technology Center
Haifa 31905, Israel
EMail: julian_satran@il.ibm.com

Dafna Sheinwald
IBM, Haifa Research Lab
MATAM - Advanced Technology Center
Haifa 31905, Israel
EMail: Dafna_Sheinwald@il.ibm.com

Pat Thaler
Agilent Technologies
1101 Creekside Ridge Drive
Suite 100, M/S RH21
Roseville, CA 95661
EMail: pat_thaler@agilent.com

Vicente Cavanna
Agilent Technologies
1101 Creekside Ridge Drive
Suite 100, M/S RH21
Roseville, CA 95661
EMail: vince_cavanna@agilent.com

14. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.