Computer Networks
Error detection

Saad Mneimneh
Computer Science
Hunter College of CUNY
New York

par·i·ty: the quality or state of being
equal or equivalent

1 Introduction

The existence of an error in the frame cannot be detected by analyzing the frame itself (why?). Therefore, extra bits must be used. These bits are added by the DLC as a trailer (see Figure 1), often known as a *parity*. The parity is a function of the data in the frame, and by checking the received parity against the received frame, the receiving DLC is able to detect whether certain types of errors have occurred.

All framing techniques are sensitive to errors. For example, if an error occurs in DLE ETX, the receiver will not detect the end of the frame. A similar problem is that errors can cause the appearance of DLE ETX in the data itself; the receiver would interpret this as the end of the frame. In both cases, the perceived parity is essentially a random bit string in relation to the perceived frame, and the receiver fails to detect the error with a probability of 2^{-L}, where L is the length of the parity. The same problem occurs in bit oriented framing with the flag disappearing or appearing due to errors. This problem is often called the data sensitivity problem of the DLC, since even though the parity is capable of detecting errors, a single error that creates or destroys a flag, plus a special combination of data bits to satisfy the perceived parity, causes an undetectable error.

Similarly, an error in the length field causes the frame to be determined at the wrong point, and the receiver to look for the parity in the wrong place. The probability of such an error is smaller using a length field than using a flag (since errors can create false flags); however, an error in the length field makes it hard for the receiver to know where to look for subsequent frames. Re-synchronization after every frame is required for recovery in order to read the frames correctly, but this makes the length field redundant.

Parity check is used by the receiving DLC to determine if a frame contains errors. If a frame is found to contain errors, the receiver requires the sender to retransmit the frame (that's mainly how error correction is performed). The following sections describe different parity check mechanisms.

2 Parity check codes

2.1 Single parity check

A one bit parity is added to the frame. The parity bit is 1 if the frame contains an odd number of ones, and 0 otherwise. Here are two examples showing the added parity bit in a box.

1001010 $\boxed{1}$
0111010 $\boxed{0}$

After adding the parity bit, a frame contains an even number of ones. Therefore, the receiver counts the number of ones. If the number of ones is odd, an error must have occurred. If the number of ones is even, the receiver *interprets* this as no error, because an even number of errors cannot be detected. Therefore, the probability of not detecting errors is:

$$Pr(\textit{undetected error}) = \sum_{i>0} \binom{k}{2i} p^{2i}(1-p)^{k-2i}$$

assuming independent errors (simplification), where k is the length of the frame, p is the probability of a bit flipping due to error (binary symmetric channel). For the rigorous, this is

$$\frac{1+(1-2p)^k}{2} - (1-p)^k$$

2.2 Horizontal and vertical parity checks

Data is logically visualized as a rectangular array with a number of rows and columns as shown below.

1	0	0	1	0	1	0
0	1	1	1	0	1	0
1	1	1	0	0	0	1
1	0	0	0	1	1	1
0	0	1	1	0	0	1

A parity bit is computed for every row and every column as before. If an even number of errors is confined to a single row, each of them can be detected by the corresponding column parity check (and vice-versa). The parity bits for the different rows and columns are shown below in boxes:

1	0	0	1	0	1	0	1
0	1	1	1	0	1	0	0
1	1	1	0	0	0	1	0
1	0	0	0	1	1	1	0
0	0	1	1	0	0	1	1
1	0	1	1	1	1	1	0

horizontal checks

← always consistent with both checks (why? Hint: addition modulo 2)

The last bit is the equivalent of a single parity check. Even with horizontal and vertical parity checks, some errors are still undetected, e.g. 4 errors forming the corners of a rectangle (check it).

2.3 Arbitrary parity check codes

From what we have seen so far, a parity bit is simply the result of an addition modulo 2. Multiple additions modulo 2 can be performed, resulting in multiple parity bits. The horizontal and vertical parity checks are an example of a parity check code obtained this way. This notion can be generalized:

$$\underbrace{s_0\ s_1\ \ldots\ s_{K-1}}_{K\text{bit frame}}\underbrace{c_0\ c_1\ \ldots\ c_{L-1}}_{L\text{bit parity check}}$$

where every c_j is the sum modulo 2 of some bits in $s_0 \ldots s_{K-1}$

$$c_i = (\sum_{j=0}^{K-1} \alpha_{ij} s_j) \bmod 2$$

where α is an $L \times K$ 0-1 matrix (row and column indices start at 0).

Here's are some example with a four bit parity check code:

s_0	s_1	s_2	c_0	c_1	c_2	c_3
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	0	1	1	1
0	1	1	1	0	1	0
1	0	0	1	1	1	0
1	0	1	0	0	1	1
1	1	0	1	0	0	1
1	1	1	0	1	0	0

$$c_0 = s_0 + s_2$$
$$c_1 = s_0 + s_1 + s_2$$
$$c_2 = s_0 + s_1$$
$$c_3 = s_1 + s_2$$

$$\alpha = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

3 Effectiveness of a code

Before we study a special type of parity check code known as Cyclic Redundancy Check (CRC), let us examine ways to measure the effectiveness of a code. We denote by the concatenation of the data bits and the parity bits as the *codeword*. For instance, if we have K data bits and L parity bits, the codeword contains $K + L$ bits, as shown below:

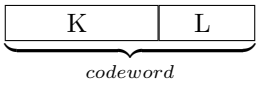

Figure 1: Codeword

The effectiveness of a code is usually measured by three parameters.

3.1 Minimum distance

The minimum distance of the code is defined as the smallest number of errors that can convert one codeword into another.

3.2 Burst detecting capability

The length of a burst of errors in a frame is the number of bits from the first error to the last, inclusive. The burst detecting capability of a code is defined as the largest integer B such that a code can detect all bursts of length $\leq B$.

3.3 Data sensitivity

The data sensitivity is defined as the probability that a random string will be accepted as error-free. This measure is useful when framing is lost, e.g. the check code looks random with respect to the received frame. We have 2^K codewords (why?) and 2^{K+L} random strings. Therefore, this probability is 2^{-L}.

	minimum distance	burst detecting capability	data sensitivity
single parity check	2	1	2^{-1}
horizontal and vertical parity checks (m rows, n columns)	4	1 + n (why?) (assuming rows are sent one after the other)	$2^{-(m+n-1)}$

Figure 2: Example code effectiveness measures

4 Cyclic Redundancy Check

Denote the data bits (K bits) as

$$s_{K-1}, s_{K-2}, \ldots, s_0$$

Similarly, denote the CRC bits (L bits) as

$$c_{L-1}, c_{L-2}, \ldots, c_0$$

We can represent the data as a polynomial

$$s(x) = s_{K-1}x^{K-1} + s_{K-2}x^{K-2} + \ldots + s_1 x + s_0$$

and the CRC as another polynomial

$$c(x) = c_{L-1}x^{L-1} + c_{L-2}x^{L-2} + \ldots + c_1 x + c_0$$

The whole frame can now be represented as a polynomial

$$f(x) = s(x)x^L + c(x) = \underbrace{s_{K-1}}\, x^{L+K-1} + \ldots + \underbrace{s_0}\, x^L + \underbrace{c_{L-1}}\, x^{L-1} + \ldots + \underbrace{c_0}$$

Why this polynomial representation? Because we are going to obtain the CRC as $c(x)$ by dividing $s(x)x^L$ by a given polynomial of degree L, $g(x) = x^L + g_{L-1}x^{L-1} + \ldots + g_1 x + 1$. The result is a polynomial of degree $L-1$, i.e. the representation of an L bit CRC.

$$c(x) = Remainder\left[\frac{s(x)\cdot x^L}{g(x)}\right]$$

where the division is carried as a polynomial division modulo 2. Therefore, all coefficients are either 0 or 1. Note that addition and subtraction modulo 2 are the same:

$$0+0=0 \qquad 0+1=1 \qquad 1+0=1 \qquad 1+1=0$$

$$0-0=0 \qquad 0-1=1 \qquad 1-0=1 \qquad 1-1=0$$

Therefore, polynomial division modulo 2 is easy to carry out as shown in the following examples.

4.1 Example 1: $s = 101$ $(K = 3)$, $g(x) = x^3 + x^2 + 1$ $(L = 3)$

$$s(x) = x^2 + 1$$
$$s(x)\cdot x^L = x^5 + x^3$$

$$\begin{array}{l|l} x^5 + x^3 & x^3 + x^2 + 1 \\ \hline x^5 + x^4 + x^2 & x^2 + x \\ \hline x^4 + x^3 + x^2 & \\ \hline x^4 + x^3 + x & \\ \hline x^2 + x & \end{array}$$

$c(x) = x^2 + x \Rightarrow c = 110$

4.2 Example 2: $s = 110101$ $(K = 5)$, $g(x) = x^3 + 1$ $(L = 3)$

$$s(x) = x^5 + x^4 + x^2 + 1$$
$$s(x)x^L = x^8 + x^7 + x^5 + x^3$$

$$\begin{array}{l|l} x^8 + x^7 + x^5 + x^3 & x^3 + 1 \\ \hline x^8 + x^5 & x^5 + x^4 + x + 1 \\ \hline x^7 + x^3 & \\ \hline x^7 + x^4 & \\ \hline x^4 + x^3 & \\ \hline x^4 + x & \\ \hline x^3 + x & \\ \hline x^3 + 1 & \\ \hline x + 1 & \end{array}$$

$c(x) = x + 1 \Rightarrow c = 011$ $(L = 3)$.

There are two questions that need to be answered:

- Why is CRC good?
- How is CRC a parity check code?

Before we answer these questions, let us look at a practical way of computing CRC using bits only. We will use Example 2 above to illustrate the idea. First, leave s as a bit string, i.e. do not bother in obtaining its polynomial representation. The polynomial $s(x) \cdot x^L$ corresponds to s shifted L positions to the left by inserting zeros. Similarly, $g(x)$ is also replaced by its bit string representation (note that the coefficients of $g(x)$ are effectively zeros and ones because of modulo 2 arithmetics).

$$
\begin{array}{rl}
s(x) \equiv & \begin{array}{cccccc} 1 & 1 & 0 & 1 & 0 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ x^5 & x^4 & x^3 & x^2 & x^1 & x^0 \end{array} \\
s(x)x^L \equiv & \begin{array}{ccccccccc} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ x^8 & x^7 & x^6 & x^5 & x^4 & x^3 & x^2 & x^1 & x^0 \end{array} \\
g(x) \equiv & \begin{array}{cccc} 1 & 0 & 0 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ x^3 & x^2 & x^1 & x^0 \end{array}
\end{array}
$$

Now we carry out the same polynomial division by keeping track of the bits instead of the actual exponents.

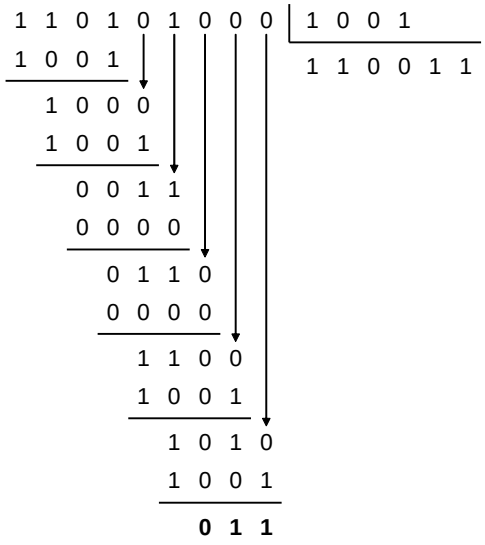

Figure 3: Polynomial division modulo 2 using bits

By observing the division, we note that we are performing the following algorithm:

1. given $L+1$ bits of s
2. multiply g by the leading bit
3. add the result to the $L+1$ bits (modulo 2)
4. shift and repeat

Note also that every addition results in a zero for the leading bit (that's because g start with a one). Therefore, the leading bit of g can be safely ignored. This algorithm can be easily implemented using a feedback shift register:

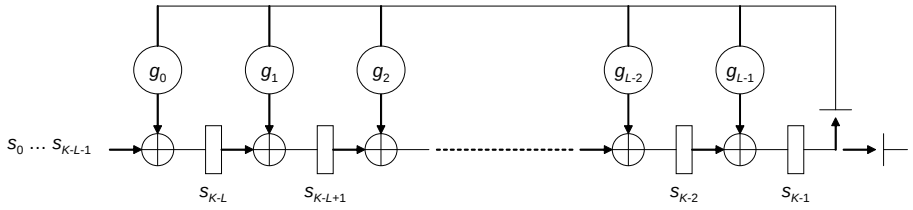

Figure 4: Feedback shift register. The register is initialized with the first L bits of s. After K shifts, the switch is moved and the CRC is read.

4.3 Why is CRC good?

Let us recall how the CRC is computed. It is the remainder of the polynomial division modulo 2 of $s(x) \cdot x^L$ by $g(x)$. Therefore, after adding the CRC to the frame, the frame can be represented as a polynomial $f(x)$:

$$f(x) = s(x) \cdot x^L + c(x) = [g(x)z(x) + c(x)] + c(x) \pmod 2$$

Since our arithmetic is done modulo 2, $c(x) + c(x) = 2c(x) = 0$, and hence:

$$f(x) = g(x)z(x) \pmod 2$$

Therefore, $f(x)$ is a multiple of $g(x)$ modulo 2. The receiver performs a division modulo 2 of $f(x)$ (what is received) by $g(x)$. If the remainder is zero, the frame is declared error-free [1].

When does CRC fail to detect errors? If the frame is received with errors, the receiver observes $y(x) = f(x) + e(x)$ instead of $f(x)$, i.e. the error can be viewed as an additive polynomial (of course $e(x)$ is not known to the receiver). When the receiver divides by $g(x)$ it obtains:

$$\frac{y(x)}{g(x)} = \frac{f(x)}{g(x)} + \frac{e(x)}{g(x)}$$

The receiver will declare the frame to be error-free only if $y(x)$ is a multiple of $g(x)$. Therefore, we have undetected errors only if $e(x)$ is a multiple of $g(x)$.

[1] Is the following strategy equivalent: the receiver performs a polynomial division modulo 2 of $s(x) \cdot x^L$ (frame without CRC) by $g(x)$ and checks if the remainder is $c(x)$?

Based on the above analysis, we can show that single errors are always detected. Let $e(x) = x^i$ for some $0 \leq i \leq K + L - 1$. The error is not detected means that $e(x)$ is also a multiple of $g(x)$; therefore:

$$g(x)z(x) = x^i \pmod 2$$

But this is not possible because $g(x) = x^L + \ldots + 1$, and multiplying $g(x)$ by any non-zero polynomial $z(x)$ must produce at least two terms (why?). Using the same type of argument, we can show that the burst detecting capability of CRC is L. In addition, the polynomial $g(x)$ can be chosen such that all double errors are detected and all odd number of errors are detected. Examples of such polynomials follow:

$\underline{L = 16}$

$g(x) = x^{16} + x^{15} + x^2 + 1$ CRC-16

$g(x) = x^{16} + x^{12} + x^5 + 1$ CRC-CCITT

$\underline{L = 32}$

$g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$

4.4 How is CRC a parity check code?

A CRC looks very different from a parity check code; however, a careful examination will reveal that CRC is nothing but a parity check code. Recall that for a general parity check code, the i^{th} bit is given by:

$$c_i = (\sum_{j=0}^{K-1} \alpha_{ij} s_j) \bmod 2$$

where $s = s_0 \ldots s_{K-1}$ is the data and α is an $L \times K$ 0-1 matrix (row and column indices start at 0).

We can construct an $L \times K$ 0-1 matrix such that the i^{th} bit of a CRC satisfies the above equation, for all $i = 0 \ldots L - 1$. $s(x) = \sum_{j=0}^{K-1} s_j x^j$. Therefore, $s(x) \cdot x^L = \sum_{j=0}^{K-1} s_j x^{j+L}$. Now consider the polynomial division modulo 2 of x^{j+L} by $g(x)$ for some j.

$$x^{j+L} = g(x)z_j(x) + c_j(x) \pmod 2$$

Note that the highest degree in $c_j(x)$ is less than L. Multiplying by s_j and summing over j, we get:

$$\sum_{j=0}^{K-1} s_j x^{j+L} = g(x) \sum_{j=0}^{K-1} s_j z_j(x) + \sum_{j=0}^{K-1} s_j c_j(x) \pmod 2$$

Let $z(x) = (\sum_{j=0}^{K-1} s_j z_j(x)) \bmod 2$ and $c(x) = (\sum_{j=0}^{K-1} s_j c_j(x)) \bmod 2$. Then,

$$s(x) \cdot x^L = g(x)z(x) + c(x) \pmod 2$$

Since the highest degree in $c(x)$ is less than L, $c(x)$ must be the remainder of the polynomial division modulo 2 of $s(x) \cdot x^L$ by $g(x)$, i.e. the CRC. But we have established that:

$$c(x) = (\sum_{j=0}^{K-1} s_j c_j(x)) \bmod 2$$

Let α_{ij} be the coefficient of the i^{th} exponent of $c_j(x)$. Two polynomials are equal if and only if their coefficients are equal; therefore, if we denote the i^{th} coefficient of $c(x)$ by c_i, we have:

$$c_i = (\sum_{j=0}^{K-1} \alpha_{ij} s_j) \bmod 2$$

References

Dimitri Bertsekas and Robert Gallager, Data Networks
Larry Peterson and Bruce Davie, Computer Networks: A Systems Approach