Error Detection with the CRC

The use of simple parity allows detection of single bit errors in a received message. However, a second error will go undetected, and further errors will not be noticed whenever the total number of bit errors is EVEN. Unless the probability of a error is very low *and* a message is very short (the case when a parity bit is added to a 7-bit ASCII character) the chances of some error event going undetected are high. One of the most useful techniques is called the **Cyclic Redundancy Check** (CRC).

The Cyclic Redundancy Check (CRC)

Consider a message having many data bits which are to be transmitted reliably by appending several check bits as shown below.

message bits	CRC bits

The exact number of CRC bits and their makeup depends on a generating polynomial. For example one such polynomial is:

$$x^{16} + x^{12} + x^{5} + 1$$

A Standard Generating Polynomial - CRC(CCITT)

The number of CRC bits corresponds to the order of the generating polynomial. The above polynomial generates a 16-bit CRC. Typically, the CRC bits are used for error *detection* only.

CRC Computation

Consider a message represented by some polynomial G(x), and a generating polynomial P(x).

In this example, let G(x) represent the binary message 110010, and let $P(x) = x^3 + x^2 + 1$; (binary **1101**).

The polynomial P(x) will be used to generate a (3-bit) CRC called C(x) which will be appended to G(x). Note that P(x) is prime.

110010	CRC

The polyomial P(x) defines the CRC bits.

Step 1 - Multiply the message G(x) by x^3, where 3 is the number of bits in the CRC.

Add 3 three zeros to the binary G(x).

Step 2 - Divide the product x^3 [G(x)] by the generating polynomial P(x).

We wish to find "the remainder, modulo P(x)"

Compute the following:

```
          100100  (ignore this quotient)
        ------------
 1101 ) 110010000
        1101
        ----
          1100
          1101
          ----
           100   = remainder = C(x)
```

Observe that if C(x) were in place of the appended zeros, the remainder would become 000.

Consider the decimal number 41 and the prime divisor 13. **Q:** What can be done to 41 to make the result divisible by 13? **A:** It is clear that 41 = 2 mod 13. On the other hand, (41-2) = 0 mod 13. In general, if the division X/Y gives a remainder Z, then (X-Z)/Y gives remainder zero.

Step 3 - Disregard the quotient and add the remainder C(x)to the product x^3 [G(x)] to yield the code message polynomial F(x), which is represented as:

$$F(x) = x^3 [G(x)] + C(x)$$

Put the remainder C(x)=100 in place of the three zeros added in Step 1.

110010	100

The message may now be transmitted

CRC Error Checking - No Errors

Upon reception, the entire received F(x) = "message + crc" can be checked simply by dividing F(x)/P(x) using the same generatingpolynomial. If theremainder after division equals zero, then no error was found.

```
          100100  (ignore this quotient)
        ------------
 1101 ) 110010100
        1101
        ----
          1101
          1101
```

```
    ----
     000   = remainder (no error)
```

CRC Error Checking - Single Bit Error

A single bit error in bit position K in a message F(x) can be represented by adding the term E(x) = x^K, (binary 1 followed by K-zeros).

```
    sent: 110010100  =  F(x)
   error: 000001000  =  E(x) = x^3
          ---------
received: 110011100  =  F(x) + E(x)    (error in bit 3)
```

The above error would be detected when the CRC division is performed:

```
          100101  (ignore this quotient)
      ------------
1101 ) 110011100  = F(x) + E(x)
       1101
       ----
        1111
        1101
        ----
         1000
         1101
         ----
          101   = remainder (error!)
```

Note that division by P(x) revealed the error. On the other hand, since F(x)/P(x) = 0 by definition, the remainder is a function only of the error. An error in this same bit would give the same non-zero remainder *regardless of the message bits*.

$$\frac{F(x) + E(x)}{P(x)} = \frac{F(x)}{P(x)} + \frac{E(x)}{P(x)} = \frac{E(x)}{P(x)}$$

The remainder is a function only of the errored bits E(x).

```
          1  (ignore this quotient)
      ------------
1101 ) 000001000   = E(x) alone
       1101
       ----
        101   = remainder (error!)
```

Since $E(x) = x^K$ has no factors other than x, a single bit error will never produce a term exactly divisible by P(x). **All single bit errors will be detected.**

CRC Error Checking - Double Bit Error

Similarly if $E(x) = x^K + x^{K+1}$, the error pattern includes two adjacent bits (e.g. E(x) = 000011000 for K=3).

Since this E(x) has no factors other than (x+1) and x, a double bit error will never produce a term exactly divisible by P(x). **All double bit (adjacent-bit) errors will be detected**

The above argument can be extended to other types of errors, and the error performance can be fully described by examining E(x) independently of any data messages.

CRC Error Checking - Undetected Errors

From the above discussion, any bit error term E(x) which is an exact multiple of P(x) will *not* be detected.

This is the case, in particular, for the two bit error **10000001**, where the two bad bits are 7-bits apart. Note that 10000001 =(1011)(1101)(11). The allowable separation between two bad bits is related to the choice of P(x).

In general, bit errors and bursts up to N-bits long will be detected for a prime P(x) of order N. For arbitrary bit errors longer than N-bits, the odds are one in 2^N than a totally false bit pattern will nonetheless lead to a zero remainder.

In essence, 100% detection is assured for all errors E(x) not an exact multiple of P(x). For a 16-bit CRC, this means:

- 100% detection of single-bit errors;
- 100% detection of all adjacent double-bit errors;
- 100% detection of any errors spanning up to 16-bits;
- 100% detection of all two-bit errors not separated by exactly 2^{16}-1 bits (this means *all* two bit errors in practice!);
- For arbitrary multiple errors spanning more than 16 bits, at worst 1 in 2^{16} failures, which is nonetheless over 99.995% detection rate.