Cyclic Redundancy Check (CRC)

A powerful way to detect errors in transmission is by attaching a fixed number of digits to the lower (in terms of significance) end of the data. These bits are the Cyclic Redundancy Check (CRC) bits.

The well-known concept of integer division forms the basis for the use of CRCs. When a dividend is divided by a divisor, a quotient and a remainder (which may be 0) result. If the remainder is subtracted from the dividend and this is divided by the divisor, the remainder is always zero. e.g. when 723 is divided by 5, the remainder is 3. If the dividend (723) and the remainder (3) are transmitted from a source to a target, the integrity of the transmission can be verified at the target by recomputing the remainder and verifying that the remainder matches the transmitted remainder. Alternatively, the target could divide the difference between the transmitted dividend and remainder and expect to see a zero remainder if there were no errors.

The concept of integer division can also be applied to division of polynomials. (An intuitive way to understand this is by considering that the digits which make up an integer can be considered the coefficients of a polynomial in base 10 e.g. $723 = 7*10^2 + 2* 10^1 + 3*10^0$.). A binary bitstream (which is a pattern of 1s and 0s) can be considered to represent the coefficients of a (dividend) polynomial. When this polynomial is divided by a generator (divisor) polynomial (which is another binary bitstream) a remainder polynomial (CRC) will result. The arithmetic is especially simplified if 1 and 0 are considered to be elements of a finite field (the Galois field of order 2 or GF(2)) . The arithmetic is sometimes referred to as modulo 2 arithmetic. For the purposes of CRC computation, it is sufficient to understand that addition and subtraction in this field reduce to simple XOR operations.

To clarify the ideas behind the CRC method of error checking a block of data, let us focus on a specific case.

Consider a block of data, say, D=1010111010101. Suppose that a 5-bit CRC field , say R, will be attached to this. Both the transmitter and the receiver should agree that the lower 5 bits are CRC bits. They further have to agree on a 4th degree polynomial called the "Generator", G, that is used in computing CRC bits.

$$\underbrace{10101010111001}_{\text{Data (D)}}\ \underbrace{-----}_{\text{CRC-bits (R)}} \rightarrow \text{transmitted data}$$

Figure 1: CRC Codes

CRC calculations are done in modulo 2 arithmetic, without carries in addition and borrows in subtraction. This means that addition and subtraction are identical, and both are equivalent to the bitwise exclusive or, XOR, of the operands.

For example,

11001 XOR 01011 = 10010
11001 + 01011 = 10010
11001 - 01011 = 10010

10010 XOR 01011 = 11001

Notice that if A XOR B = C, then A= C XOR B.

Remember that a left shift by 5, of binary data D, makes it $D.2^5$. Then the transmitted data can be represented as $D.2^5$ XOR R.

In general, we may suppose that there are *d* bits of data *D* to be transmitted and we use a *r*-digit CRC filed, R. So *d+r* bits are transmitted, represented mathematically by $D.\ 2^r$ XOR R.

How do we choose the r bits of R? If the transmitter and receiver agrees on an *r+1* bit pattern, G, then the *r* bits could be the remainder when $D.\ 2^r$ is divided by G. The most significant digit of G is assumed to be 1. To see this, suppose that we must choose the r bits of R so that the transmitted bits, $D.\ 2^r$ XOR R, is divisible by G. This means that

$D.\ 2^r$ XOR R= n G

where n is a (binary) integer. It follows that

$D.\ 2^r$ = nG XOR R

This implies that if $D.\ 2^r$ is divided by G, the remainder is R. So, in other words, we can calculate the *r* bits of R as:

$$R = Remainder \frac{D.2^r}{G}$$

The calculations are done with modulo-2 arithmetic using XOR operations. In this context, to divide 110001 by 111, we simply apply the bit-wise exclusive-OR operation repeatedly as follows

```
          1011
        ______
    111 |110001
         111
         ---
         0010
          000
          ---
          0100
           111
           ----
           0111
            111
            ---
            000
```

This is exactly like ordinary long division, only simpler, because at each stage we just need to check whether the leading bit of the current three bits is 0 or 1. If it's 0, we place a 0 in the quotient and XOR the current bits with 000. If it's 1, we place a 1 in the quotient and XOR the current bits with the divisor, which in this case is 111. As can be seen, the result of dividing 110001 by 111 is 1011, leaving a remainder of 000.

Two frequently-used generator polynomials are 16 and 32 bits long. The polynomial $x^{16}+x^{12}+x^5+1$ or, in binary form, 10001000000100001, is known as the "X25 standard" . The polynomial
$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$
is called the CRC-32 or "Ethernet standard" .

If we assume that any corruption of data affects the transmitted string in a completely random way, i.e., such that the corrupted string is totally uncorrelated with the original string, then the probability of a corrupted string going undetected is 1/(2^n), with an n-bit CRC field. Thus, a 16-bit CRC has a probability of 1/(2^16) = $1.5\text{x}10^{-5}$ of failing to detect an error in the data, and a 32-bit CRC has a probability of 1/(2^32), which is about $2.3\text{x}10^{-10}$ (less than one in a billion).

The next page shows a sample CRC calculation.

Sample CRC Calculation

Data: 1101101

Generator: 111 $(x^2 + x + 1)$

There will be 2 bits in CRC field

```
           1010001
     111 | 1101101 00
           111       CRC bits go here
           ---
            011
            000
            ---
             111
             111
             ---
              000
              000
              ---
               001
               000
               ---
                010
                000
                ---
                 100
                 111
                 ---
                  11  remainder
                      = CRC bits
```

Transmitted Data: 1101101 11

Receiver's end:

```
           1010001
     111 | 1101101 11
           111
           ---
            011
            000
            ---
             111
             111
             ---
              000
              000
              ---
               001
               000
               ---
                011
                000
                ---
                 111
                 111
                 ---
                  00  remainder is 0, if transmission is OK
```