CRC of Ethernet

CRC is an error check technique used to identify if an error has occurred in a transmitted data or not. In its simplest form, CRC is calculated for transmitted data by getting remainder of a modulo2 division of the data by a predefined polynomial. This polynomial is called generator polynomial and it can be of any degree. CRC for Ethernet is the main focus in this content without discussion about much theory behind CRC.

In general, an n-bit CRC is calculated by representing the data stream as a polynomial M(x). Then this M(x) is multiplied by x^n, where n is the degree of the Generator polynomial G(x). Then dividing the result by the Generator polynomial G(x). The resulting remainder is appended to the polynomial M(x) and transmitted. At the receiver side, The complete transmitted polynomial is then divided by the same Generator polynomial to get the resulting remainder which shall be a constant.

Multiplication of M(x) by x^n means trailing n-bits zeros to the end of M(x).

For example assume that M(x) is 10111. Which is as a polynomial shall be written as $M(x) = x^4 + x^2 + x^1 + 1$. If we multiplied it by x^4 this will result in

$F(x) = M(x) * x^4$

$F(x) = (x^4 + x^2 + x^1 + 1) * x^4$

$F(x) = x^8 + x^6 + x^5 + x^4$

so the resulted F(x) is represented in binary as 101110000. For Ethernet protocol, we multiply data to be sent first by x^{32} to leave space for appending the resulted CRC in this trailing space.

IEEE 802.3 defines the polynomial M(x) as the destination address, source address, length/type, and data of a frame. To generate CRC of this M(x), the first 32-bits are ones complemented.

To describe complement operation of the first 32-bit mathematically, assume the data to be sent is k bit length. i.e. M(x) has k bits. So the multiplication and complement operation can be described as follows

$$x^{32}M(x) + x^kL(x) \tag{1}$$

where L(x) is 32 successive ones.

Next Operation is to divide by the Generator polynomial G(x) as a modulo2 division, which can be described mathematically as follows

$$[x^{32}M(x) + x^kL(x)]/G(x) = Q(x) + R(x)/G(x) \tag{2}$$

where Q(x) are a quotient. R(x) are the remainder of the division operation.

IEEE 802.3 defines the CRC that will be appended to the message is the one's complement of the remainder R(x)

$$CRC_{tx} = L(x) + R(x) \tag{3}$$

The transmitted message T(x) can be represented by

$$T(x) = x^{32}M(x) + CRC_{tx} \tag{4}$$

Note that here T(x) is k+32 bits.

substituting CRC as in 3 we can write T(x) as:

$$T(x) = x^{32}M(x) + L(x) + R(x) \tag{5}$$

At the receiver, we are interested in obtaining the remainder of the received message so that we make sure if it has been received correctly or not. So the same CRC-32 technique is applied at the receiver.

$$x^{32}T(x) + x^{k+32}L(x) \tag{6}$$

At receiver side, the same operation is applied to obtain original message. This can be clarified as follows

Leave 32 bits for the CRC and complement first 32 bits

$$x^{32}T(x) + x^{k+32}L(x) \tag{7}$$

then we intent to get remainder of modulo2 division by the Generator polynomial.

$$CRC_{rx} = rem\left\{[x^{32}T(x) + x^{k+32}L(x)]/G(x)\right\} \tag{8}$$

which can be rewritten as

$$CRC_{rx} = rem\left(x^{32}[T(x) + x^{k}L(x)]/G(x)\right) \tag{9}$$

Using 5 we will have:

$$CRC_{rx} = rem\left(x^{32}[x^{32}M(x) + x^{k}L(x) + L(x) + R(x)]/G(x)\right) \tag{10}$$

Taking the divisor inside

$$CRC_{rx} = rem\left(x^{32}[\{[x^{32}M(x) + x^{k}L(x)]/G(x)\} + \{L(x)/G(x)\} + \{R(x)/G(x)\}]\right) \tag{11}$$

Using equation 2 we can now rewrite as

$$CRC_{rx} = rem\left(x^{32}[\{Q(x) + R(x)/G(x)\} + \{L(x)/G(x)\} + \{R(x)/G(x)\}]\right) \tag{12}$$

Knowing that $rem(A + B + C) = rem(A) + rem(B) + rem(C)$

$$CRC_{rx} = rem\left(x^{32}\{Q(x) + R(x)/G(x)\}\right) + rem\left(x^{32}\{L(x)/G(x)\}\right) + rem\left(x^{32}\{R(x)/G(x)\}\right) \tag{13}$$

It should be clear that remainder of first term is R(x). Last term should result also in R(x) as remainder as R(x) is a polynomial of degree 31 while G(x) is a polynomial of degree 32.

$$CRC_{rx} = x^{32}R(x) + rem\left(x^{32}\{L(x)/G(x)\}\right) + x^{32}R(x) \tag{14}$$

Since we are using modulo2 arithmetic, the term R(x) will cancel each other

$$CRC_{rx} = rem\left(x^{32}\{L(x)/G(x)\}\right) \tag{15}$$

This means that the CRC operation at the receiver side depends only on L(x) and G(x). The resulted remainder is the remainder of the following modulo2 division

$$x^{32}L(x)/G(x) \tag{16}$$

which is a constant value known as residual value. This explains why we always have a constant value that shall be obtained at receiver side no matter what are the content of the message.

To compute the CRC of a message, the generator polynomial G(x) is chosen as:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \quad (17)$$

This polynomial can be represented in Hex as 104C11DB7. The constant residual obtained from that polynomial is C704DD7B. The following MATLAB code can be used to make modulo2 division and to obtain this residual value

```
clear

pkg load communications

dividend = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, ...
            1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, ...
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, ...
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
divisor =  [1,0,0,0,0,0,1,0,0,1,1,0,0,0,0,0, ...
            1,0,0,0,1,1,1,0,1,1,0,1,1,0,1,1,1];

len_end = length(dividend);
len_sor = length(divisor);

if (len_sor <= len_end)
   len_diff = len_end - len_sor;
   for app_idx = len_sor+1 : len_sor+len_diff
       divisor(:, app_idx) = 0;
   end
else
    printf("Division is not possible, dividend must be longer than divisor\n")
end

if (len_diff >= 0)
    for div_idx = 1:len_diff+1
        if (dividend(div_idx) == 1)
            dividend = xor(dividend, divisor);
            div_res(div_idx) = 1;
        else
            div_res(div_idx) = 0;
        end
        if (div_idx == len_diff+1)
            break;
        end
        divisor = shift(divisor, 1);
        divisor(:,1) = 0;
    end
end

quotient  = dec2hex(bi2de(div_res, 'left-msb'))
remainder = dec2hex(bi2de(dividend, 'left-msb'))
divisor = dec2hex(bi2de(divisor, 'left-msb'))
```

Reversed or Not Reversed: Ethernet packet contents are sent LSB first except for the CRC part which is sent MSB first. This means when calculating CRC of received packets we have to take into consideration that coming data is LSB first except for CRC, MSB is received first. Same point has to be considered also at transmitter side that makes us provide message to the CRC calculator as LSB first (reverse input message) and when transmitting the message, we reverse the resulted CRC before appending it to the message. Here, we need a little clarification.

- In order to reverse input message to the CRC calculator, we can either reverse the order of input message or use a reversed circuit of the CRC calculator. The CRC calculator is the logic that calculates the needed CRC. We can design this logic to consider LSB is coming first or MSB first. Therefore, there are two ways to calculate the CRC in a reverse manner: either to apply LSB-first message on a direct circuit or apply MSB-first message on a reversed circuit, but don't apply LSB-first message on a reversed circuit or MSB-first message on a direct circuit.
- The calculated CRC in a reverse manner, doesn't equal CRC in a direct manner. Even if we reversed any of them, they are not equivalent also. However, In case we calculated CRC in a reverse manner, we have to reverse the resulted CRC before appending it to the message. The reason behind that is, the receiver side is calculating the whole received packet (original message + reversed CRC). Therefore, the standard of Ethernet CRC is reversing the whole packet. When reversing the whole packet, we will get back the correct CRC again (reversed original message + CRC).
- One last point to mention here is the residual value of Ethernet CRC (C704DD7B) is obtained because we invert the resulted CRC before transmission as described in equation 3. If we didn't invert before transmission, we should get 0 when we calculate CRC at receiver side.

Reference: https://www.cl.cam.ac.uk/research/srg/bluebook/21/crc/node2.html