

CSE 123 : Computer Networks

Winter 2022 : Discussion-4

P1 TA : Shreyas Anantha Ramaprasad

CRC

- Steps to Calculate CRC:
 - Keep CRC as your last field of the structure and set it to 0
 - Convert frame to char array by calling `convert_frame_to_char`
 - Calculate the CRC remainder(pseudocode coming up)
 - Use `memcpy` to store the remainder in the crc bits
- Steps to verify CRC:
 - Compute CRC remainder
 - If remainder is not zero, drop the packet

Bitwise Operators

- $X \& Y$

- Eg: If $X = 10$, $Y = 6$ -----> $X \& Y = 2$

- $X =$ 00000000 00000000 00000000 00001010

- $Y =$ 00000000 00000000 00000000 00000110

- $X \& Y =$ 00000000 00000000 00000000 00000010

- $X | Y$

- Eg: If $X = 10$, $Y = 6$ -----> $X | Y = 14$

- $X =$ 00000000 00000000 00000000 00001010

- $Y =$ 00000000 00000000 00000000 00000110

- $X | Y =$ 00000000 00000000 00000000 00001110

Bitwise Operators

- $X \wedge Y$

- Eg: If $X = 10$, $Y = 6$ -----> $X \wedge Y = 12$

- $X =$ 00000000 00000000 00000000 00001010

- $Y =$ 00000000 00000000 00000000 00000110

- $X \wedge Y =$ 00000000 00000000 00000000 00001100

- $\sim X$

- Flip the bits

- Eg: if $X = 10$ -----> $\sim X = -11$ (if signed) or $\sim X = 4294967285$ (if unsigned)

- $X =$ 00000000 00000000 00000000 00001010

- $\sim X =$ 11111111 11111111 11111111 11110101

Bitwise Operators

- Left-shift : $X \ll Y$
 - Eg : $X = 10 \text{ -----} \rightarrow X \ll 2 = 40$
 - Shifts X by Y bits to the left
 - $X = 00000000\ 00000000\ 00000000\ 00001010$
 - $X \ll 2 = 00000000\ 00000000\ 00000000\ 00101000$
 - Equivalent to multiplying by 2^Y
- Right-shift : $X \gg Y$
 - Eg : $X = 10 \text{ -----} \rightarrow X \gg 2 = 2$
 - Shifts X by Y bits to the right
 - $X = 00000000\ 00000000\ 00000000\ 00001010$
 - $X \gg 2 = 00000000\ 00000000\ 00000000\ 00000010$
 - Equivalent to dividing by 2^Y (Integer Division)

CRC Computation

```
// Function returns the remainder from a CRC calculation on a char* array of length byte_len
char crc8(char* array, int array_len){
    // The most significant bit of the polynomial can be discarded in the computation, because:
    // (1) it is always 1
    // (2) it aligns with the next '1' of the dividend; the XOR result for this bit is always 0
    char poly = 0x07; //00000111
    char crc = array[0];
    int i, j;
    for(i = 1; i < array_len; i++){
        char next_byte = ith byte of array;
        for(j = 7; j >= 0; j--){ // Start at most significant bit of next byte and work our way down
            if(crc's most significant bit is 0){
                left shift crc by 1;
                crc = crc OR get_bit(next_byte, j); // get_bit(next_byte, j) returns the a bit in position j from next_byte
            } else{ // crc's most significant bit is 1
                left shift crc by 1;
                crc = crc OR get_bit(next_byte, j);
                crc = crc XOR poly;
            }
        }
    }
    return crc;
}
```

Miscellaneous

- Be aware of padding in structures. If you want to pack your structures, use :
 - `struct __attribute__((__packed__)) Frame`
 - or `#pragma pack(1)` -> This will apply to all structures that you define
- Max number of bytes corrupted in one frame is 32
- Bitwise operators (&, |, ~ and ^) have lower operator precedence than comparison operators == and !=
 - E.g. to see whether the most significant bit of a byte is 0
 - Wrong: `if (byte & 0x80==0)`
 - Correct: `if ((byte & 0x80)==0)`
- Ensure that you never have more than 8 outstanding frames
- CRC should be included in both message frames and acknowledgement frames as both can be corrupted