CS 118 Assignment 2

Charles Zhang

October 21, 2022

Problem 1

This is done with the assumption that we know which block is the last block!

1.1)

To fix this, we add a single `0` byte as padding, regardless of if the original frame ends with a `0` byte.

1.2)

If the input data contains all `0` bytes, then the coded data will be 1 byte longer than the original input data. This is because each `0` byte is encoded as a 1. The final `0` will be padded with another `0`, resulting in an extra byte in the encoding.

In general, one extra byte comes from the padding solution in 1.1. Extra bytes can also come from blocks encoded with a length of 255, as those blocks have to add the length byte without removing a `0` byte, Therefore, the encoded data will be one byte longer for each 255 frame, plus the extra byte of padding at the end of the last block.

1.3)

The definition of overhead for this problem is:

$$\frac{\text{Length of encoded data} - \text{Length of original data}}{\text{Length of original data}}$$

As mentioned above, extra bytes come from 255 frames and the padding at the end. Therefore, the overhead for packets is:

$$\frac{(L + \#\text{ of 255 frames} + 1) - L}{L} = \frac{\#\text{ of 255 frames} + 1}{L}$$

If $L >> 254$, the number of 255 frames is negligible, since each extra byte of padding is offset by 254 data bytes. Even in the worst case scenario, where every frame is a 255 frame, this extra padding comes at a 1:254 ratio, so the numerator will still be extremely small when compared to L, leading to approximately $\frac{1}{254}$ overhead.

If $L << 254$, there are no 255 frames. This leaves us with an overhead of $\frac{1}{L}$, which, in the worst possible case ($L = 1$), is 1.

1.4)

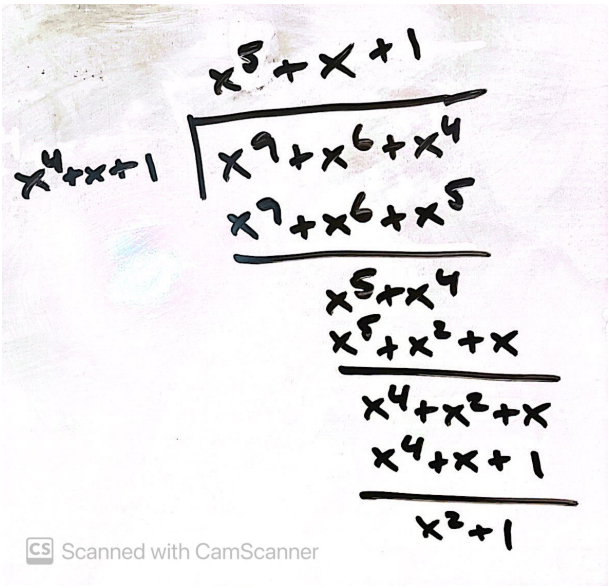
```
Let D be the decoded data
For each block in the encoded data:
    Let L be the length byte of the block
    If L is 1 and the current block is the last block:
        Break from loop
    Else if L < 255:
        For each byte of the next L bytes:
            Add current byte to D
            Go to next byte
        If the current block is not the last block:
            Add a 0 byte to D
    Else:
        For each byte of the next 254 bytes:
            Add current byte to D
            Go to next byte
    Move to the next byte, which is the next block
Return D
```

Problem 2

2.1)

$$100101 \rightarrow x^5 + x^2 + 1$$

Given the generator $x^4 + x + 1$, we know that $r = 5$, and therefore, the shifted message is $x^9 + x^6 + x^4$.



$$\boxed{0101}$$

2.2)

$$x^4 + x + 1 \rightarrow 10011$$

Knowing the message is `100101` and the CRC is `0101`, we know that the message + CRC is `1001010101`.

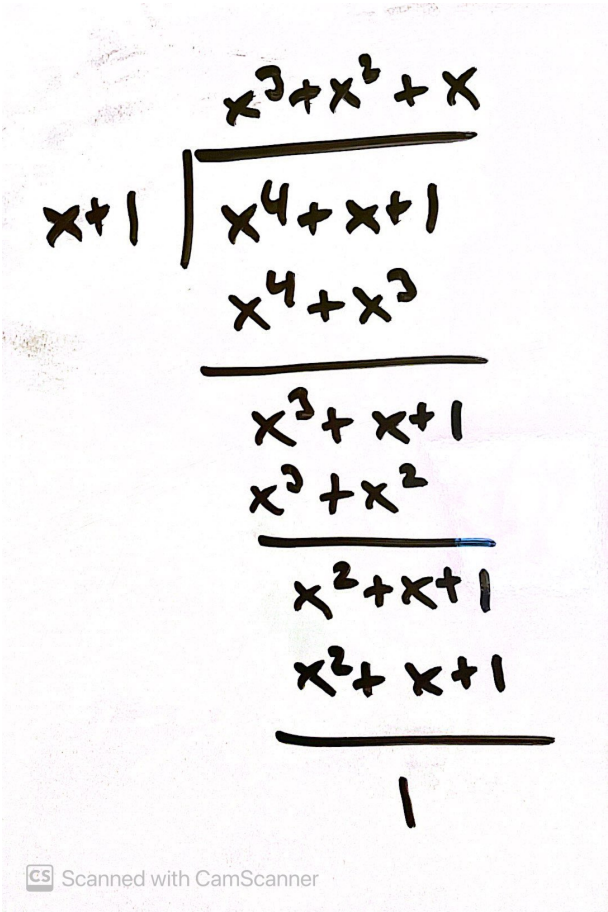
$$1001010101 + 10011 = \boxed{1001000110}$$

2.3)

This generator does detect all 1-bit errors, since it has more than 2 terms. A 1-bit error creates a single-term error polynomial, which we know for a fact is not divisible by any value with 2+ terms, such as our generator. As a result, the message containing the error would not be divisible by the generator, and the error would be detected.

2.4)

This generator doesn't detect all odd-bit errors because this guarantee relies on $x + 1$ being a factor of the generator, but $x + 1$ is not a factor of $x^4 + x + 1$, as seen below:



2.5)

As said in the problem, we want to find how many bursts of length 10 are multiples of the generator, meaning how many polynomials exist whose highest term is x^5 and whose lowest term is 1. In terms of binary, this means we need to find the number of possible bit strings of the form `1XXXX1`, where `X` means the bit can be either `0` or `1`. This problem reduces to finding the number of bit strings of length 4, which is $2^4 = 16$. Therefore, there are 16 undetected burst errors of length 10 for the generator $x^4 + x + 1$.

The results of multiplying the generator by the given polynomials are as follows:

$$(x^4 + x + 1)(x^5 + x^4 + x + 1) = \boxed{x^9 + x^8 + x^6 + x^5 + x^2 + 1}$$

$$(x^4 + x + 1)(x^5 + x + 1) = \boxed{x^9 + x^6 + x^4 + x^2 + 1}$$

$$(x^4 + x + 1)(x^5 + 1) = \boxed{x^9 + x^6 + x^5 + x^4 + x + 1}$$

2.6)

From 2.5, we know that, to generate a burst of length N with a degree 4 CRC, we need to have a multiplier of the form $x^{N-5} + ... + 1$. As a result, we know that the number of undetected bursts possible in a burst of length N is 2^{N-6}.

2.7)

Expanding the logic from 2.6 to a burst of length N and a CRC degree of k, we see that we need a multiplier of the form $x^{N-k-1} + ... + 1$ to generate an undetectable burst. Therefore, there are 2^{N-k-2} undetectable bursts for a given burst length and CRC degree. We also know that a burst of length N is of the form $x^{N-1} + ... + 1$, and that therefore, there are 2^{N-2} unique bursts of length N. Therefore, the odds of an undetectable burst happening are:

$$\frac{2^{N-k-2}}{2^{N-2}} = 2^{-k} = \frac{1}{2^k}$$

For CRC-32, $k = 32$, so the odds of an undetectable burst occurring are $\frac{1}{2^{32}} = \frac{1}{4294967296}$, which is an extremely low probability. This tells us that CRC-32 does a good job at moving towards the goal of quasi-reliability.

Problem 3

3.1)

The sender needs to send STATUS packets because the Nack sent back by the receiver is the only communication that the sender has from the receiver, other than error reports. As a result, if we don't send STATUS packets and the receiver never detects an error, the sender has no way to know if the receiver is actually receiving the messages being sent. It could be possible that the receiver has disconnected, and the sender is sending packets nowhere or that the final packet was never received.

3.2)

This property guarantees that there are two results of the sender sending a packet: the receiver sends back a Nack, indicating that a packet was dropped somewhere or the STATUS timer expires. In the first case, the sender will resend dropped packets until the second case takes place. In the second case, the sender will send a STATUS packet, forcing the receiver to send back a Nack, confirming that it has received all previous packets without error. As a result, sent packets are guaranteed to reach the receiver.

3.3)

To maintain this property, the timer must be started when the timer is currently stopped and the sender sends a packet. It must be stopped when the sender receives a Nack.

3.4)

First, D would be sent and dropped and the STATUS timer would start. Since no other data packets are sent, the sender doesn't know that D was lost until the STATUS timer expires, the sender sends a STATUS packet, and the receiver sends back a Nack, telling the sender that D never made it. Then, the sender can finally resend D, starting the STATUS timer once again. Since there are still no other data packets sent, we again have to wait for the STATUS timer to expire, the sender to send a STATUS packet, and the receiver to send back a Nack, informing the sender that D has made it. Therefore, the overall latency is 2 STATUS timer durations prior to sending the two STATUS packets, plus 2 round-trip times for the two STATUS/Nack exchanges.