Questions

* 1. Can we chance decoding algorithm? (e.g., expect extra O at the end). What’s wrong with no padding: 2XY indicates that there is no trailing zero
  2. Do we consider the frame boundary (O). For the in general part, should we give an estimate? E.g., generally the number of bits in a frame == number of data bits
  3. Packet vs. frame in this context? What is a packet of size L >> 254 (frames can only be up to length 254, so what is a packet? A group of frames??)? Why isn’t the coding scheme efficient for small packets? Is it because of the extra O’s between frames?

Nima Amir Dastmalchi (505320372)

**Problem 1**

* 1. The minimal amount of padding necessary is 1. We can pad the last block with one byte of 0 (O) at the end. For example, if the data is XYOZ, then we add a O to make it XYOZO. Now, we can encode the data with the same encoding rules. The first block is XYO and the second block is ZO. The first block encoded becomes 3XY and the second block encoded becomes 2Z. Since we always add an extra O at the end regardless of the data being transmitted, once data is decoded, we must remove the extra zero at the end of the last block.
  2. Consider input data of all O’s of length n bytes. In this case, the length of the coded data will be 2n + 1. The data consists of n blocks of 1 byte (which is a O byte). We add an extra O byte at the end (due to the answer in problem 1.1). Each block will be encoded as 1 (because the length is 1). However, we need to add the framing character between each block. So, n bytes are added between the n+1 frames. Thus, there are bytes. For example:

Input: OOOOOOOO (8 bytes)

Output: 1O1O1O1O1O1O1O1O1 (17 bytes)

In general, the coded data will be almost the same length as the input data. If the data contains enough zeroes to create a coded blocks of length less than 254 bytes, then the output will be the exact same length. This is because any data of format D0 (where D consists of non-zero bytes) is encoded as XD, where x is a single byte indicating the size of D. Note that size of D0 is equal to that of XD. On the other hand, blocks of size 254 bytes will be encoded as blocks of 255 bytes (1 byte for the size (255) and 254 bytes for the data). Thus, some blocks may contain an extra byte of data.

**Problem 2**

* 1. Shift M by 4 bits (G is 5 bits, so shift by 4 bits)

Calculate the remainder of shifted M divided by G

Remainder 0101?

M + CRC =1001010101 = x^9 + x^6 + x^4 + x^2 + 1 (should be divisible by G)

* 1. Add G to M + CRC to get a new M + CRC divisible by G (undetected error)
  2. This generator detects all 1-bit errors. To have an undetected error, we need to add the error polynomial to the message, where k is another non-zero polynomial. Because , will have at least 3 terms. A 1-bit error requires the addition of a 1-term polynomial to the message. Therefore, no 1-bit error will be undetected by this generator.
  3. If the generator has a factor of , then the CRC can catch all odd bit errors. Because divided by has a non-zero remainder of , is not a factor of . Thus, this generator does not detect all odd bit errors.
  4. Any polynomial represented by a bit string between and can cause an undetected burst of length 10. This range contains possible polynomials. **Question: Are we talking about burst of length 10?**
  5. Let be the length of an arbitrary error burst. Thus, .

Let be the degree of the generator . Thus, .

Since must be of form , there are possible bursts of length .

For an error polynomial to be undetected, it must be a multiple of , That is,

for some multiple polynomial . By replacing the functions by their known forms, we get:

Thus, must be of form . There are possible polynomials for , so there are possible undetected error polynomials.

Now, we compute the ratio of undetected errors to all possible bursts:

Thus, probability of not detecting an error of burst of arbitrary length is , where is the degree of the generator.

This tells us that a very small fraction of errors will be undetected by CRC-32 (. Thus, the CRC error detection scheme enables a “quasi-reliable” scheme that very rarely has undetected errors.

**Problem 3**

* The STATUS is needed because if the last packet in the sequence is lost, then the receiver will not be able to detect an error. If the receiver crashes (no longer works), no NACK will be sent and the sender thinks that the receiver is receiving the messages. Also, if the NACK gets lost, then the sender will not retransmit. Finally, all packets could be lost and no NACK will be sent by the receiver.
  + Problem with NACK: what if receiver gets packets out of order and falsely detects a packet drop and sends a NACK.
  + Fix: Add STATUS
* Because the STATUS message acts as a periodic acknowledgement to data.
* The timer can be stopped when a STATUS message is sent, sent data is acknowledged, and no more data is sent by the sender.