Nima Amir Dastmalchi (505320372)

**Problem 1**

* 1. The minimal amount of padding necessary is 1. We can pad the frame 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. Even if the frame ends with O, we still add a O. 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 n + 1 bytes. Each block consists of one O, which will be encoded as 1. We also add an extra O byte as padding to the end due to problem 1.1. So, for a frame of length n bytes, the length of coded data is n + 1 (if we ignore the frame delimiters).

Input: OOOOOOOO (8 bytes)

Output: 1 1 1 1 1 1 1 1 1 (9 bytes)

In general, the coded data can be at most 2 bytes longer than the original input data. There are 2 types of blocks:

1. Less than 254 bytes: In this case, the coded block will be the same length. Consider the block XO, where X is less than 254 bytes. XO will be encoded as nX, where n is the length of X. Note that both the data and coded data are of length n + 1.
2. Exactly 254 bytes: In this case, the coded block will have one extra byte. Consider the block X, where X is 254 bytes with no O’s. Then, X will be encoded as 255X, where 255 is an extra byte.

Also, recall that each data frame is padded with an extra byte. So, in the worst case, the data contains no O’s, so all blocks are 254 bytes (except for the last block because we always pad with a O). If this data is of length n, then there are floor(n / 254) extra length bytes for each block and 1 extra zero byte at the end. So, the coded data can be at most floor(n / 254) + 1 bytes longer, where n is the length of the input data.

* 1. Very large packet of length L >> 254 bytes.

In the worst case, the packet will contain no O bytes, so every 254 bytes of data will be encoded as one block with an extra byte indicating the size of the block. So, an extra are added. There will also be an extra byte of padding, so we must add 1 (problem 1.1). The overhead will be:

For , the overhead approaches .

Very small packet of length L << 254 bytes.

The worst-case is a small packet of size 1. In this case, we are transmitting one byte B. Due to problem 1.1, we add an extra byte of padding. So, the original data is of length 1 while the coded data is of length 2. This results in the following overhead:

Thus, the worst-case small packet size has a 100% overhead.

* 1. Pseudocode for the receiver decoding algorithm:

Read the next frame into the variable Frame

Let DecodedData be an empty byte stream

While True:

Let CurrentBlock be an empty byte stream

Let Size be the first byte of Frame

Append the next Size – 1 bytes to CurrentBlock

If Size < 255:

Add O to CurrentBlock

Append CurrentBlock to DecodedData

Remove Size bytes from Frame

Remove an O from the end of DecodedData

Output DecodedData

**Problem 2**

* 1. The CRC for the message is 0101 = . See work below.

Diagram, text, letter

Description automatically generated

* 1. The resulting message is 1001000110 = . See work below

Diagram, letter

Description automatically generated

* 1. 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 2 terms (the largest and the smallest terms cannot cancel out). 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.
  2. No, the generator does not detect all odd bit errors. The CRC can catch all odd bit errors iff the generator has a factor of . Because divided by has a non-zero remainder of , is not a factor of . Thus, this generator does not detect all odd bit errors. Also, problem 2.4 is a counterexample. Observe that we added the error polynomial to the message to get an undetected error, which has an odd number of bits.
  3. The three error polynomials are shown below.

Text, letter

Description automatically generated

* 1. An undetected burst error of length 10 on a generator of degree 4 must satisfy by below equation:

To satisfy the above equation, must be of form . There are possible polynomials of this form, each resulting in a unique error polynomial.

* 1. 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**

* STATUS message act as a periodic acknowledgement to data. The STATUS packet is needed for the following reasons:
  + If the last packet in the sequence is lost, then the receiver will not be able to detect a lost packet because it would think that the second to last packet was actually the last packet. So, the sender should ensure that even the last data packet was received by sending a STATUS packet.
  + If the receiver crashes (no longer works), no NACK will be sent. Without STATUS packets, the sender would think that all packets are being successfully transmitted to the receiver because the receiver is not sending any NACKs. In reality, this is because the receiver is down.
  + If the NACK gets lost, then the sender will not know that a previous packet was dropped. So, a STATUS packet is necessary.
  + All packets could be lost and no NACK will be sent by the receiver.
* While there is unacknowledged data, the sender will send a STATUS message. The receiver will continue sending NACKS to the sender to acknowledge its received data. If there are packets that are dropped (or any of the reasons from previous problem have taken place), then the sender will continue sending STATUS messages until all data is acknowledged. It will also retransmit messages if necessary. Because we assume that the link delivers most packets without errors, the messages will eventually be sent to and acknowledged by the receiver.
* The timer can be stopped when all data that was sent by the sender is acknowledged via a STATUS message by the receiver. As soon as new data comes in, the STATUS timer should restart to allow for retransmission of new data if those packets are dropped.
* The worst-case time between the initial sending of D and the sender figuring out that D was received is 2 \* ST + RT, where ST is the length of STATUS timer and RT is the length of a round trip.

As shown in the image on the next page, once the first STATUS is sent, the receiver sends a NACK indicating that D was not received. Then, sender will send another D, which the receiver will receive. Once the second STATUS is sent, the receiver will acknowledge that it is not expecting the next data packet (D+1). Thus, this will span 2 status timers and one round trip.

Diagram

Description automatically generated