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?

XYOOLO

Blocks: 3XY, 1, 2L

Packet/frame: 3XY12L

3.4 Assume D is the first one

3 Link could be totally dead if there is no NACK.. That’s why we need STATUS (timer based on threshold)

3.4 We need something to send before setting off the timer (passed to sender from routing layer – when we start the timer).

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 n + 1. 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: 11111111 (9 bytes)

In general, the coded data will be almost the same length as the input data. For the case of blocks less than 254 bytes, the coded block will be the same length. For example, consider an arbitrary block XO, where X is less than 254 bytes. Then, we encode XO as nX, where n is the length of X. Note that both the input and code are length n + 1. For the case of blocks equal to 254 bytes, the coded block will be 255 bytes. For example, consider an arbitrary block X, where X is 254 bytes and does not contain any O’s. Then, X will be encoded as 255X, where 255 is an extra byte.

For both cases (254 and <254 bytes), an extra byte of padding (according to problem 1.1) will be added.

* 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. There will also be an extra byte of padding (problem 1.1). The overhead will be:

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. This results in the following overhead:

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

* 1. Pseudocode for the receiver decoding algorithm:

Let Frame be the next frame without O delimiters

Let DecodedData be an empty stream

While True:

Let Size be the first byte of Frame

Read the next Size – 1 bytes into CurrentBlock

If Size < 255:

Add O to CurrentBlock

Add CurrentBlock to the end of DecodedData

Remove an O from the end of DecodedData

Output DecodedData

**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. 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. 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.
* We send STATUS packets as long as there remains unacknowledged data. So, as long as packets are being dropped, a STATUS message will eventually be sent, which informs the sender of the packets that have been successfully transmitted. The sender can then retransmit the packets that were possibly dropped. This process continues until all packets sent by the sender are guaranteed to have been received by the receiver in the correct order.
* 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.
* **At what point should the timer be stopped? Is there any case a package does not get there but a NACK is not sent. Keep sending STATUS while not hearing NACK once STATUS timer expires. When to start sending STATUS packets and when to stop? Start the timer when data is sent. Keep sending until NACK is sent back from receiver.**
  + **Once sender gets D from upper layer, start sending STATUS’s. We get E, F, G, etc in the queue. At what point should we stop sending STATUS packets? Once all data is sent (G?)**
  + **Stop the timer once we get the NACK for the next data that we have not gotten yet.**
* The timer can be stopped when a STATUS message is sent, sent data is acknowledged, and no more data is sent by the sender.
  + Problem with NACK: what if receiver gets packets out of order and falsely detects a packet drop and sends a NACK.
  + Fix: Add STATUS