Homework 2

Due on Oct 21

Directions: Start early! We will pick 1 to grade out of 3 but you **must** hand in Problems 1,2, and 3 for completion credit. Problems 4 and 5 are only for exam practice; do not hand in.

1. **Framing,** 20 points: Bit stuffing can be expensive to implement if framing is done in software because it can cause the stuffed data not to be aligned on 8 bit boundaries. Thus many software framing techniques used on dialup lines do byte stuffing. Byte stuffing will guarantee that the stuffed data length is a multiple of 8. But Byte stuffing has a worst case overhead of 50% as you have seen in the slides. Alyssa P. Hacker has come up with what she calls the most efficient byte stuffing technique ever, much more efficient than even HLDC bit stuffing.

Suppose the framing character is the zero byte (which we denote by O). We want to code the data to remove all occurrences of O. Alyssa's initial idea is as follows (it needs one more modification to work). We divide the data into blocks of up to 254 bytes such that:

EITHER each block is less than 254 bytes and ends with a O

OR the block is 254 bytes in length and does not contain a O

We then encode each block by placing the length of the block at the start, followed by the bytes in the block, but without the trailing zero byte O, if any. Notice that we can get rid of the zero byte O because the length of each block tells us where to place the zero byte during decoding. The exception is if the length is 254 in which case the block DOES NOT contain a trailing O and we use a length field of 255 to encode this case. (We cannot use 254 because that is reserved for a block of 253 bytes followed by a trailing O). This last exception allows us to encode long sequences that do not contain a zero byte. As an example if the input is XYOOLO, where X, Y, L are arbitrary bytes, the first block is XYO, the second block is O, the third block is LO. The first block encoded becomes 3 X Y, the second block encoded becomes 1, the third block encoded becomes 2 L. This is not the final encoding because this does not quite work as you will see in **1.1**

1.1) The problem does not specify how one encodes the data if the last block is less than 254 bytes in length and does not end in a O. To fix this bug, one could try to pad the last block. What is the minimal amount of padding necessary to make sure we can use Alyssa's rules and yet decode correctly. (5 points)

1.2) If the input data contains all O's (all zero bytes), what is the length of the coded data? In general, how much longer can the coded data be than the original input data? (5 points)

1.3) What is the worst case overhead for a very large packet of length L > > 254? What is the worst case overhead of your scheme for a very small packet L < < 254? Overhead is the worst case number of non-data bytes divided by the number of data bytes. (5 points)

1.4) Write pseudocode for the receiver decoding algorithm incorporating whatever padding scheme you used in Part 1.1 above. (5 points)

-

2**, CRCs Polynomial View**: Consider the CRC generator $x^4 + x + 1$.

- Consider the message 100101. Calculate the CRC for this message using the polynomial generator above (3 points)
- The simplest way to create an undetected error is to add an error polynomial equal to the generator to the message + CRC. What is the resulting message that is received? (2 points)
- Does this generator detect all 1-bit errors? Why? (Hint: review arguments in Notes) (1 points)
- Does this generator detect all odd bit errors (1 point, see Notes)?
- How many undetected burst errors (starting at Offset 0) can there be of burst length 10? To help you do this, note that a burst error of length 10 is a polynomial that starts with x^9 and ends with 1 with optional terms for the remaining powers between 8 and 1. For an undetected error, the resulting burst error must be divisible by the generator $x^4 + x + 1$. Instead of seeing which bursts are divisible by the generator, compute how many bursts of length 10 are multiples of the generator. For example, if we multiply $x^4 + x + 1$ with $x^5 + 1$ we get a length 10 burst. Given that the only way to get x^9 as the highest power and 1 as the lowest power is to multiply by polynomials whose highest power is x^5 and lowest power is 1, first write down three such polynomials. Specifically multiply the generator by $x^5 + x^4 + x + 1$, by $x^5 + x + 1$, and $x^5 + 1$ and simplify the three results after cancelling any terms. (3 points)
- Based on the last examples, how many such polynomials are there in general (that when multiplied by the generator cause an undetected burst, for any degree 4 CRC (5 points)
- Based on the last two answers, can you argue briefly why the probability of **not** detecting a burst of arbitrary length is $1/2^k$ where k is the degree of the CRC (4 in this case). Hint: consider the ratio of the number of undetected bursts to the total number of possible bursts of any given length. What does this tell you about the power of CRC-32? (5 points)

-

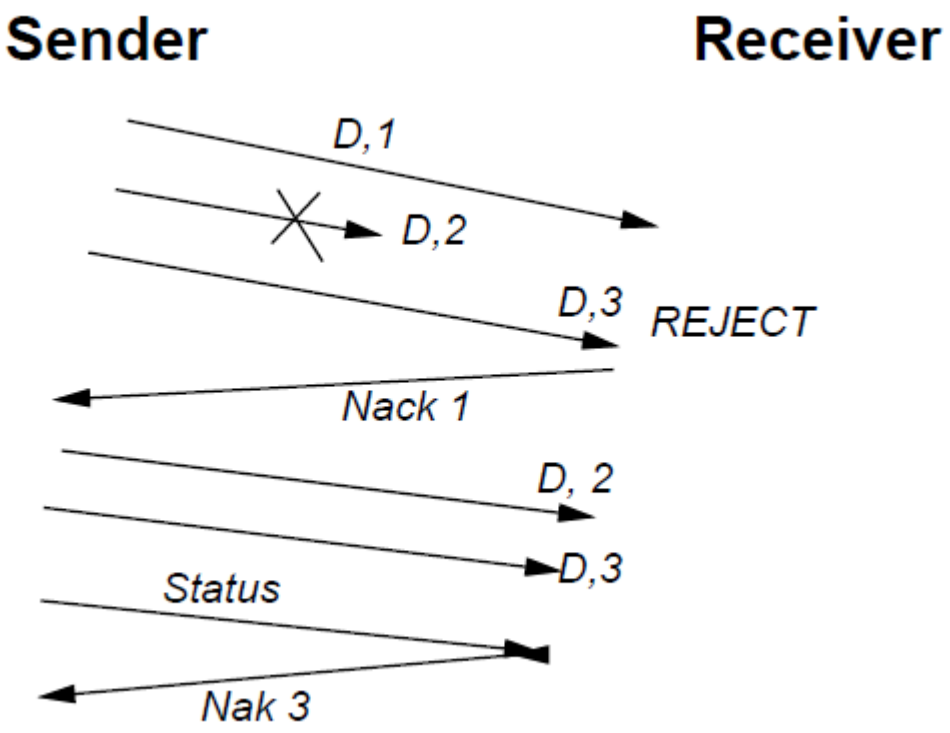
3. **Error Recovery:** Peter Protocol has been consulting with an Internet Service Provider and finds that they use an unusual error recovery protocol shown in the Figure below (next page). The protocol is very similar to Go-back-N with numbered data packets; the key difference is that the sender does not normally send ACKs. The receiver sends a message called a Nack (negative acknowledgement) only if it detects an error in the received sequence or if it receives a so-called STATUS packet. In the figure below, the sender sends off the first three data packets. The second one is lost; thus when the receiver gets the third packet it detects an error and sends a Nack which contains the highest number the receiver has received in sequence. When Nack 1 gets to the sender, the sender retransmits data packets 2 and 3. Periodically, based on a timer, the sender transmits a STATUS packet. The receiver always replies to a STATUS packet using a Nack

   - Why is the STATUS packet needed? What can go wrong if the sender does not send STATUS packets? (3 points)
   - A STATUS packet is sent when a STATUS timer expires. The sender maintains the following property: ``While there remains unacknowledged data, the STATUS timer is running." Why does this property guarantee that any data packet given to the sender will evntually reach the receiver (as long as the link delivers most packets without errors)? (2

points)

- Under what conditions must the timer be stopped and started so as to maintain the property (5 points)
- Consider sending a single data packet D that is lost. After that no STATUS, Nack, or retransmissions of D get lost and no other data packets are sent. What is the worst-case latency before the receiver receives D and the sender knows the receiver has got D. In other words, what is the worst case time between the initial sending of D and the sender figuring out that D was received. Assume the STATUS timer is greater than twice the Round-trip time. (10 points). In the figure below, Nak 3 should be read as Nack 3

Figure for Problem 3



1.

4. **(Exam Practice Only)** Data Link Protocols on Synchronous Links: So far in all our Data Link protocols we have assumed the links to be asynchronous in that the delay of a frame or ack could be arbitrary. Now we consider the case that the time taken for a message or ack is 0.5 time units. Further senders send frames only at integer times like 0,1,2. When a receiver gets an error-free frame (sent at time n) at time n + 0.5, the receiver sends an ack back that arrives (if successful) just before time n + 1. Suppose we use the standard alternating bit protocol except that the sender also waits to send at integer times.

   - Does the sender need to number the data frames? If your answer is yes give a counterexample to show what goes wrong when it does not. (10 points)
   - Does the receiver need to number the ack frames? If your answer is yes give a counterexample to show what goes wrong when it does not. (10 points)
   - Describe a simple protocol for the sender to initialize the receiver state after a crash? (5 points)

- **5. (Exam Practice only, do not hand in) HDLC Framing**: The HDLC protocol uses a flag 01111110 at the start and end of frames. In order to prevent data bits from being confused with flags, the sender stuffs a zero after every 5 consecutive ones in the data. We want to understand further that not all flags work but perhaps some others do work so the HDLC flag is not the only possible one.

   - Consider the flag 11111111 and a similar stuffing rule to HDLC (stuff a 0 after 5 consecutive 1's). Show a counterexample to show this does not work. (5 points)
   - Consider the flag 11111111. Find a stuffing rule that works and argue that it is correct. What is the worst case efficiency of this rule? (Recall HDLC had a worst case efficiency of 1 in 5 bits, or 20%) (5 points)
   - Hugh Hopeful has invented another new flag for HDLC (Hopeful Data Link Control) protocol. He uses the flag 00111100. In order to prevent data bits from being confused with flags, the sender stuffs a one after receiving 001111. Does this work? Justify your answer with a short proof or counterexample. (8 points)
   - To reduce the overhead, Hugh tries to stuff a 1 after receiving 0011110. Will this work? Justify your answer with a short proof or counterexample. (7 points)

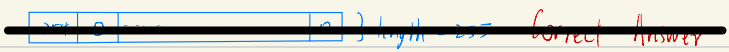
1.1

Always add a 0 byte to the end of data regardless of it is end with 0 or not.

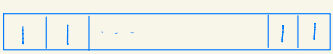
For example, L0 would become L00, L would become L0

1.2

One plus original length of data. Since for all data we pad a zero at the end of it. In Alyssa's algorithm, we remove a tail zero byte and add a leading length byte, which does not change the length.



~~OR~~



1.3

The wrost case for $L \ll 254$ is 1, Take L as data, for instance, the ecoding of L is 1L (after zero padding). so $\frac{\text{non-data bytes}}{\text{data bytes}} = \frac{1}{1} = 1$.

The best case for $L \gg 254$ is $\frac{1}{254}$. Since we add at most 1 byte for every 254 bytes in data frame.

1.4

```
\\F is the input frame
decoder(byte[] F){
        int L = 0 //current position in F
        byte[] result //output frame
        while(L < F.length){
                //S is the current size of the block
                int S = F[L]
                if(S <= 254)
                        result.add(F[L+1], …, F[L+S-1], 0)
                else if(S == 255)
                        result.add(F[L+1], …, F[L+S-1])
                L += S
        }
        //remove the 0 padding if it exists
        if(result[result.length-1] != 0)
                RaiseException(No Pad at End)
        else
                result.remove(result.length-1);
        return result
}
```

2.1

```
          100110
      ------------
100011 )100101000000
        100011
        ------
         001100
         000000
         ------
          011000
          000000
          ------
           110000
           100011
           ------
            100110
            100011
            ------
             001010
             000000
             ------
CRC ------>  01010
```

2.2

```
  10010101010  ------> message + CRC
+   100011     ------> generator
------------
  10010001001  ------> recieved message
```

2.3

Yes, because it has 3 terms. Any multiple of a polynomial with at least 2 terms will also have at least 2 terms

2.4

No, because it's not a mutiple of x+1.

```
      11110
    ---------
11 )100011
    11
    --
     10
     11
     --
      10
      11
      --
       11
       11
       --
       01
```

2.5

The number of polynomials whose highest power of x^5 and lowest power of x^0 is $2^4 = 16$.

Thus, there are 16 undetected brust errors of length 10.

2.6

The number of polynomials whose highest power of x^4 and lowest power of x^0 is the Combinations of x^4, x^3, x^2, x^1 which equals to $2^4 = 16$. So there are 16 such polynomials.

2.7

The number of n-burst errors is equal to 2^(n-2). Since the number of n-burst errors is equal to the number of combinations of the n terms generator.
The number of undetected n-burst errors is equal to 2^(n-k-2). This is because the number of undetected n-burst errors is equal to the number of combinations of the n terms polynomial created by polynomial and generator multiplication whose highest degree is n and lowest degree is 1. If the generator is k degree, then all such polynomial (not the n terms polynomial) must have highest degree of n-k and lowest degree of 1. Which equals to 2^(n-k-2).

So, the probability is

$$\frac{\text{undetected}}{\text{total}} = \frac{2^{n-k-2}}{2^{n-2}} = \frac{1}{2^k}$$

For CRC-32, it has $\frac{1}{2^{32}}$ chance to fail, which is really small.

3.1

The STATUS is needed because there are cases where the sender doesn't know that error has occurred. For example, if all packets are lost, the sender will receive no NACK, or if the last packet is lost, the sender will not receive NACK as well. Thus the sender will never find out the issue.

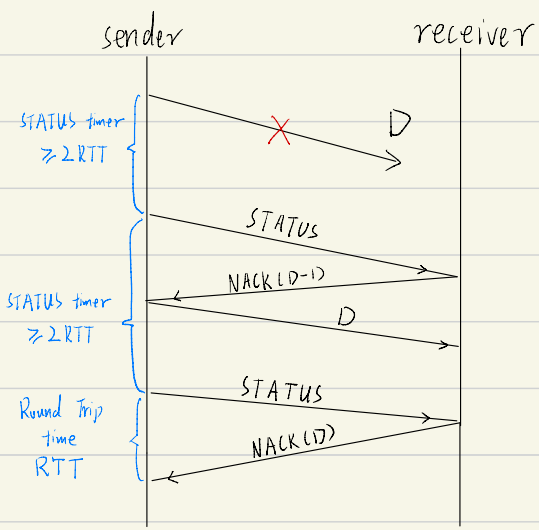
3.2

Since the sender keeps sending STATUS packets as long as there are unacknowledged data packets, the sender guarantees that all such data packets will be transmitted until acknowledged.

3.3

The timer should start if there are packets in sender's sending queue, otherwise, stop.

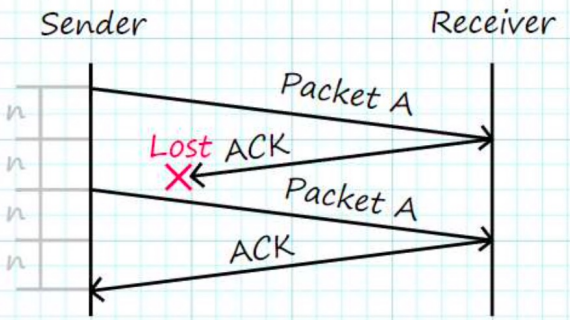
3.4



So the worst-case latency should ≥ 5RTT

4.1

Yes, the sender must number the frames. Without this, lost ACKS could lead to duplicate packets. The following is a counterexample



In this situation, the sender only wanted to send a single Packet A, but due to a lost ACK, the sender assumed it was not received so it retransmits A and the receiver accepts it since it cannot tell the difference between identical data and retransmitted data

4.2

No, the sender does not need to number the ACK frames. Given the synchronous nature of the data link, any ACK received at just before time n+1 must be an ACK for the packet sent at n. Since there is no ambiguity as to which packet is being ACK'ed, the numbers are useless

4.3

After a crash, wait time n before sending a RESET packet. This works because all other packets will have expired after a period of n, so there will be no possible confusion with other packets sent. This is okay to wait because the link was already sending at most once every time n, so the loss in throughput is minimal

5.1

The data could merge with the end flag to form a new endflag

Counter Example: 1111 Error

Outputted Frame: 11111111<u>11111111</u>1111

5.2

Stuff a 0 after every 1. This will clearly stop the flag from showing up in the data. This will stop any issues with the endflag because it is impossible for a 1 to interact with the endflag without there being a 0 in between

Worst case overhead: 1/1 or 100%

5.3

This rule will work. This rule breaks the flag without initializing the flag again. It also stops any data that will potentially interact with the endflag since 001111 is needed in order to do that.

5.4

This rule will not work. Even though it stops the flag in the data, it does not succeed in stopping errors at the endflag.

Counter Example: 001111 Error

Outpptted Frame: 00111100<u>00111100</u>111100