UCLA CSC 118 Computer Networks

Midterm Exam

Elan Markowitz [redacted]

Directions: Write your name on the exam and on every page you submit. Write something for every question. Students who do not write something for everything lose out over students who write down wild guesses. You will get some points if you attempt a solution but nothing for a blank sheet of paper. Write something down, even wild guesses. Problems take long to read but can be answered concisely.

Question	Maximum	Score
1	40	35
2	20	20
3	20	20
4	20	20
Total		95

Nice!

1,000,000

1, Over [redacted] important answer. 4 points apiece.

- a) **Spam and Layering:** Some ISP's use anti-spam devices at the network that check for the content of every packet to see if it is likely spam (i.e., junk mail) and drop those packets. The ISP claims that looking at email contents (not headers) is not a layer violation. More specifically, they claim that if the routing or transport layer changes, they would not have to reprogram their antispam device. Explain what is wrong with this argument. Did not do this — 2
They are breaking layering by looking at the data itself. Even though they don't look at other layers' headers, they are still violating layering.
- b) **Media Impairments:** Cat 5 cable has a Nyquist Bandwidth of 100 Mhz. In order to transmit at 100 Mbps over Cat 5 cable why can you not use Manchester encoding?
Nyquist bandwidth = nyquist limit, thus you can send 100×10^6 signals/sec. However, each bit in manchester encoding requires 2 signals (high-to-low or low-to-high) Thus you can only achieve 50 Mbps of data transmission
- c) **Clock Recovery:** What happens to an eye pattern when the bandwidth becomes too small? These eyes start to "close" (ISI). The sampling margin gets smaller and smaller or disappears completely as interpsibol interference increases.
- d) **Preambles and Transitions:** Why is 01011 01011 01011 a good preamble for 4-5 encoding whereas 0101010101 is a good preamble for Manchester encoding (Hint: in 4-5 encoding besides bit synchronization what other kind of synchronization is needed).
Nice!
for manchester because it ensures transitions only happen at the half time units.
01010101 is good 4-5 encoding sends 4 data bits + 1 bit for encoding, this guarantees transitions. 01011 01011 01011 ensures that the receiver is also in sync regarding each bit set.
- e) **Media:** Because radio ranges are approximate, it is hard to ensure that the 802.11 networks of neighboring homes do not overlap. What property of 802.11 should the home owners exploit to minimize interference? There are 3 channels that are entirely non-overlapping. If each owner is on a different channel then will not be interference.
- f) **Framing:** If 11111111 is the flag used for a bit stuffing approach to framing, what is the worst-case stuffing overhead? Explain. You need to stuff a Zero after every one. 50% stuff bits
otherwise, if the data ends in "1" that bit will be read as part of the 11111111 Stop flag. worst case, you are sending only 1's and need to send
- g) **Latency vs Throughput:** In the old days of batch processing, a number of jobs were run at the same time on an IBM 360. Explain in terms of latency and throughput what batch processing is trading off. Be precise (say latency and throughput of what, and whether each measure increases or decreases). It increases throughput of the network but increases the latency for the client.
Job sent
Job processed result returned
vs.
Batch sent
Batch processed
Batch returned
client
IBM360
- h) **Restarting Data Link Protocols:** Suppose after a crash, the sender sends a RESTART message using a pseudo-random number generator that uses the sender ID as a seed. What could go wrong? The problem is that the pseudo-random number generator will output the same number for a given Sender ID. Therefore there is no differentiation between multiple crashes from the same sender.
client waits for whole batch but less time is wasted transmitting Jobs
- i) **Multiplexing:** When a voice call is made, 64kbps of data bandwidth is reserved across any digital line in the path of the call. Why is strict multiplexing reasonable for a voice call instead of statistical multiplexing? You can't have calls dropping anytime there's a big collision (want reliability). Strict multiplexing ensures a data frame will not have a collision. Statistical multiplexing does not have this reliability. — 3
- j) **End-to-end argument:** In the Internet, vendors sometimes require that their packets be encrypted (i.e., coded such that the packet cannot be read by intruders listening on the wire). Why is it better to have the packet encrypted by the source and decrypted at the destination (i.e., end-to-end) instead of being encrypted and decrypted by each router in the path (i.e., hop-by-hop)?

end-to-end is safer because it has been shown that something can go wrong in hop-to-hop. Nice!
Additionally a packet could be routed through an infected/malicious router which would have access to the unencrypted data. Would be very easy for a state actor to ensure most domestic data gets routed through them, so they can snoop your data.

Elan Markowitz
901485758

2, Recovering Bits, Nyquist Limit: In HW 1, we showed that sending a bit every 1 usec was possible without intersymbol interference. even though the output of a 1 lasts a long time (say 4 usec). This is because the output had the right "shape" so that earlier bits are always at 0 Volts whenever later bits are sampled. However, the shape of the output is crucial. In the figure below, suppose the output is as shown. Note that the output makes a swing down to -2.1 V at 3 usec and is back to 0 at 4 usec. With such an output intersymbol will take place when sending bits every 1 usec. Assume the bits to be sent are 111. Write on back of paper if needed.

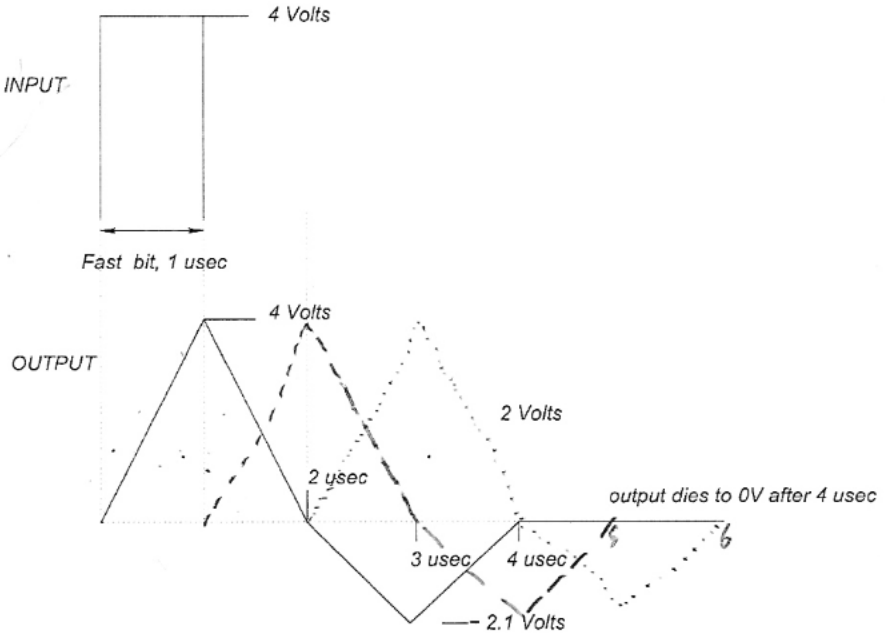


Figure 1:

- (4 points) **a)** Suppose that the 3 bits are sent at times 0, 1, and 2 usec respectively. Draw a picture of the waves corresponding to the first, second, and third bits on the picture above. The first is drawn, draw the second as a dashed wave, and the third as a dotted wave.
- (4 points) **b)** Suppose that the 3 bits are sampled at times 1, 2, and 3 usec respectively. What is the sampled value of the output in volts at these three sampled times? 4V, 4V, 1.9V
- (2 points) **c)** Suppose the receiver declares the received bit to be a 1, when the voltage is more than 2V, and to be a 0 otherwise. What 3 bits will the receiver physical layer output? 110
- (4 points) **d)** Describe (don't draw) an example in which InterSymbol Interference occurs even when sending once every 2 usec. and sampling at 1,3,5,.... If you send 11 then you will get interference on the second sample and will sample 1.9V
- (6 points) **e)** With this new shape, what is the fastest rate that the sender can send and guarantee *zero* intersymbol interference. 2 points for the answer, and 4 points for a simple argument that if you send at this rate, no matter what set of bits are sent in the past, there will be *zero* interference when the current bit is sampled.

You can only send a signal every 3 msec, and sample at 1,4,7,... Each signal takes 3 msec to go from 4V to rest at 0V. Thus if you send 3 msec after the previous bit, when your signal hits 4V (1 msec after sending) the previous signals will all have died out at 0V.

[redacted]

32. **CRCs** Peter Protocol only knows how to compute 8 bit CRCs efficiently (CRC-8) and cannot figure out how to implement CRC-16 efficiently. No worries, he thinks, I can just use *two* CRC-8's in the same frame as shown in the figure. He uses a standard CRC (shown as CRC-1). He then adds a second CRC, shown as CRC-2, that applies to the original data plus CRC-1.

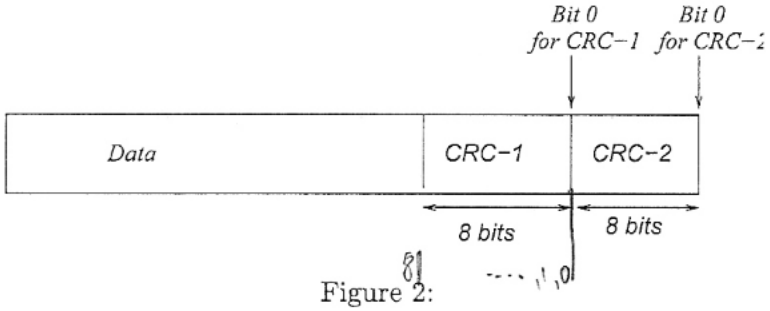


Figure 2:

- (5 points) Suppose there is an error in positions i, j, . ., k in the Data. Assume that positions i, j, . . k are numbered starting with the LSB of CRC-1 (in other words, the first bit of the data will be Bit 8.) Write down the error polynomial with respect to CRC-1

  error polynomial is $x^i + x^j + x^k$

- (5 points) Now write down the error polynomial with respect to CRC-2. Note that the numbering of bits will change with respect to CRC-2 because with respect to CRC-2, bit 0 is the LSB of CRC-2. So the same error will result in a *different* error polynomial for CRC-2

  $x^{i+8} + x^{j+8} + x^{k+8}$

- (6 points) Using your understanding of polynomial division, show that if CRC-1 fails to detect an error in the data, then CRC-2 will also not detect the error.

  $x^{i+8} + x^{j+8} + x^{k+8} = x^8(x^i + x^j + x^k)$ Anything that evenly divides $x^i + x^j + x^k$ will also divide $x^8(x^i + x^j + x^k)$ Therefore any error not caught by CRC-1 will also be missed by CRC-2

- (4 points) Based on your last answer, how does Peter Protocol's idea of using two CRC-8's compare with using a single CRC-8? How does it compare to using CRC-16?

  It is identical to CRC-8 with regard to error detection but 8 bits are wasted by having CRC-1. Compared to CRC-16, it has the same number of bits used for error checking but has far greater reliability. CRC-16 can detect errors of double the distance compared to CRC-8 (And Peter Protocol's protocol)

[redacted]

4, Data Link Protocols on Synchronous Links revisited: Recall in HW 2 one of your "extra" problems assume 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 $n+1$. Instead of using the standard alternating bit protocol, assume that sender and receiver use no numbers in messages or acks. However, sender messages carry a single bit a, which which is set to the value of a state variable *ack_received* at the sender. The state variable *ack_received* at the sender is set to true at time n if and only if an ack was received in time period $[n-1, n]$.

- **a), 5 points:** Consider a message sent successfully by the sender at time n. The receiver gets it at time $n+0.5$ and sends an ack that is lost. Thus *ack_received* at time $n+1$ is false. Based on this write a single If-then-else clause of pseudo-code to specify what the sender should do (e.g., retransmit, send a new message) at time $n+1$. Explain why briefly.
- **b), 5 points:** Based on the same example, write down 1 line of pseudo-code to specify what the receiver does (e.g., deliver or drop message) when it receives (a, m) where a is the value of the *ack_received* flag when the sender sent (a, m). Justify your answer. Explain why briefly.
- **c) 5 points:** Now consider the case when sender and receiver start the protocol, and the sender sends a single message that is not lost. Based on your code in b) what should the value of *ack_received* be initialized to at the sender. Explain why briefly.
- **d) 5 points:** Now consider the case when the sender and receiver start with the initialization as in c). The first message sent at time 0 is lost. Any message sent at time 1 gets through. Based on your code in a) and c) what goes wrong? What do you conclude about this protocol?

a) +5

```
if (ack_received):
    send next message with ack_received=True;
else:
    retransmit with ack_received=False;
```

n → n+0.5 (✗)

If the ack is not received then we should retransmit because the message may have never reached the receiver.

b) +5

```
if (not a):
    drop message; send ack;
else:
    deliver message; send ack;
```

if a is false we think the data is a repeat message, and drop it. However we need to send an Ack because the sender did not receive the last one. This prevents deadlock.

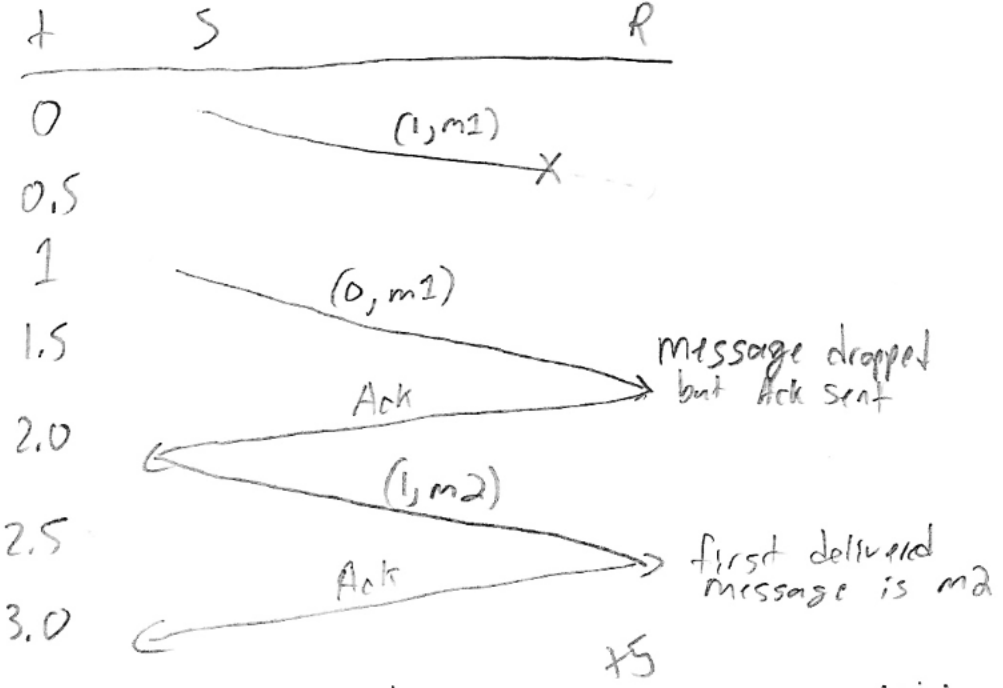
c) Based on my code, the initialization does not matter because all received messages get acked, and as long as the receiver has not received a message from the sender in the last 1.5 time, the ~~n~~ will be delivered. We will choose to initialize to true no ack has been missed at start.

c) Sender should initialize ack_received to true otherwise the receiver will drop the first message.

+5

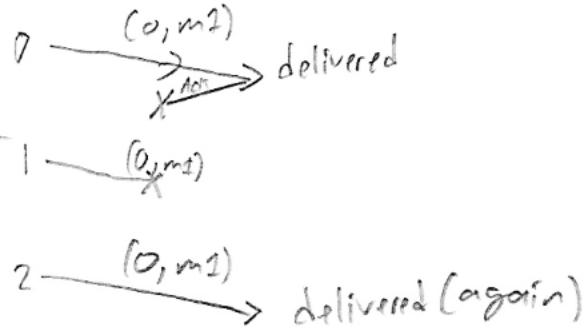
next page

d)



In this scenario you will drop any message that was interrupted on the way to the receiver. This is no more unreliable than not having an ack_received bit at all.

The protocol could be greatly improved by having the receiver remember if it received a message in the last 1 time period. Then it would only drop messages when a==0 and a message was just received. This solves all single message failures. However, it would still have failures modes. Ex: ack lost, repeat message lost,



This would be a big improvement because it can't fail if only a single message is lost.