CS164 — Fall 1999 Midterm

No aids allowed.

Problem 1. Write down the technical term that best fits each of the following definitions.

1. A device that allows an analog signal to be carried over a digital channel.

   I was expecting a "codec" (or coder/decoder), but I also accepted things like a "network adapter card". Half credit for "modem" which does the opposite function of digital signals over analog phone lines.

2. That method of sending information over a channel in which there is a one to one correspondence between the physical state of the channel and the information being sent at that time.

   "base band", or "dc signalling" or "NRZ"

3. A technique for increasing the throughput of a system without decreasing the individual processing times, in which different stages of service for different customers are carried out in parallel.

   "pipelining", although I also accepted "multiplexing" and even "cut-through switching"

4. A sequence of K bits that begins and ends with a bit error, does not contain any long runs of correct bits within the sequence, and falls between two long runs of correct bits.

   burst error

5. A source of delay in lightly loaded wide area packet switched networks, which explains why the same bits of data appear to travel faster if they are sent in short packets than if they are sent in long packets, even though the same sequence of channels running at the same transmission rate are used in both cases.

   store-and-forward delay

6. What a transmitting station in a local area network is trying to do by comparing the signal leaving its own transmitter with the signal arriving at its own receiver.

   I was expecting "collision detection" but I also accepted answers related to equalization or startup

7. A method in which the sender appends to the end of each packet or frame of data the remainder after division by a binary polynomial, which usually allows the receiver to tell whether or not his copy of the data is identical to the sender's copy.

   "CRC", or "cyclic redundancy check" or "polynomial code"

8. A electrical cable whose cross-section consists of several concentric rings of different material.

   "coaxial cable". A surprising number of people said "optical fiber" for which I gave half credit because it said "electrical"

9. An acknowledgment strategy for simplifying buffer management at a receiver based on accepting data from the sender only in the correct sequence.

   I was expecting "go-back-N" or "stop-and-wait", as opposed to "selective repeat". I gave partial credit for "sliding window" and "cumulative acks"

10. A digital transmission method in which there is guaranteed to be a state transition at the middle of each bit time.

    Manchester encoding. Since it was the signalling scheme used in the original 10 Mbps Ethernet, I also accepted "Ethernet"

Problem 2. In Ethernet, the framing is done using the following set of rules. First, two consecutive frames must be separated by an idle period that lasts for at least 96 bit times. Next, each frame is separated from the idle period in front of it by a 64-bit preamble, consisting of an alternating sequence of "0"s and "1"s. And, finally, the frame itself consists of a minimum of 64 bytes and a maximum of 1518 bytes including a 14-byte header in front, the actual payload and a 4-byte CRC error check at the end.

a Find the maximum possible number of frames per second that can be transmitted by one host, assuming the network runs at a data rate of 10^7 bits/sec. [HINT: Assume that the host tries to send a continuous stream of very short frames.]

*First find the number of bit-times required to send the minimum length frame. (There is no point to look at longer frames, because they occupy the channel longer and hence you can send fewer per second.) This is 64 bits (for preamble), plus 64 bytes * 8 bits/byte (for the frame), plus 96 bits (for the idle period afterwards), for a total of 672 bit times per packet. Now to convert this to packets/second, we divide the raw bit rate of* 10^7 *bits/sec by 672 bits/packet to get 14,881 packets/sec.*

Note how I kept the units on all the numbers and checked to make sure the dimensions of the final answer, i.e., (bits/sec)/(bits/packet), came out to the right value. Lots of people came up with nonsensical answers — like dividing 10^7 *by the *product* instead of the *sum* of the three components, which gives a maximum of 3 packets/sec — that could have been avoided by checking units.*

b Find the maximum number of payload bits per second that can be sent by a single host over Ethernet as a function of the frame size F, $64bytes <= F <= 1518bytes$, assuming the network runs at a data rate of 10^7 bits/sec.

Assume the frame size is F *bytes,* $64bytes <= F <= 1518bytes$*. As above, the number of such frames per second we can send is given by* $(10^7 bits/sec)/(64 + 8F + 96bits/frame)$*. Since the payload bits per frame is* $8(F - 18)$ *bits/frame, after we remove the 14 byte header, 4 byte CRC, and convert it into bits, we have to multiply these two to give us payload bits sent per second as* $10^7(F - 18)/(F + 20)$ *bits/sec.*

Problem 3. Recall that in 4B/5B encoding, Alice converts each group of four data bits into a 5-bit code before it is transmitted over the link, while Bob converts each incoming 5-bit code back into the corresponding group of four data bits as it arrives.

a Briefly explain the reasons for transmitting these extra bits over the link.

The extra bits in 4B/5B are there to allow us to chose code words that have a minimum number of signal level transitions on the wire to maintain clock synchronization in the data stream. A secondary reason is to provide some non-data code words for control of the data link, so that bit stuffing is not required.

b Suppose Bob receives a 5-bit sequence that *cannot* be translated into a group of four data bits. Can we conclude that a transmission error must have occured? If not, what else might be going on?

It could be a non-data code word.

Problem 4. What is the difference between a simple parity code and a two dimensional parity code?

A simple parity code adds one bit to the end of the string so that the total number of ones in the string is always even (or odd). A two-dimensional parity code arranges the data into a rectangular table and adds a simple parity bit to the end of each row and the bottom of each column. Simple parity can't detect burst errors reliably, but 2D parity can detect all burst errors where the burst length is less than the row size of the table.

Show that any pattern of one, two, or three bit-errors will be always be detected by a two dimensional parity code.

A parity code will detect an error if and only if the number of bit errors in the row (or column) it is trying to protect is an odd number. Therefore, a single bit error is always detected, since the row and column it belongs to have an odd number of errors (i.e., one).

With two bit errors, they could be in the same column (hence different rows), the same row (hence different columns) or different rows and columns. In all three cases, there is at least one row or column with an odd number of errors.

With three bit errors, they could all be in a single column (which contains three errors, an odd number), they could be distributed across two columns (in which case one of those columns has a single error, an odd number), or distributed across three columns (in which case all three of those columns have an odd number of errors).

Give an example show that some patterns of four bit-errors cannot be detected by a two dimensional parity code.

A "square" pattern of four errors will not be detected because all of the affected rows and columns contain exactly two errors, which is an even number. That is for rows i and j and columns x and y, a pattern of four errors at coordinates (i, x), (i, y), (j, x) and (j, y) will not be detected.

Problem 5. In HDLC, the distinguished bit string "01111110" is used as a separator between frames, and any sequence of 7 or more consecutive 1's is used to indicate that an unrecoverable error condition has occurred and the data link protocol must be restarted.

a Briefly explain bit stuffing, and how it would be used in this case.

Bit stuffing is an algorithm for inserting bits into a data stream to prevent the appearance of a reserved low-level control sequence on the channel. Stuff bits get added by a FSM encoder at the transmittter, and deleted by a matching FSM decoder at the receiver; neither one uses any look-ahead into the data stream. Bit stuffing provides "data transparency", so that the user can send any string of data through the channel without interacting with the low level controller.

With this particular flag, the encoder inserts a "0" after seeing 5 consecutive "1" bits. The receiver keeps a running count of the number of consecutive "1" bits; if the count ever reaches 5, then the next bit is thrown away as a stuff bit if it is a "0", or interpretted as completing a real control flag if it is a "1".

b Give a simple example to show that stuff bits may be needed inside the CRC field, and not just within the user-defined payload field.

Since the CRC is just the remainder after doing long division, it can have any value. Thus, if the data field happens to end with "...0111" and the CRC field happens to start with "1110...", then a "0" stuff bit will need to be inserted after the second "1" of the CRC.

c Suppose your boss asks you to develop an "improved" version of HDLC in which the frame-separator bit string is changed to "01110110" to break up the long sequence of consecutive "1" bits in the original frame separator. How does this change affect the number of stuff bits required?

This change increases the number of stuff bits required, because the new flag contains the start of another flag pattern inside, beginning at bit 5. To see this, remember that the bit stuffing algorithm must be modified to work with this new flag pattern, so that it inserts a "0" stuff bit after seeing the first six bits of the new flag in the data stream, i.e., "011101". Now suppose the user decides to transmit a long data stream that contains a repeating pattern of a single "0" alternating with three consecutive "1"s, like this:

*"01110111011101110...". First, the bit stuffing algorithm insert a "0" stuff bit after seeing "011101", changing the data stream into "011101***0***11011101110...", to tell the receiver that a real control flag does not start at the beginning of the string. However, that does not tell the receiver anything about whether a real control flag could start a little later in the data stream. For example, what if the user decided to send only four bits of data "0111" instead of the long string, and then the transmitting FSM decided to send a real flag. The incoming data stream up to and including the first stuff bit would be identical in both cases. Thus the bit stuffing algorithm would need to insert additional stuff bits to show that no flag is starting at position 5, position 9, position 13, etc.*

d Give an example to show that with the "improved" frame separator described in part (c) you might need to insert a stuff bit inside the frame separator bit string itself! Can this situation happen with the original frame separator?

Suppose the data stream happened to end with "...0111". Then after transmitting the first two bits of a real control flag, i.e., "01..." the FSM encoder sees that last six bits transmitted over the channel are "011101" which triggers the insertion of a stuff bit.

The same thing cannot happen with the original flag of "01111110" because the FSM encoder is counting the number of consecutive "1"s since the last time it transmitted a "0". Thus, no matter how many "1"s there were at the end of the data stream, the number of consecutive "1" gets reset to zero by the "0" at the start of the original flag.