Note: In all written assignments, please show as much of your work as you can. Even if you get a wrong answer, you can get partial credit if you show your work. If you make a mistake, it will also help the grader show you where you made a mistake. Your submitted homework should be printed out (i.e., please don’t hand in hand-written answers, unless you need to hand-annotate the printed text, or draw a figure). See the class web page for more information about handing in homework assignments.

Problem 1: Sleuthing. [1 point] Suppose that your department has a local DNS server, but does not have a web cache, for all computers in the department. You are an ordinary user (i.e., not a network/system administrator) but you’re in CS453 so you know how to use Wireshark. How can you determine if an external Web site was likely accessed from a computer in your department in the past few seconds? Explain. Answer: I’d measure the RTT from when my computer sent out the DNS request to when the reply was received. If that RTT is small, then the response must have come from the local DNS server via its local DNS cache, i.e., without having to consult the root, TLD, or remote authoritative name servers.

Problem 2: How many port numbers are in use? [2 points] Consider incoming TCP and UDP segments arriving at a server, and suppose that (e.g., using Wireshark), we see that 150 different destination port numbers being used. The server acts only as a server (in the socket sense, i.e., it does not initiate communication with any other computers as a client; it only responds to incoming segments)

a) Describe a scenario (what server applications are running using which protocols) in which more than 150 sockets are in use at the server. Answer: Suppose there are 149 UDP sockets being used with 149 clients; each UDP socket has a unique local port number. A web server with say 1000 ongoing client connections will have 1000 TCP sockets for those 1000 client connections. However, these 1000 sockets, as well as the welcoming/accepting socket will all have port number 80. This we have 1149 sockets, but only 150 local port numbers being used.

b) Describe a scenario (what server applications are running using which protocols) in which exactly 150 sockets are in use at the server. Answer: Suppose there are 150 UDP sockets being used; each has a unique port number.

c) Is it possible that less than 150 sockets are in use at the server? Explain. Answer: this can’t happen since in this case, one socket would have to have two or more associated local port numbers.

Problem 3: UDP and TCP checksum [1 point]. (a) Suppose you have the following two bytes 00110101 and 01101001.
a) What is the ones complement of these two bytes? Answer: The ones complement of a binary number is obtained by changing all of the 1’s to 0’s and vice versa. Thus, the ones complement of 00110101 and 01101001 is 11001010 and 10010110.

b) What is the ones complement sum of these two bytes? Answer:

```
00110101
01101001
--------
10011110
```

And taking the ones complement of this sum gives 0110001.

c) For the bytes in part (a), give an example where one bit is flipped in each of the two bytes and yet the one’s complement sum doesn’t change. Answer: If we flip the third rightmost bit of the numbers 00110101 and 01101001, we get 00110001 and 01101101 which has the sum:

```
00110001
01101101
--------
10011110
```

Problem 4: Two senders, an alternating receiver. [4 points]. Consider a scenario in which Host A and Host B must deliver messages to Host C. Hosts A and C are connected by a channel; hosts B and C are connected by another channel (independent of the channel connecting A and C) with the same properties. The transport layer at Host C should alternate in delivering messages from A and B to the layer above (that is, it should first deliver the data from a packet from A, then the data from a packet from B, and so on). The receiver at C is also resource-constrained and cannot buffer received but not-delivered messages.

A channel that can corrupt but not lose messages. Now assume that the channel connecting A and C, and B and C, can corrupt messages but will not lose messages, i.e., that any message that is sent will come out the other side, but possibly corrupted. Messages can be corrupted in either direction on the channels! Design a minimalist protocol for senders A and B (which will be quite similar), and receiver C that performs the alternating delivery at C as described above. Your protocol should be minimalist in that it should only use the mechanisms needed to ensure correct operation.

a) In words, what messages will your protocol use, and what is contained in a message? In words, what will A do and what will B do? Answer: In my solution, C is going to request a message from A, and then from B, and so on, rather than having A and B directly send messages to C without being asked (in which case C will have to just drop the unwanted messages, since it can’t buffer). C will have to use sequence numbers - since the channel can corrupt messages, C may have to retransmit requests
that were corrupted or re-request data items that were corrupted. Also, I am going to add a timer here (although it is not strictly necessary) to simplify my A and B so that they can ignore corrupted messages and just wait for C to timeout and transmit. So in my design A and B are simple and identical and only need to know whether a duplicate or a new data message is being requested.

b) Draw three timing diagram of the sender and receiver exchanging messages. Show what happens in the case of (i) no message corruption, show a message being delivered from A, then a message from B, and then a message from A. (ii) a message being corrupted on the A-to-C channel, and (iii) a message being corrupted on the C-to-A channel.

c) Specify your protocol using finite state machines for A, B, and C. Here are my finite state machines:

Note: There are multiple protocols that one can use to solve this problem. One is a somewhat straightforward extension of the rdt protocol we did in class. A second one I’ve thought of is more efficient than more straightforward protocol. Extra credit will be given for a particularly efficient solution in terms of the number of messages sent/received.

**A channel that can corrupt and lose messages.** Now assume that the channel connecting A and C, and B and C, can corrupt and lose messages. Design a minimalist protocol for senders
A and B (which will be quite similar), and receiver C that performs the alternating delivery at C as described above. Answer: My solution to c) above works here, since I already added a time in, which will allow recovery from lost messages (in my solution c above, A and B treated a corrupted message as a lost message, and simply ignored it – the same as if it were lost).

d) Specify your protocol using finite state machines for A, B, and C.

Problem 5: Pipelined Protocol Quickies [2 points]

a) Consider the cross-country example in Figure 3.17. How big would the window size have to be for the channel utilization to be greater than 80 percent? You may assume that the size of a packet is 1500 bytes, including both header and data. Answer: here we have an RTT of .03 seconds (30 ms) and a Gigabit (1,000,000,000 bps) transmission link and a packet length of 1500 bytes or 12,000 bits. The time to send one packet is thus 12,000/1,000,000,000 = 12 usec. Using the formula:

\[ U = \frac{N \times (L/R)}{(RTT + L/R)} \]

We have:

\[ .8 = \frac{N \times .000012}{.000012 + .03} \]

or \( N = 2000 \) packets. As a check, it takes 2000*.000012 = .024 seconds to send 2000 packets, which is .8 of the RTT of .030 seconds.

b) For the Go-Back-N protocol, give an example (i.e., a timeline showing the sender and receiver and the messages they exchange, e.g., as in notes page 3-49 or 3-53) showing a Go-Back-N sender receiving an ACK for a packet that currently falls outside its window.

c) For the Selective Repeat Protocol, give an example (i.e., a timeline showing the sender and receiver and the messages they exchange, e.g., as in notes page 3-49 or 3-53) showing a SR receiver
receiving an ACK that is within its window, but for which it can not advance its sending window forward.