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. DNS Basics. Consider the DNS iterative query, as shown on notes page 3-10 (in module 3 notes). Suppose the round trip time (RTT) between the computer cis.poly.edu (and any other computer in poly.edu) and its local name server, dns.poly.edu, is 10 milliseconds, and that the RTT from dns.poly.edu to any other Internet site is 100 milliseconds. The local DNS server cache at dns.poly.edu, and that the caches of all other DNS servers are initially empty.

   a) How long is the time between when cis.poly.edu first makes sends a DNS query for the IP address of gaia.cs.umass.edu, and the time that it gets a reply to that DNS query? Explain how you arrived at your answer. Answer: The request to the local DNS server and the reply from the local DNS server take a total RTT of 10 ms. Each of the three queries from the local DNS server to the root server, TLD server and Authoritative name server each take 100 ms RTT, for a total of 310 ms.

   b) Now suppose a second computer in poly.edu send a DNS query to its local name server, dns.poly.edu, and again asks for the IP address of gaia.cs.umass.edu. How long is it until this query is satisfied? Explain your answer. Answer: since the local DNS server will have cached the IP address for gaia.cs.umass.edu, the second request goes to the local DNS server, which immediately returns a reply to the second computer for a RTT delay of 10 ms.

Problem 2. Caching. We’ve seen caching occur at a local DNS server, in a user’s web browser, and in an HTTP web cache in an institutional network. Answer the following questions for each of these three types of caches: browser cache, web cache, local DNS server cache (so overall you’ll have 6 answers to the two questions below):

   a) Is the information returned (from a browser, web or DNS cache, if cached there) guaranteed to be the same up-to-date information as what is stored at the remote origin?
server where this information is maintained? Answer: web browser: yes. Web cache: yes. DNS local server cache: no.

b) If your answer to a question n (a) is YES, then how does the cache (browser cache, web cache, or local DNS server cache) determine that the content is up-to-date? If your answer to a question is NO, then how will that locally cached copy ever be synchronized with (i.e., the same as, at least for some amount of time) as the up-to-date version at the origin server? Answer: The web browser and web cache will always check with the origin server to make sure the cached web page is up-to-date. The IF-MODIFIED-SINCE field is used to tell the server the date of cached copy. If the cached copy is up to date the server will respond but not include the content (since the cache has the most up-to-date version). If the server has a more recent copy of the content, it will return that content to the browser or web cache. The local DNS server cache maintains a time-to-live value for the cached DNS record. When that timer expires, the content will be removed from the cache and the next request for that DNS record will cause the local DNS server to get a fresh copy.

Problem 3. Reliable data transfer. This problem is a variation on the reliable data transfer protocol that we designed and studied in class, and will take a bit of thought. You will be designing a series of protocols, each a bit more complex, as each will have to deal with additional impairments (packet corruption, packet loss) on the channel. This problem is a variation of what we did in class, so make sure you understand that (in particular notes pages 8-29 in the module4 notes). Here we go ….

Suppose we have two computers A and B. B has a large supply of ordered pieces of data, D1, D2, D3, … and A is going to request that these pieces of data be sent from B to A one at a time. Specifically an application at A will make a request to your transport protocol at A to get the next piece of data from B. Your transport protocol at A will then interact with your transport protocol at B, and your transport protocol eventually deliver the requested piece of data (just once – no duplicates and no missing pieces of data) up to the application at A.

Perfect channel. Let first assume that the channel is perfect, i.e., that it won’t corrupt or lose any messages. You must design the protocol at A and the protocol at B for this channel, to allow to request and receive data items from B and deliver them to the application at A, one at at time, following a request by the application at A to get the next data time. You should only use the protocol mechanisms (sequence numbers, timers, checksums) that are needed for this scenario. [Hint: you don’t need these for a perfect channel]
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: A will send a message to B requesting the next data item. B will put the data item into a packet and send to A. This protocol is simple, since there are no errors.

b) Draw a timing diagram of the sender and receiver exchanging messages, e.g., as in page 18 of the module4 notes.

c) Now specify your protocol using a finite state machine. Remember that there is a finite state machine for A and another for B (recall notes page 16 in Module4 notes). Here are partially-complete finite state machine representations to get you started:
A channel that can corrupt but not lose messages. Now assume that the channel 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 both the A-to-B and B-to-A directions on the channel! You must design the protocol at A and the protocol at B for this channel, to allow to request and receive data items from B and deliver them to the application at A, one-at-a-time, following a request by the application at A to get the next data time. You should only use the protocol mechanisms (sequence numbers, timers, checksums) that are needed for this scenario.

\[d\) 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? \textit{Answer: here, we will need to add a checksum field since an A-to-B or B-to-A message can be corrupted. As with our rdt protocol, we’ll need to use a sequence number to indicate the number of the data item being requested so that the B will know which item A is requesting – a copy of something previously sent, or a new data item. Note that we do need a sequence number on the item being returned by B to A if the A-to-B request is corrupted and B returns the last item sent, A won’t know whether its request was corrupted or not, and so won’t know what numbered item B is returning.}

e) Draw three timing diagram of the sender and receiver exchanging messages, e.g., as in page 18 of the module4 notes. Show what happens in the case of (i) no message corruption, (ii) a message being corrupted on the A-to-B channel, and (iii) a message being corrupted on the B-to-A channel.
(c) error from B-to-A

f) Now specify your protocol using finite state machines for A and B.
A channel that can corrupt AND lose messages. This is the trickiest. Now assume that the channel can corrupt messages and possibly lose messages. Messages can be corrupted or lost in either the A-to-B or B-to-A directions on the channel! You must design the protocol at A and the protocol at B for this channel, to allow to request and receive data items from B and deliver them to the application at A, one-at-a-time, following a request by the application at A to get the next data time.

g) 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 addition to the solution above for d) – f) using checksums and sequence numbers, we’ll add a timer to A so that if a message (either A-to-B or from B-to-A is lost) that A will timeout and resend.

h) Draw three timing diagram of the sender and receiver exchanging messages, e.g., as in page 18 of the module4 notes. Show what happens in the case of (i) no message loss (don’t worry about corrupted messages), (ii) a message being lost on the A-to-B channel (don’t worry about corrupted messages), and (iii) a message being lost on the B-to-A channel (don’t worry about corrupted messages). Answer the case of no loss is the same as e) above.

i) Now specify your protocol using finite state machines for A and B. Answer: Note that the change from parts d) – f) are minimal – just handling the timeout event. Note that the FSM for this looks pretty close to our answer above.
Problem 4. CDN and Netflix quickies. Answer each of the questions below briefly – a couple of sentences each will suffice.

a) What is the purpose of the manifest file that is sent to the Netflix client? *Answer: the manifest file contains the URLs for different pieces of the video that have been encoded at a given encoding rate. For a given segment of time in the video, there will be several different encodings.*

b) Suppose the path from the video server sending video data to a Netflix client becomes congested. Might the quality of the video (as determined by the video’s encoding rate, say 256Kbps, 512 Kbps, or 1 Mbps) change? Explain briefly. *Answer: yes, if the path from the server from which the video is being retrieved becomes congested, the client may then request subsequent chunks of video from that server at a lower encoding rate. We saw this happening in Module 3, slide 3-32*
c) Is it possible that a Netflix client will change the video server from which it is receiving a video in the middle of a video? Explain briefly. Answer: yes, if the path from the server from which the video is being retrieved becomes so congested that even the lowest encoding rate of video can’t be sent to the client without loss, the client may then request subsequent chunks of video from a different server. We saw this happening in Module 3, slide 3-32.