CS 5480/6480: Computer Networks – Spring 2012
Homework 2
Due by 9 AM MT on February 22nd 2012

Important:

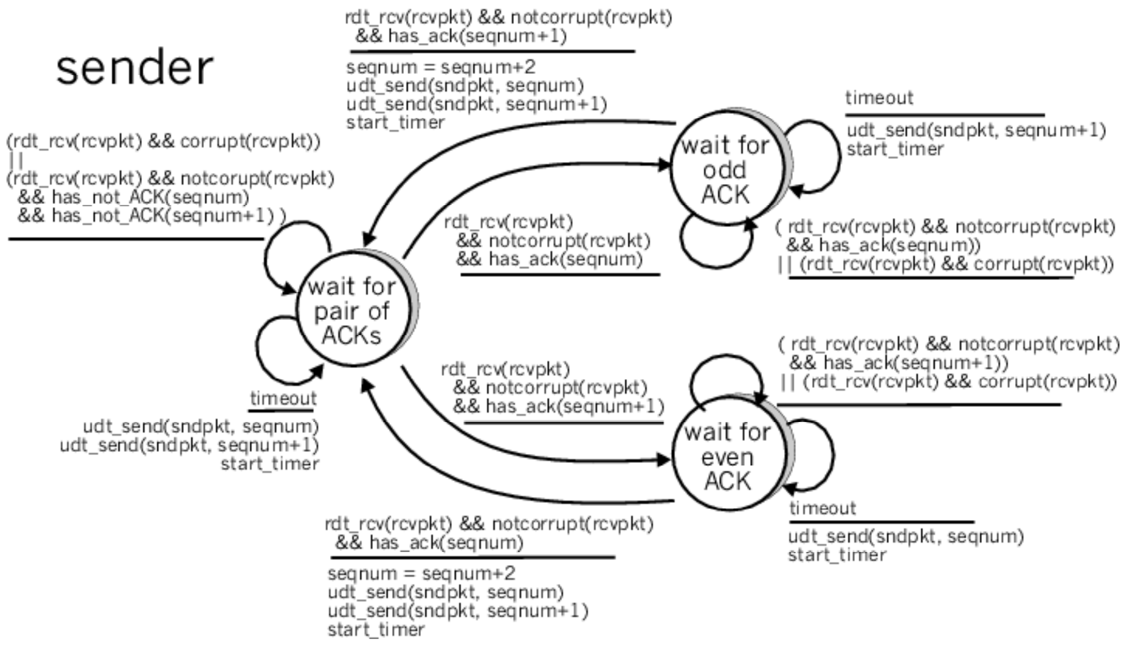
- **No cheating will be tolerated.**
- **No extensions will be granted**

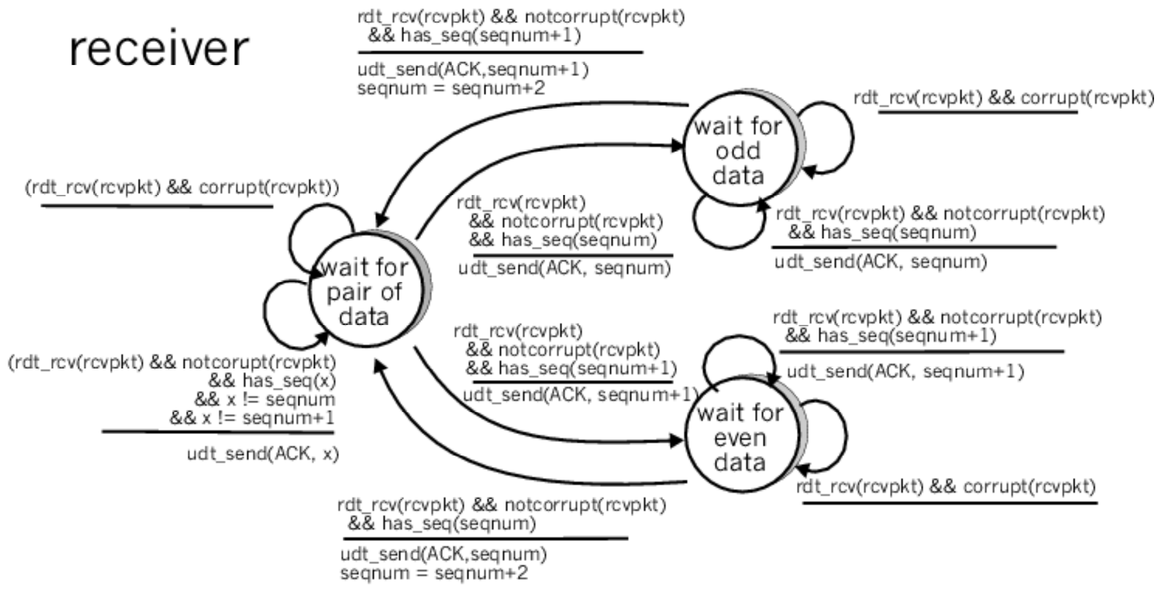
Total points for cs5480: 31
Total points for cs6480: 41

Question 1 *(Reliable Data Transfer Protocol Design) 6 points**:** In the generic SR protocol, the sender transmits a message as soon as it is available (if it is in the window) without waiting for an acknowledgment. Suppose now that we want an SR protocol that sends messages two at a time. That is, the sender will send a pair of messages and will send the next pair of messages only when it knows that both messages in the first pair have been received correctly. Suppose that the channel may lose messages but will not corrupt or reorder messages. Design an error-control protocol for the unidirectional reliable transfer of messages. Give an FSM description of the sender and receiver. Describe the format of the packets sent between sender and receiver, and vice versa. If you use any procedure calls other than the ones used in class (for example, udt_send(), start_timer(), rdt_rcv(), and so on), clearly state their actions. Give an example (a timeline trace of sender and receiver) showing how your protocol recovers from a lost packet.*

In our solution, the sender will wait until it receives an ACK for a pair of messages (seqnum and seqnum+1) before moving on to the next pair of messages. Data packets have a data field and carry a two-bit sequence number. That is, the valid sequence numbers are 0, 1, 2, and 3. (Note: you should think about why a 1-bit sequence number space of 0, 1 only would not work in the solution below.) ACK messages carry the sequence number of the data packet they are acknowledging.

The FSM for the sender and receiver are shown in Figure 2. Note that the sender state records whether (i) no ACKs have been received for the current pair, (ii) an ACK for seqnum (only) has been received, or an ACK for seqnum+1 (only) has been received. In this figure, we assume that the seqnum is initially 0, and that the sender has sent the first two data messages (to get things going). A timeline trace for the sender and receiver recovering from a lost packet is shown below:


sender
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_ack(seqnum+1)
seqnum = seqnum+2
udt_send(sndpkt, seqnum)
udt_send(sndpkt, seqnum+1)
start_timer
(rdt_rcv(rcvpkt) && corrupt(rcvpkt)) || (rdt_rcv(rcvpkt) && notcorupt(rcvpkt) && has_not_ACK(seqnum) && has_not_ACK(seqnum+1))
wait for pair of ACKs
timeout
udt_send(sndpkt, seqnum)
udt_send(sndpkt, seqnum+1)
start_timer
wait for odd ACK
timeout
udt_send(sndpkt, seqnum+1)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_ack(seqnum)
(rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_ack(seqnum)) || (rdt_rcv(rcvpkt) && corrupt(rcvpkt))
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_ack(seqnum+1)
(rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_ack(seqnum+1)) || (rdt_rcv(rcvpkt) && corrupt(rcvpkt))
wait for even ACK
timeout
udt_send(sndpkt, seqnum)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_ack(seqnum)
seqnum = seqnum+2
udt_send(sndpkt, seqnum)
udt_send(sndpkt, seqnum+1)
start_timer


receiver
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq(seqnum+1)
udt_send(ACK,seqnum+1)
seqnum = seqnum+2
(rdt_rcv(rcvpkt) && corrupt(rcvpkt))
wait for pair of data
(rdt_rcv(rcvpkt) && notcorupt(rcvpkt) && has_seq(x) && x != seqnum && x != seqnum+1
udt_send(ACK, x)
wait for odd data
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq(seqnum)
udt_send(ACK, seqnum)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq(seqnum)
udt_send(ACK, seqnum)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq(seqnum+1)
udt_send(ACK, seqnum+1)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq(seqnum+1)
udt_send(ACK, seqnum+1)
wait for even data
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq(seqnum)
udt_send(ACK,seqnum)
seqnum = seqnum+2

```
Sender                                      Receiver

    make pair (0,1)
    send packet 0
                       Packet 0 drops
    send packet 1
                                                 receive packet 1
                                                 buffer packet 1
                                                 send ACK 1
    receive ACK 1
```

```
(timeout)
resend packet 0
                                             receive packet 0
                                             deliver pair (0,1)
                                             send ACK 0
receive ACK 0
```

Question 2 *(GBN & SR Protocols) 6 points:*

(a) *4 points**:** Consider the GBN and SR protocols. Suppose the sequence number space is of size k. What is the largest allowable sender window that will avoid the occurrence of problems such as that shown on slide 3.54 for each of these protocols?*

In order to avoid the scenario of Figure 3.27, we want to avoid having the leading edge of the receiver's window (i.e., the one with the “highest” sequence number) wrap around in the sequence number space and overlap with the trailing edge (the one with the "lowest" sequence number in the sender's window). That is, the sequence number space must be large enough to fit the entire receiver window and the entire sender window without this overlap condition. So, we need to determine how large a range of sequence numbers can be covered, at any given time, by the receiver and sender windows.

Suppose that the lowest-sequence number that the receiver is waiting for is packet m. In this case, it's window is [m,m+w-1] and it has received (and ACKed) packet m-1 and the w-1 packets before that, where w is the size of the window. If none of those w ACKs have been yet received by the sender, then ACK messages with values of [m-w,m-1] may still be propagating back. If no ACKs with these ACK numbers have been received by the sender, then the sender's window would be [m-w,m-1].

Thus, the lower edge of the sender's window is m-w, and the leading edge of the receivers window is m+w-1. In order for the leading edge of the receiver's window to not overlap with the trailing edge of the sender's window, the sequence number space must thus be big enough to accommodate 2w sequence numbers. That is, the sequence number space must be at least twice as large as the window size, $k \geq 2w$.

(b) 2 points: Answer true or false to the following questions and briefly justify your answer: (i) With the SR protocol, it is possible for the sender to receive an ACK for a packet that falls outside of its current window. (ii) With GBN, it is possible for the sender to receive an ACK for a packet that alls outside of its current window.

(i) True. Suppose the sender has a window size of 3 and sends packets 1, 2, 3 at $t0$. At $t1$ $(t1 > t0)$ the receiver ACKS 1, 2, 3. At $t2$ $(t2 > t1)$ the sender times out and resends 1, 2, 3. At $t3$ the receiver receives the duplicates and re-acknowledges 1, 2, 3. At $t4$ the sender receives the ACKs that the receiver sent at $t1$ and advances its window to 4, 5, 6. At $t5$ the sender receives the ACKs 1, 2, 3 the receiver sent at $t2$. These ACKs are outside its window.

(ii) True. By essentially the same scenario as in (i).

Question 3 (RTT Estimate/Timeout Computation) *5 points***:** Run the *ping* command to a destination at least 10 hops away. Collect the delay for 101 ping packets. Setting *(i)* the delay of the first ping packet (call it packet #0) to be the initial estimated RTT, *(ii)* the delay of the second ping packet (call it packet number #1) to be the first sample RTT, and *(iii)* the initial devRTT to 0, use the TCP formulae to find the estimated RTT, estimated RTT variation, and the timeout interval from the ping delay data for packet #1 to packet #100. Plot the sample RTT, the estimated RTT, and the timeout interval on the y-axis of a graph with the packet sequence numbers (1-100) on the x-axis. Please also print the destination host name and IP address to which the ping packets were sent, and the time of the day you collected the data.

***Question** 4 (TCP Congestion Control) 6 points**:** Assuming TCP Reno is the protocol experiencing the behavior shown in figure below, answer the following questions. In all cases, you should provide a short discussion justifying your answer. (9/6 points for each part)*

a. Identify the intervals of time when TCP slow start is operating.

b. Identify the intervals of time when TCP congestion avoidance is operating.

c. After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

d. After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

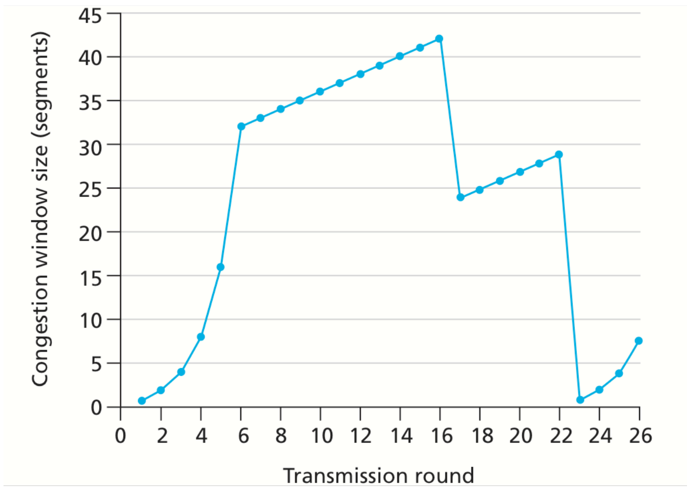
e. What is the value of ssthresh at the 18th transmission round?

f. During what transmission round is the 70th segment sent?

g. Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh?

h. Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 16th round. What are the ssthresh and the congestion window size at the 19th round?

i. Again suppose TCP Tahoe is used, and there is a timeout event at 22nd round. How many packets have been sent out from 17th round till 22nd round, inclusive?



a) TCP slowstart is operating in the intervals [1,6] and [23,26]
b) TCP congestion advoidance is operating in the intervals [6,16] and [17,22]
c) After the 16th transmission round, packet loss is recognized by a triple duplicate ACK. If there were a timeout, the congestion window size would have dropped to 1.
d) After the 22nd transmission round, segment loss is detected due to timeout, and hence the congestion window size is set to 1.
e) The threshold is set to half the value of the congestion window when packet loss is detected. When loss is detected during transmission round 16, the congestion windows size is 42. Hence the threshold is 21 during the 18th transmission round.
f) During the 1st transmission round, packet 1 is sent; packet 2-3 are sent in the 2nd transmission round; packets 4-7 are sent in the 3rd transmission round; packets 8-15 are sent in the 4th transmission round; packets 16-31 are sent in the 5th transmission round; packets 32-63 are sent in the 6th transmission round; packets 64 – 96 are sent in the 7th transmission round. Thus packet 70 is sent in the 7th transmission round.
g) The congestion window and threshold will be set to half the current value of the congestion window (8) when the loss occurred. Thus the new values of the threshold and window will be 4.
h) Threshold is 21, and congestion window size is 1.
i) round 17, 1 packet; round 18, 2 packets; round 19, 4 packets; round 20, 8 packets; round 21, 16 packets; round 22, 21 packets. So, the total number is 52.

***Question 5** (TCP Fairness) 3 points**:** Consider the plot used in the class to explain TCP fairness. Using this plot, explain why fairness cannot be achieved if MIMD or MIAD is used instead of AIMD.*

In both cases, multiplicative increase will increase the bandwidth of the session that has a higher initial bandwidth. A multiplicative decrease will be unable to bring about equal sharing of bandwidth. An additive decrease by equal amounts will make the disparity between the sessions even higher in comparison to multiplicative decrease.

Question 6 (Number of Transmissions) *5 points*:

(a) 1 point: What is the average number of retransmissions necessary to reliably transmit a packet from a source to a destination when the network can drop packets independently with probability 0.2?

Expected No. of Retransmissions = Expected no. of transmissions – 1 = 1/(1 – 0.2) – 1 = 0.25.

(b) 2 points: What is the approximate number of transmissions necessary to reliably transmit a packet from a source to 10000 receivers when the network can drop packets independently for each receiver with a probability 0.2?

Let M be the number of required transmissions. Then the average number of transmissions, E[M] for a loss probability p and the number of receivers R, is given by the following expression.

$$E[M] = 1 + \sum_{m=1}^{\infty} 1 - (1 - p^{m})^{R}$$

For R = 10000, and p = 0.2, E[M] is approximately equal to 7.584.

(c) 2 points: What is the probability of at least one receiver not receiving a packet when it is multicast from a source to 10000 receivers and when the network can drop packets independently for each receiver with a probability 0.02?

The probability of at least one receiver not receiving a packet when it is multicast from a source to R receivers and when the network can drop packets independently for each receiver with probability = $1 - (1\text{-}p)^{R}$. When R = 10000 and p = 0.02, this probability is almost 1.

Question 7 (required for cs6480, extra credit for cs5480) *10 points***:**
Read the following paper: "*Fast and Robust Signaling Overload Control,*" Sneha K. Kasera et al, in the IEEE International Conference on Network Protocols, November 2001. Extended version is available from
http://www.cs.utah.edu/~kasera/myPapers/icnp2Ton.pdf.
Answer the following questions that are based on this paper.

(a) 4 points: The overload control occupancy algorithm described in Section 4.1 of the paper uses a multiplicative increase and multiplicative decrease strategy. Does this strategy suffer from "fairness" issues discussed in the context of TCP congestion control? Rewrite the f_{n+1} equation so that an additive increase and multiplicative decrease (AIMD) strategy is followed with α being the maximum additive increment and β being the minimum multiplicative decrement.

The overload control occupancy algorithm described in Section 4.1 of the paper uses a multiplicative increase and multiplicative decrease strategy. This strategy does not suffer from the unfairness that is possible when two TCP connections sharing a bottleneck link use MIMD. This is because, in the context of signaling overload control presented in the paper, we are not concerned about fairness across calls.

f_{n+1} = restrict(f_{min}, x, 1) where restrict(a,x,b) = max(a, min(x,b))

$x = \max(\beta, \rho_{targ}/\rho_n) f_n$ when $\rho_n > \rho_{targ}$ (multiplicative decrease)
$x = f_n + \min((\rho_{targ}/\rho_n - 1) f_n, \alpha)$ when $\rho_n <= \rho_{targ}$ (additive increase)

(b) 2 points: Is queue length a robust measure for system load? Explain.

Queue length is not a robust measure of system load for several reasons including the following two. First, the queue length threshold must be changed if the system's processing capacity changes. Second, queue length does not indicate the processing times required by individual tasks queued in the system. Each task might need a different amount of processing resources. On the other hand, processor occupancy is a robust measure of the system load because it is dimensionless. Therefore, the processor occupancy threshold is dimensionless as well. This means that once we set an occupancy threshold we do not need to change that when there is a change in the processor or the processor capacity.

(c) 4 points: What is two-layer throttling? Justify the choice of r and d in Section 6. When the throttling fraction, $f = 0.5$, the cost of releasing a call, c_r, is 0.5, and the cost of dropping a call, c_d, is 0.1, what will be the reduction in cost of throttling a call using two-layer throttling. Suggest a different choice of r and d such that all the properties of a good solution (as stated in Section 6) are satisfied?

Two-layer throttling is: dropping some connection requests at the MTP3 layer (so that cost of dropping is small) and dropping some connection requests at the ISUP layer (so that REL messages containing congestion information are sent to adjacent switches). The choice of r and d in Sec.6 satisfies the requirement that as the system becomes more and more loaded, lesser and lesser REL messages should be sent. The choice here progressively decreases the number of REL messages sent as the load on the switch progressively increases.

Without two layer throttling, the cost of releasing a call = 0.5.
$d = (1 - f)*(1 – f) = 0.25$, $r = f*(1-f) = 0.25$.

With two layer throttling, the cost of releasing a call
$= ((0.25*0.1) + (0.25*0.5))/0.5 = 0.3$.
So, there is a 40% reduction in the cost of throttling.

Any increasing function of f can be equated with the ratio $r/(r+d)$ to get values of r and d satisfying the properties. For example, equating $r/(r+d)$ to $2f$ gives $r = 2f^2$ and $d = f – 2f^2$. Equating $r/(r+d)$ to f^2 gives $r = f^3$ and $d = f – f^3$.