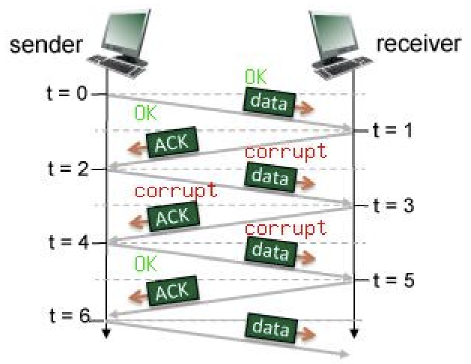
CMPSC 381
Data Communications and Networks
Fall 2012
Bob Roos

Lab 7
Due Thursday, 8 November, 1:30 pm

Create a document with your answers to the problems and upload it to your dropbox.

1.
Reliable Data Transfer: rdt2.2 (sender and receiver actions)

Consider the figure below, which shows four data packets and three corresponding ACKs being exchanged between a sender and receiver according to protocol rdt2.2, see pages 209-212 in the text. (Here are the finite state machine representations for the rdt2.2 sender and receiver.) The channel connecting the sender and receiver can corrupt (but not lose or reorder) packets. The actual corruption or successful transmission/reception of a packet is indicated by the corrupt and OK labels, respectively, shown above the packets in the figure below.



- Fill out the table below indicating *(i)* the state of the sender and the receiver just *after* the the transmission of a new packet in response to the received packet at time *t* , *(ii)* the sequence number associated with the data packet or the ACK number associated with the ACK packet sent at time *t*.

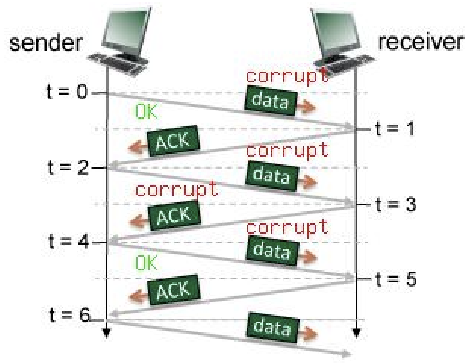
- Fill out the table below indicating *(i)* the state of the sender and the receiver just *after* the the transmission of a new packet in response to the received packet at time *t* , *(ii)* the sequence number associated with the data packet or the ACK number associated with the ACK packet sent at time *t*.

t	sender state	receiver state	packet type sent	seq. # or ACK # sent
0		Wait 0 from below	data	
1			ACK	
2			data	
3			ACK	
4			data	
5			ACK	
6			data	

- How many times is the payload of the received packet passed up to the higher layer at the receiver in the above example? At what times is the payload data passed up?

2.
Reliable Data Transfer: rdt2.2 (sender and receiver actions)

Consider the figure below, which shows four data packets and three corresponding ACKs being exchanged between a sender and receiver according to protocol rdt2.2, see pages 209-212 in the text. (Here are the finite state machine representations for the rdt2.2 sender and receiver.) The channel connecting the sender and receiver can corrupt (but not lose or reorder) packets. The actual corruption or successful transmission/reception of a packet is indicated by the corrupt and OK labels, respectively, shown above the packets in the figure below.



- Fill out the table below indicating *(i)* the state of the sender and the receiver just *after* the the transmission of a new packet in response to the received packet at time *t* , *(ii)* the sequence number associated with the data packet or the ACK number associated with the ACK packet sent at time *t*.

t	sender state	receiver state	packet type sent	seq. # or ACK # sent
0		Wait 0 from below	data	
1			ACK	
2			data	
3			ACK	
4			data	
5			ACK	
6			data	

- How many times is the payload of the received packet passed up to the higher layer at the receiver in the above example? At what times is the payload data passed up?

3.
Computing an Internet checksum

Consider the two 16-bit words (shown in binary) below. Recall that to compute the Internet checksum of a set of 16-bit words, we compute the one's complement sum [1] of the two words. That is, we add the two numbers together, making sure that any carry into the 17th bit of this initial sum is added back into the 1's place of the resulting sum); we then take the one's complement of the result. Compute the Internet checksum value for these two 16-bit words:

```
00100100 01111011        this binary number is 9339 decimal (base 10)
10011100 10011111        this binary number is 40095 decimal (base 10)
```

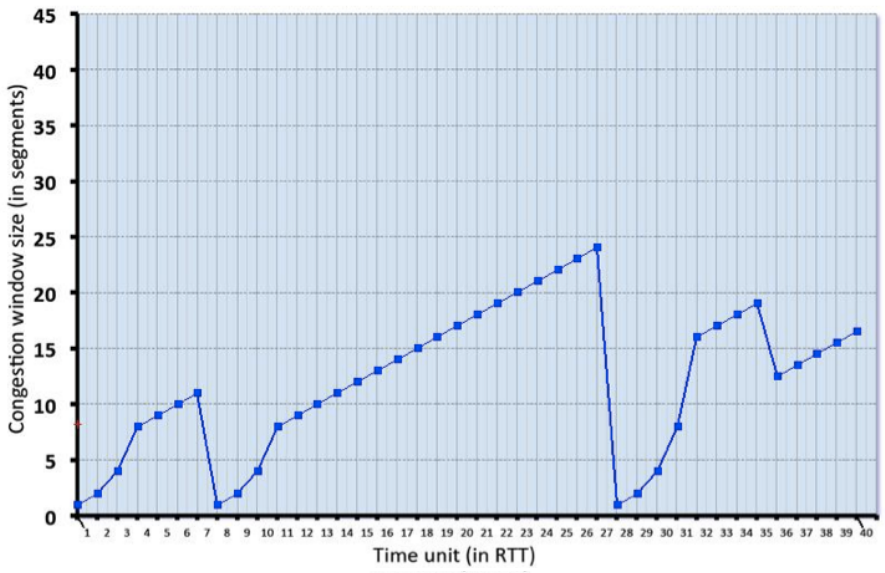
4.
Computing an Internet checksum

Consider the two 16-bit words (shown in binary) below. Recall that to compute the Internet checksum of a set of 16-bit words, we compute the one's complement sum [1] of the two words. That is, we add the two numbers together, making sure that any carry into the 17th bit of this initial sum is added back into the 1's place of the resulting sum); we then take the one's complement of the result. Compute the Internet checksum value for these two 16-bit words:

```
10100110 00101010        this binary number is 42538 decimal (base 10)
11101011 00101101        this binary number is 60205 decimal (base 10)
```

5.
TCP in action: slow start, congestion avoidance, and fast retransmit.

Consider the figure below, which plots the evolution of TCP's congestion window at the beginning of each time unit (where the unit of time is equal to the RTT); see Figure 3.53 in the text. In the abstract model for this problem, TCP sends a "flight" of packets of size `cwnd` at the beginning of each time unit. The result of sending that flight of packets is that either *(i)* all packets are ACKed at the end of the time unit, *(ii)* there is a timeout for the first packet, or *(iii)* there is a triple duplicate ACK for the first packet. In this problem, you are asked to reconstruct the sequence of events (ACKs, losses) that resulted in the evolution of TCP's `cwnd` shown below.

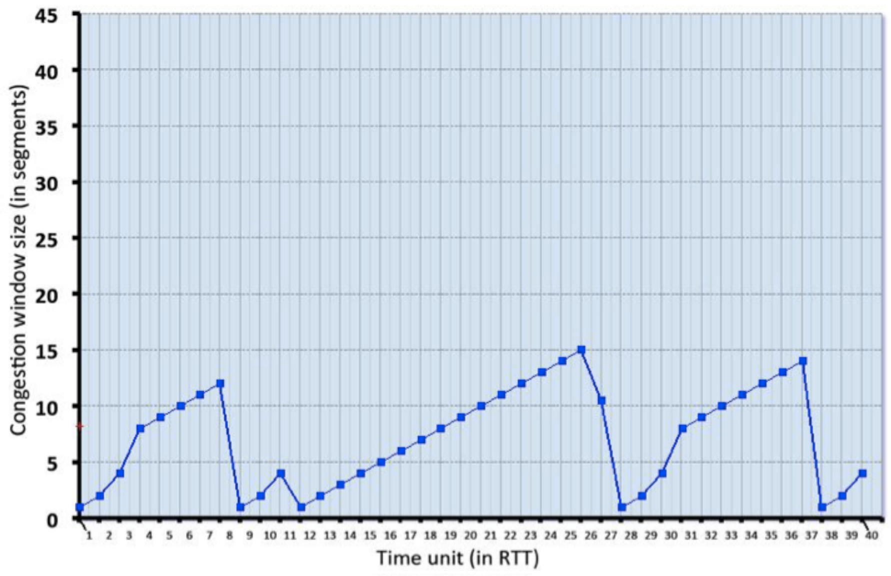


Consider the evolution of TCP's congestion window in the example above and answer the following questions. The initial value of `cwnd` is 1 and the initial value of `ssthresh` (shown as a red +) is 8.

- Give the times at which TCP is in slow start, congestion avoidance and fast recovery at the start of a time slot, when the flight of packets is sent.
- Give the times at which the first packet in the sent flight of packets is lost, and indicate whether that packet loss is detected via timeout, ot by triple duplicate ACKs.
- Give the times at which the value of `ssthresh` changes, and give the new value of `ssthresh`.

6. **TCP in action: slow start, congestion avoidance, and fast retransmit.**

Consider the figure below, which plots the evolution of TCP's congestion window at the beginning of each time unit (where the unit of time is equal to the RTT); see Figure 3.53 in the text. In the abstract model for this problem, TCP sends a "flight" of packets of size `cwnd` at the beginning of each time unit. The result of sending that flight of packets is that either *(i)* all packets are ACKed at the end of the time unit, *(ii)* there is a timeout for the first packet, or *(iii)* there is a triple duplicate ACK for the first packet. In this problem, you are asked to reconstruct the sequence of events (ACKs, losses) that resulted in the evolution of TCP's `cwnd` shown below.



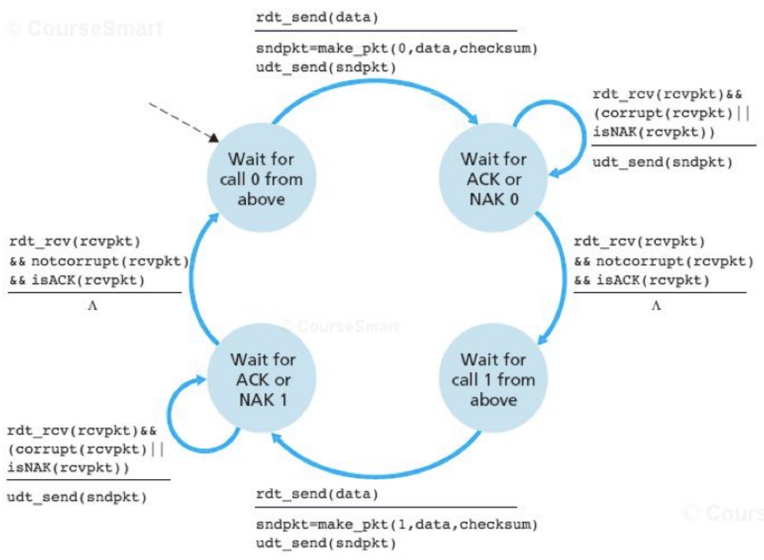
Consider the evolution of TCP's congestion window in the example above and answer the following questions. The initial value of `cwnd` is 1 and the initial value of `ssthresh` (shown as a red +) is 8.

- Give the times at which TCP is in slow start, congestion avoidance and fast recovery at the start of a time slot, when the flight of packets is sent.
- Give the times at which the first packet in the sent flight of packets is lost, and indicate whether that packet loss is detected via timeout, ot by triple duplicate ACKs.
- Give the times at which the value of `ssthresh` changes, and give the new value of `ssthresh`.

7. Problem P3, page 288.

8. Problem P9, page 289.

9. Problem P10, page 289. Note: The assumption here is that there is some constant such that the RTT is *always* less than this constant. I have reproduced the rdt2.1 diagrams below.



3.11 ♦ rdt2.1 sender

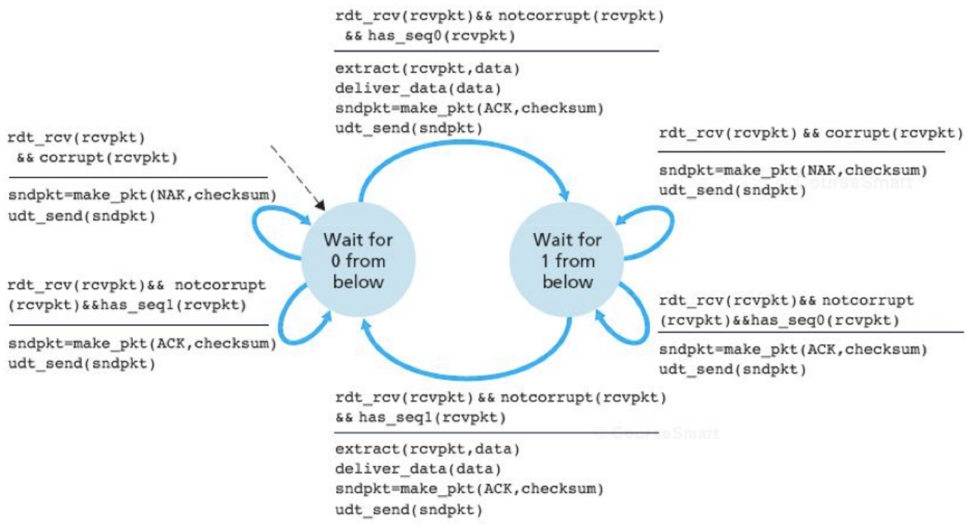

rdt_rcv(rcvpkt)&& notcorrupt(rcvpkt)
&& has_seq0(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt)
&& corrupt(rcvpkt)
sndpkt=make_pkt(NAK,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
sndpkt=make_pkt(NAK,checksum)
udt_send(sndpkt)
Wait for 0 from below
Wait for 1 from below
rdt_rcv(rcvpkt)&& notcorrupt
(rcvpkt)&&has_seq1(rcvpkt)
sndpkt=make_pkt(ACK,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt)&& notcorrupt
(rcvpkt)&&has_seq0(rcvpkt)
sndpkt=make_pkt(ACK,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK,checksum)
udt_send(sndpkt)

Figure 3.12 ♦ rdt2.1 receiver

10. Download program `sim.py`. Read the header information. The point of this simple simulation is to show that even if the router can switch one packet from each input queue, “keeping up” with the packet arrival rate, dropped packets can still occur.

    (a) Initially, the program is set up for 3 input and output ports, with 3 packets arriving each time unit The simulation runs for 1000 time units, so there are a total of 3000 packets. About what percentage of these packets are dropped? (Run several simulations and take an average.)

    (b) By modifying the value of `maxq`, the maximum queue size, what queue size seems to be needed before the percentage of dropped packets drops below 5 (on average)? Again, try multiple experiments.

(c) Return `maxq` to 5, but now play around with the probability, `prob`, currently set to 1. What value of `prob` seems to reduce the number of dropped packets below 5 (on average)?

(d) Suppose we now have *exactly one* packet arriving at each input port at each time unit (rather than randomly selecting an input port). Will the input queues ever overflow? What about the output queues (assuming that we randomly select an output port for each incoming packet)? You don't have to use the simulation to answer this, but you can try modifying the code so that exactly one packet per port arrives at each time unit.