You will hand in your solution using SVN. Follow the instructions on Piazza to add and commit new files.

**TCP Congestion**

1. Consider a TCP system implementing slow start and congestion avoidance with fast retransmit and fast recovery. When a connection is setup the congestion window is initialized to one segment and the slow start threshold to 64 segments. To simplify the problem, assume that the timeout is equal to the RTT (an exact estimate) and specify time in units of RTT, such that one time slot is one RTT.

   Packet transmissions are such that at each time slot the sender sends all packets in the congestion window. If ACK's are received in the next time slot, there is no timeout. In addition, to simplify matters either the entire window is acknowledged or none of its segments are acknowledged.

   For a particular connection, ACK’s are received in time slots 1-9, 11-18, 20-23, 25-45, and 47-50. Timeouts occur in slots 10, 19 and 24; in slot 46, three duplicate ACK’s are received for the packets sent in time slot 45.

   For the system described, plot both the congestion window and the slow start threshold (on the same graph) versus time (slots), between slots 0 and 50. The values for each slot are the ones that the CW and the threshold assume at the end of the slot. Remember to consider the differences between slow start and congestion avoidance with fast retransmit and fast recovery when changing the congestion window. Ignore the value of SWS.
CW has exponential growth until it reaches 64 (slot7), then it grows linearly in slots 8 and 9, reaching 66 packets when the timeout occurs. CW goes to 1, the threshold is set to 33. The exponential growth brings the CW back to 64 (first power of two larger than the threshold), after which additive increase brings the CW to 66 when the second timeout happens. Again, the threshold is halved, and the CW brought back to 1. The CW increases exponentially to 2, 4, 8, 16 packets when one more timeout happens, at slot 24. CW is set to 1 once more, and the threshold is halved. The exponential increase brings the CW value to 8 (the threshold value) at round 27, after which the linear increase starts. At slot 45, when the three acks arrive, the CW value is 26, the threshold and the CW are halved, and the CW starts growing linearly.

(Comments: The dash line is also accepted as a right answer)

Random Early Detection

2. Consider a RED gateway with MaxP = 0.06, and with an average queue length halfway between the two thresholds MinThreshold and MaxThreshold.

   a. Find the drop probability \( P_{\text{count}} \), for count = 1 and count = 20

   b. Calculate the probability that none of the 20 packets are dropped. (Describe the method, and provide a numerical result approximate to the 4th decimal place)

   Sol:
   
   (a) \( \text{AvgLength} = \frac{\text{MinTh} + \text{MaxTh}}{2} \)
   
   \( \text{TempP} = \frac{\text{MaxP} \cdot (\text{AvgLength} - \text{MinTh})}{\text{MaxTh} - \text{MinTh}} = 0.06 \cdot 0.5 = 0.03 \)
   
   \( P_1 = \frac{1 - 1 \cdot \text{TempP}}{\text{TempP}} = \frac{0.97}{0.03} = 0.031 \)
   
   \( P_{20} = \frac{1 - 20 \cdot \text{TempP}}{1 - 0.6} = 0.03 \)

   (b) \( P_{\text{No\_drops}} = (1 - P_1) \cdot (1 - P_2) \cdot \ldots (1 - P_{20}) = 0.3814 \)

TCP and Network Delay
3. Consider the following two causes of a 2-second network delay (assume ACKs return instantaneously).

- One intermediate router with a 2-second outbound per-packet bandwidth delay and no competing traffic.
- One intermediate router with a 100-ms outbound per-packet bandwidth delay and with a steadily replenished (from another source) 20 packets in the queue.

a. How might a transport protocol in general distinguish between these two cases?

b. Suppose TCP Vegas sends over the above connections, with an initial CongestionWindow of 3 packets. What will happen to CongestionWindow in each case? Assume BaseRTT = 2 second and β is one packet per second.

Sol:
(a) If we send a single packet, we see a 2 seconds delay (in either case). Now consider the case where we send a burst of 10 packets. In the first case we will get ACKs back at 2 seconds intervals and the last packet will have an RTT of 20 seconds. However, the second case will have a 2 second RTT for the first packet and a 3 second RTT for the last packet.

The technique of packet-pairs (sending multiple instances of two consecutive packets right after one another and analyzing the minimum time difference between their ACKs) achieves the same effect. In fact, packet-pair is sometimes thought of as a technique to determine the minimum path bandwidth. In the first case, the two ACKs of a pair will always be 2 second apart. In the second case, the two ACKs will sometimes be only 100 ms apart.

(b) In the first case, TCP-Vegas will measure RTT=6 as soon as there is a full window of outstanding packets. This means the ActualRate is down to 0.5 packet/second. However, BaseRTT is 2 second, so ExpectedRate = CongestionWindow/BaseRTT =1.5packets/second. Hence, Diff is 1 packets/second and CongestionWindow will not be changed.

In the second case, when a burst of these packets is sent, the measured RTTs are 2.0, 2.1, and 2.2 seconds. Expected rate is: Expected Rate = (3 pkts)/(2s) =1.5packets/second. Note that the actual rate is 3/2.2 = 1.36packets/second (because it takes 2.2 seconds to get those three packets). So diff is 0.14, which is smaller than beta, so you do not decrease the congestion window size, and it will increase or stay the same depending on whether alpha is greater than or less than 0.14.

**Fair Queueing**

4. Suppose that the following packets arrivals:

- Flow 1 A:5, B:4, C:6, D:7
- Flow 3 J:1, K:5, L:1, M:1, N:6, O:5, P:4
- Flow 4 Q:6, R:6, S:6, T:5
Assume that at the current time, all the packets have arrived and are now sitting in the per-flow queues. The number after the colon is the size of the packet, so packet A is 5 units in size.

(a) List the order of departure of these packets under the packetized weighted fair queuing scheme, with equal weights for each flow. Break ties by going with the lowest-numbered flow.

*Sol:*

*Bit-by-bit schedule is:*

<table>
<thead>
<tr>
<th>Flow 1:</th>
<th>Flow 2:</th>
<th>Flow 3:</th>
<th>Flow 4:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAABBBBBC</td>
<td>EEEGFFGHHH</td>
<td>JKKKKKL</td>
<td>QQQQQQQ</td>
</tr>
<tr>
<td>CCCCCDDDD</td>
<td>/DDDDDD</td>
<td></td>
<td>/SSSSS</td>
</tr>
</tbody>
</table>


*Comments: If follow the scheme mentioned in the slides, the solution should be: AEJQKBFLMNCGODHTIP (both the above solutions are right)*

(b) Now list the order of departure assuming that the four flows have weights 3, 1, 2, 1 respectively.

*Sol:*

To model different sending weights, we multiply packet sizes for the flows by 2, 6, 3, 6 respectively.

Then we have

<table>
<thead>
<tr>
<th>Flow 1:</th>
<th>Flow 2:</th>
<th>Flow 3:</th>
<th>Flow 4:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAABBBBBC</td>
<td>EEEGFFGHHH</td>
<td>JKKKKKL</td>
<td>QQQQQQQ</td>
</tr>
<tr>
<td>CCCCCDDDD</td>
<td>/DDDDDD</td>
<td></td>
<td>/SSSSS</td>
</tr>
</tbody>
</table>


*Comments: If follow the scheme mentioned in the slides, the solution should be: AEJQKBCMNDROPSGHTI (both the above solutions are right)*

**TCP Slow Start**

5. Although slow start with congestion avoidance is an effective technique for coping with congestion, it can result in long recovery times in high-speed networks.

   a. Assume a RTT delay of 80 ms and a link with an available bandwidth of 1 Gbps and a segment size of 1024 bytes. Determine the **window size needed** to keep the pipe full and **the time it will take to reach that window size** after a timeout using the
Jacobson Algorithm, assuming that the sender was transmitting at the full window before the timeout,

b. Repeat for a segment size of 16 Kbytes. (Assume 1K = 1024)

**Sol:**

a. The size of the window (in bits) is $80\text{ms} \times 1\text{Gbps} = 80 \times 10^6\text{bits}$. Each packet has 8192 bits, so the window size is:

$$\left\lceil \frac{8 \times 10^7 \text{bits}}{8192 \text{ bits/packet}} \right\rceil = 9766 \text{ packets}$$

After the timeout, our congestion threshold is half the full window size (4883 packets). TCP will use exponential growth until the congestion threshold is passed, at which point additive increase begins. Therefore, we can use exponential increase to a window size of $2^{12} = 4096$. In the 13th RTT, after receiving 787 packet, (0.08 RTT) the window size becomes to 4883. So in the first 13 RTTs, we will transmit 4883 packets out. Now we do additive increase to get to the full window. This will take 9766 – 4883 = 4883 RTTs using additive increase (one packet increase in the window per RTT). So, the total time to reach the full window is $(13 + 4883)^\text{RTT} = 4896 \times 80\text{ms} = 391.68s$.

b. The window size (in bits) does not change, but we need to refresh the window size for packets of 16KB. Each packet is 131'072 bits, so the window is:

$$\left\lceil \frac{8 \times 10^7 \text{bits}}{131072 \text{ bits/packet}} \right\rceil = 611 \text{ packets}$$

This makes the congestion threshold 305 packets, as what we described in a, there should be $(9+306)^{0.08} = 25.2$ s