ECE 4380/6380: Computer Communications **Summer 2021**
Problem Set 10 **Due: 11:59 pm Wednesday, July 28**

Assigned reading: Peterson and Davie, Chapter 6, Sections 1 through 4. All problems have equal weight.

1. **Weighted fair queueing**.
A router implements weighted fair queueing (WFQ) for three incoming flows over a single outgoing channel. In this problem, you will determine the order that packets are sent out from the router and compare your solution with the fair queueing solution presented in class. Assume for simplicity that the problem starts at time t=0 and that sending a packet of length N requires N time units. Packets arrive on the three flows, named A, B, and C, as given in the table. The weights to use with WFQ are also given in the table.

Stream	Weight	Packet length	Arrival time
A	1	8	0, 10, 50, 60, 100
B	4	20	10, 30, 50
C	1	10	45, 50, 55, 95

All service counters start at 0 at time 0.

(a) For each packet sent on the outgoing link, record the start time, the service counter for each flow at the time, and the name of the packet (e.g., write B_3 for flow B's third packet).

Name	Start Time	Service Flow A	Service Flow B	Service Flow C
A1	0	0	0	0
A2	10	0	0	0
B1	18	8	0	0
B2	38	8	5	5
C1	58	8	10	5
A3	68	8	10	15
B3	76	16	10	15
C2	96	16	15	15
A4	106	16	16	25
A5	114	24	24	25
C3	122	32	25	25
C4	132	32	32	35

(b) The queueing delay for a packet is the difference between the arrival time of the packet and the time that it *begins* to be sent on the outgoing link. Compare your solution to that developed in class (that is, the same set up but with the weights for all flows equal to 1). Explain how the weight for flow B has changed the queueing delays for flow B's packets.

While B1 and B2 have the same queueing delay as the non-weighted example, B3's queueing delay is reduced by 18 units, as it moves ahead of A4 and C2.

2. **Additive increase – multiplicative decrease**.
Consider a simple congestion-control algorithm that

- uses linear increase and multiplicative decrease but not slow start,
- works in units of packets rather than bytes, and
- starts each connection with a congestion window equal to one packet.

The sender sends an entire window full in one batch. For every ACK of such a window full that the sender receives it increases its effective window (which is counted in packets) by one. When there is a timeout, the effective window is cut into half the number of packets. Assume that the delay is latency only, and that when a group of packets is sent, only a single ACK is returned with a single cumulative acknowledgement. Note that cumulative rather than selective ACK's are used (like is done in TCP and in the other sliding window protocols we have studied). For simplicity, assume a perfect timeout mechanism that detects a lost packet exactly 1 RTT after it is transmitted.

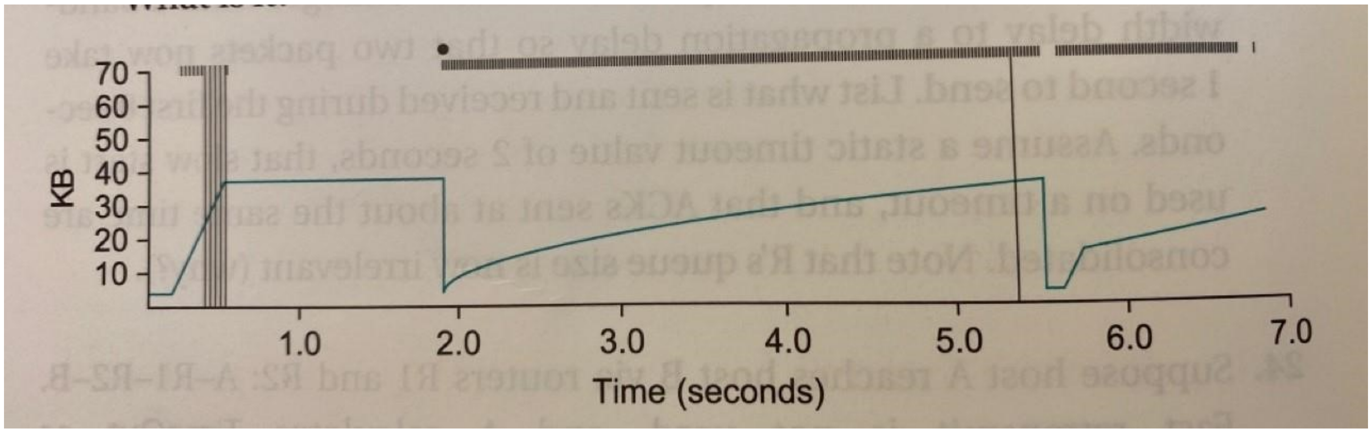
Make a table showing the size of the congestion control window and the packets that are sent. Assume that the following packets are lost: 9, 25, 30, 38, 50. Here is a table showing how it begins.

RTT	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Packets sent	1	2-3	4-6	7-10	9-10	11-13	14-17	18-22	23-28	25-27	28-31	30-31	32-34	35-38
Window size	1	2	3	4	2	3	4	5	6	3	4	2	3	4
Packet lost				9					25		30			38

RTT	15	16	17	18	19	20
Packets sent	38-39	40-42	43-46	47-51	50-51	52-54
Window size	2	3	4	5	2	3
Packet lost				50		

(Note after the 4-th round the cumulative ACK is for packets 7 and 8 because packet 9 was lost. The receiver keeps packet 10 in its buffer, but the sender does not know if packet 10 was received or not. Since the sender's window size is 2 in the 5-th round it sends two packets.)

3. **TCP trace**. Consider the TCP trace in the figure below. Identify time intervals representing slow start on startup, slow start after timeout, and linear-increase congestion avoidance. Explain what is going on from T = 0.5 to T = 1.9. The TCP version that generated this trace includes a feature absent from the TCP that generated Figure 162 (or Figure 6.11 in the book). What is this feature? This trace and the one in Figure 164 (or Figure 6.13 in the book) both lack a feature. What is it?



Slow Start on Startup: T=0.1 to 0.5
Slow Start on Timeout: T=5.7 to 6.9
Linear-Increase Congestion Avoidance: T=2.0 to 5.5

T=0.5 to 1.9: At some point, a lot of packets got lost around 0.3-0.4 seconds from the slow start. As such, many ACKs were not coming back, which means the congestion window doesn't change (until it realizes what happened at 1.9 seconds.)

This trace is missing the "Fast Retransmit" feature, which retransmits a lost packet faster than normal.
Both traces are missing the "Fast Recovery" feature, which avoids the slow start on timeout phases.

4. **RED router**. In RED gateways, explain why MaxThreshold is less than the actual size of the available buffer pool.

   So that it drops a few of the earlier packets to notify the source to decrease its congestion window. That way, it has a lower probability to drop more packets in the future.

5. **ECN versus RED**. For the DECbit congestion-avoidance mechanism, a router calculates its average queue size and if the average queue size is greater than a threshold, then it sets the congestion bit in the packet header. Experience with this approach has suggested that the congestion bit be set if the average queue size at the router is greater than one. As a consequence, an end host may see a large fraction of its packets marked by a congested router. On the other hand, a RED router will drop a packet before there is queue overflow as the mechanism to signal congestion to the end host. Clearly, we don't want to use a queue size of 1 to decide to drop packets as this may result in a large number of packets begin dropped. Dropping a large fraction of the packets will force timeouts at the source host instead of a fast retransmit and a multiplicative decrease in its congestion window. The optional standard explicit congestion notification (ECN) is similar to DECbit in that packets are marked instead of dropped. However, to minimize changes to TCP, an end host uses multiplicative decrease when it finds a packet that has been marked. Which algorithm for marking packets is likely to work better with ECN: the algorithm described with DECbit (i.e., calculate the average queue size and mark all packets if the average is above a threshold) or the algorithm described with RED (i.e., probabilistically decide if a packet should be marked based on the average queue size and the count of packets received since the last packet was marked).

RED is likely to work better with ECN than DECbit. This is because DECbit will mark more packets than necessary and heavily decrease the congestion window, thus slowing down the network.

6. **TCP**. Host A is sending an enormous file to Host B over a TCP connection. Over this connection there is never any packet loss and the timers never expire. Assume the transmission rate of the connection between Host A and Host B by *R* bps. Suppose that the process in Host A is capable of sending data into its TCP socket at a rate *S* bps, where $S = 10 \times R$. Further suppose that the TCP receive buffer is large enough to hold the entire file, and the send buffer can hold only one percent of the file. What would prevent the process in Host A from continuously passing data to its TCP socket at rate *S* bps? In giving your answer explain how TCP flow control or TCP congestion control limit the rate the process in Host A passes data to TCP, or explain why TCP flow control and TCP congestion control do not limit the rate and identify the mechanism that does limit the rate.

This scenario uses TCP congestion control to limit the rate. The sender is trying to throw out information faster than the network can handle it. As such, the network (rather than the receiver) tells the sender what its window size should be to avoid congestion and dropping packets.

7. **Adapting the Congestion Window**.
In this problem we consider a simple network and examine the throughput under three scenarios. In the first scenario, you are given the optimal congestion window size and timeout value. The other two scenarios consider adapting the congestion window using slow start and two different mechanisms to trigger retransmissions.

A simplified model for a network is considered. Suppose that between nodes A and B there is a router R. The link from A to R has an infinite bandwidth (i.e., packets are not delayed). However, the link from R to B introduces a transmission delay of 1 second per packet (e.g., if 2 packets arrive at R it takes 1 second to transmit the first packet, and the second packet waits for 1 second before being transmitted during the next second). To simplify the model, assume that acknowledgments from B through R back to A are sent instantaneously. Further assume that router R has queue size of one, in addition to the packet it is sending. At each second, the sender A first processes any arriving ACK's and then responds to any timeouts. Assume the application process at source A has an infinite backlog of packets to send to destination B, and that packets are numbered 1, 2, 3, …

Assume that the congestion window is fixed at one packet, and the timeout time set for each packet sent is 1 second. Note that at time 0, source A sends 1 packet because its congestion window is fixed at one, the packet is immediately transmitted by the router R (and the router's queue is empty). At time 1 the packet has completely arrived at the destination B, and an ACK is instantaneously returned to the source A. Source A cancels the timer for packet 1 and sends packet 2. Complete the table below. (Hint: this part is trivial, and the purpose is to just get you familiar with the table structure.)

Time	A receives ACK for packet #	Size of A's congestion window	A sends packet #	R sends packet #	R queues packets
0	-	1	1	1	Ø
1	1	1	2	2	Ø
2	2	1	3	3	Ø
3	3	1	4	4	Ø
4	4	1	5	5	Ø
5	5	1	6	6	Ø
6	6	1	7	7	Ø
7	7	1	8	8	Ø

As you found above, the network works well if the window size and timeout values are known. The problem is that TCP does not know the best window size or timeout value. For this part of the problem assume the network is the same and that the timeout value is fixed at 2 seconds (i.e., we will not adapt the timeout value). Now A sends data packets to B over a TCP connection, using slow start. The TCP buffer sizes at both A and B are arbitrarily large, so ignore the advertised window size. For the remainder of this problem we will assume that when TCP encounters a timeout it reverts to stop-and-wait as the outstanding lost packets in the existing window at the time of the timeout get retransmitted one at a time. The slow start phase does not begin again until all the packets in the existing window are acknowledged. Also, once one timeout and retransmission is pending, subsequent timeouts of later packets are suppressed until the earlier acknowledgment is received. That is, make a list of timeouts that occur and process them one at a time.

Fill in the table below. Note that to fill in the table you will need to keep track of some additional information. For example, you will need to keep track of when timers are set and if they expire, the list of packets that are in the window at both the source and destination, which packets are dropped, etc. (Hint: your solution should show that packets 5 and 7 are dropped because router R's queue is full when the packet arrives.)

Time	A recvs ACK #	Size of A's congestion window	A sends packet #	R sends packet #	R queues packets
0	-	1	1	1	~~0~~
1	1	2	2-3	2	3
2	2	3	4-5	3	4
3	3	4	6-7	4	6
4	4	1 (5 timed-out)	5	6	5
5	4 (Missing 5)	1	6	5	6
6	6 (Cum. ACK)	1	7	6	7
7	6 (Dup. ACK)	2 (Slow Start)	8-9	7	8
8	7	3	10-11	8	10

For the above part, you examined how the window size changes and the order packets are transmitted given a fixed timeout. For this part of the problem, we will look at how duplicate ACK's trigger retransmissions instead of timeouts. So, changes from the previous part are:

- The router has buffer space for three packets instead of just one.
- Fast retransmit is done the first time that a second duplicate ACK is received (that is, the third ACK of the same packet). Fast retransmit was not considered in the previous part. Also, ignore fast recovery; when a packet is lost let the window size be one.
- The timeout interval is infinite (that is, ignore all timeout timers as they are not needed for this example).

(Hint: your solution should show that in total five packets are dropped because router R's queue is full when the packet arrives. The first packet dropped is number 9 at time 4)

Time	A recvs ACK #	Size of A's congestion window	A sends packet #	R sends packet #	R queues packets
0	-	1	1	1	~~0~~, ~~0~~, ~~0~~
1	1	2	2-3	2	3, ~~0~~, ~~0~~
2	2	3	4-5	3	4, 5, ~~0~~
3	3	4	6-7	4	5, 6, 7
4	4	5	8-9	5	6, 7, 8
5	5	6	10-11	6	7, 8, 10
6	6	7	12-13	7	8, 10, 12
7	7	8	14-15	8	10, 12, 14
8	8	9	16-17	10	12, 14, 16
9	8	9	18	12	14, 16, 18

10	8	1	9	14	16, 18, 9
11	8	1	~~O~~	16	18, 9, ~~O~~
12	8	1	~~O~~	18	9, ~~O~~, ~~O~~
13	8	1	9	9	9, ~~O~~, ~~O~~
14	9	2	10-11	9	10, 11, ~~O~~
15	9	2	~~O~~	10	11, ~~O~~, ~~O~~
16	10	3	12-13	11	12, 13, ~~O~~
17	11	4	14-15	12	13, 14, 15
18	12	5	16-17	13	14, 15, 16