HW6 Solutions

5.20

(a)

T=0.0 'a' sent
T=1.0 'b' collected in buffer
T=2.0 'c' collected in buffer
T=3.0 'd' collected in buffer
T=4.0 'e' collected in buffer
T=4.1 ACK of 'a' arrives, "bcde" sent
T=5.0 'f' collected in buffer
T=6.0 'g' collected in buffer
T=7.0 'h' collected in buffer
T=8.0 'i' collected in buffer
T=8.1 ACK arrives; "fghi" sent

(b) The user would type ahead blindly at times. Characters would be echoed between 4 and 8 seconds late, and echoing would come in chunks of four or so. Such behavior is quite common over telnet connections.
(c) With the Nagle algorithm, the mouse would appear to skip from one spot to another. Without the Nagle algorithm the mouse cursor would move smoothly, but it would keep moving for one RTT after the physical mouse were stopped.

5.29

Sample RTT	Estimated RTT	Difference	|Difference|- Deviation	Deviation	New TimeOut (EstRTT + 4*Dev)	Old Timeout (2*EstRTT)
	1			0.1	1.4	2
5	1.5	4	3.9	0.5875	3.85	3
5	1.9375	3.5	2.9125	0.9515625	5.74375	3.875
5	2.3203125	3.0625	2.1109375	1.2154297	7.18203125	4.640625
5	2.6552734	2.6796875	1.4642578	1.3984619	8.249121094	5.31054688
5	2.9483643	2.3447266	0.9462646	1.516745	9.015344238	5.89672852
5	3.2048187	2.0516357	0.5348907	1.5836063	9.53924408	6.40963745
5	3.4292164	1.7951813	0.2115749	1.6100532	9.869429207	6.85843277
5	3.6255643	1.5707836	-0.03927	1.6051445	10.04614236	7.25112867
5	3.7973688	1.3744357	-0.230709	1.5763059	10.1025924	7.59473759
5	3.9476977	1.2026312	-0.373675	1.5295966	10.06608395	7.89539539
5	4.0792355	1.0523023	-0.477294	1.4699348	9.958974611	8.15847097
5	4.194331	0.9207645	-0.54917	1.4012885	9.799485043	8.3886621
5	4.2950397	0.805669	-0.59562	1.3268361	9.602383888	8.59007933

With the J/K algorithm, we time out 3 times … twice for the first row estimate, and once for the second row (remember that each time a packet times out, we retransmit with twice the timeout value). In the original algorithm, we time out 5 times, twice for row 1, and once for each of row 2, 3, and 4.
In J/K the timeout increases to 10.1 (the max), and then decreases to 5, whereas in the original algorithm, the timeout increases monotonically to 10, when the estimated RTT is 5.

6.16

(a) In slow start, the size of the window doubles every RTT. At the end of the ith RTT, the window size is 2^i KB. It will take 10 RTTs before the send window has reached 2^{10} KB = 1MB.

(b) After 10 RTTs, 1023KB = 1MB − 1KB has been transferred, and the window size is now 1MB. Since we have not yet reached the maximum capacity of the network, slow start continues to double the window each RTT, so it takes 4 more RTTs to transfer the remaining 9MB (the amounts transferred during each of these last 4 RTTs are 1MB, 2MB, 4MB, 1MB; these are all well below the maximum capacity of the link in one RTT of 12.5MB). Therefore, the file is transferred in 14 RTTs.

(c) It takes 1.4 seconds (14 RTTs) to send the file. The effective throughput is (10MB / 1.4s) = 7.1MBps = 57.1Mbps. This is only 5.7% of the available link bandwidth.

6.17

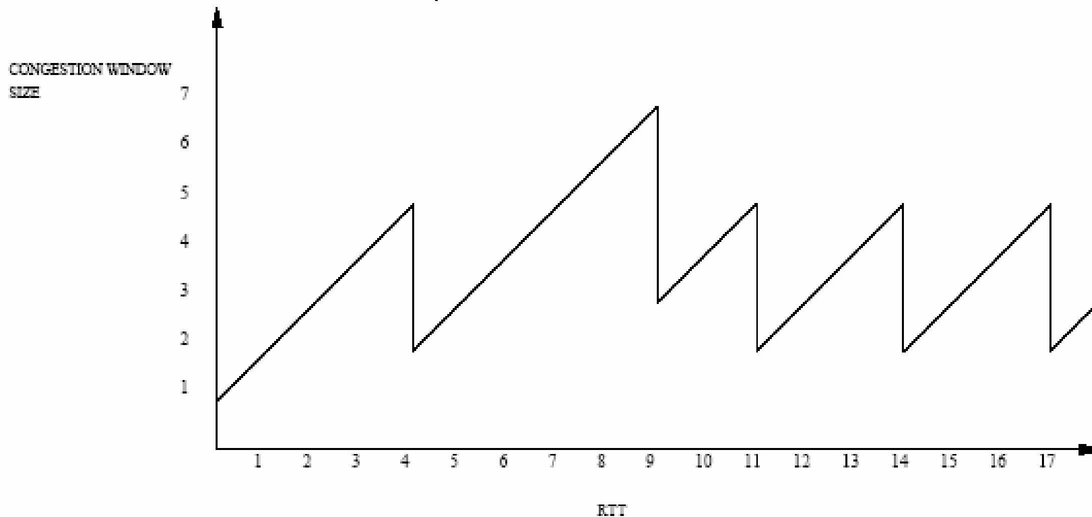
Let the sender window size be 1 packet initially. The sender sends an entire windowful in one batch; for every ACK of such a windowful that the sender receives, it increases its effective window (which is counted in packets) by one. When there is a timeout, the effective window is cut into half the number of packets.

Now consider the situation when the indicated packets are lost. The window size is initially 1; when we get the first ACK it increases to 2. At the beginning of the second RTT we send packets 2 and 3. When we get their ACKs we increase the window size to 3 and send packets 4, 5 and 6. When these ACKs arrive the window size becomes 4. Now, at the beginning of the fourth RTT, we send packets 7, 8, 9, and 10; by hypothesis packet 9 is lost. So, at the end of the fourth RTT we have a timeout and the window size is reduced to 4/2 = 2.

Continuing, we have

RTT	5	6	7	8	9
Sent	9-10	11-13	14-17	18-22	23-28

Again the congestion window increases up until packet 25 is lost, when it is halved, to 3, at the end of the ninth RTT. The plot below shows the window size vs. RTT.



6.18

From the figure for the preceding exercise we see that it takes about 17 RTTs for 50 packets, including the necessary retransmissions. Hence the effective throughput is $50/17 \times 100 \times 10^{-3}$ KB/s = 29.4KB/s.

6.19

The formula is accurate if each new ACK acknowledges one new MSS-sized segment. However, an ACK can acknowledge either small size packets (smaller than MSS) or cumulatively acknowledge many MSS's worth of data.
Let N = CongestionWindow/MSS, the window size measured in segments. The goal of the original formula was so that after N segments arrived the net increment would be MSS, making the increment for one MSS-sized segment MSS/N. If instead we receive an ACK acknowledging an arbitrary AmountACKed, we should thus expand the window by
Increment = AmountACKed/N
= (AmountACKed × MSS)/CongestionWindow

6.26

The router is able in principle to determine the actual number of bytes outstanding in the connection at any time, by examining sequence and acknowledgement numbers. This we can take to be the congestion window except for immediately after when the latter decreases.
The host is complying with slow start at startup if only one more packet is outstanding than the number of ACKs received. This is straightforward to measure.
Slow start after a coarse-grained timeout is trickier. The main problem is that the router has no way to know when such a timeout occurs; the TCP might have inferred a lost packet by some other means. We may, however, on occasion be able to rule out three duplicate ACKs, or even two, which means that a retransmission might be inferred to represent a timeout.
After any packet is retransmitted, however, we should see the congestion window fall at least in half. This amounts to verifying multiplicative decrease, though, not slow start.

6.28

Slow start is active up to about 0.5 sec on startup. At that time a packet is sent that is lost; this loss results in a coarse-grained timeout at T=1.9.
At that point slow start is again invoked, but this time TCP changes to the linear-increase phase of congestion avoidance before the congestion window gets large enough to trigger losses. This occurs sometime around T=2.4.
At T=5.3 another packet is sent that is lost. This time the loss is detected at T=5.5 by fast retransmit; this TCP feature is the one not present in Figure 6.11 of the text, as all lost packets there result in timeouts. Because the congestion window size then drops to 1, we can infer that fast recovery was not in effect; instead, slow start opens the congestion window to half its previous value and then linear increase takes over. The transition between these two phases is shown here, at T=5.7.