Home Work 5 Solution

EE 450

Dr. Walker

P3.16)

This way upon receiving each ACK the sender will send out the next packet and the sender will send each packet per receiver request which is expressed through the ACKs.

There will be one main problem. This way sender cannot use the existence of an ACK as a confirmation of the received packet and some other approach need to be considered to track lost data packets.

P3.39)

If the arrival rate goes beyond R/2 then the number of packets queued will go beyond the capacity and the loss rate will increase. The dropped packets will be queued for retransmission which in turn will use some of the transmission line capacity in future which again decrease the throughput. When the arrival rate equals R/2, 1 out of every three packets that leaves the queue is a retransmission. So pushing more through the transmission line (going beyond R/2) will not help with having a throughput more than λ_{out}.

The same reasoning holds for the case where we have lost packets. If the loss rate be half and the maximum rate be R/2 then λ_{out} will not go beyond R/4.

P3.45)

a) Loss rate by definition the ratio of the packets lost over the total number of packets sent. According to the problem in each cycle only one packet is lost so calculating the total number of packets leave the sender in each cycle will provide us with enough information to calculate the loss rate.

In the beginning of the Cycle the window size is W/2 uppon transmission of each window one packet will be added to the window size, till at window size W a packet will be lost and a new cycle begins by reducing the window size to W/2.

So the total number of sent packets in a cycle is:

$$\frac{W}{2}+\left(\frac{W}{2}+1\right)+\cdots+W=\sum_{n=0}^{W/2}(\frac{W}{2}+n)$$

$$=\left(\frac{W}{2}+1\right)\frac{W}{2}+\sum_{n=0}^{W/2}n$$

$$=\left(\frac{W}{2}+1\right)\frac{W}{2}+\frac{W/2(W/2+1)}{2}$$

$$=\frac{W^2}{4}+\frac{W}{2}+\frac{W^2}{8}+\frac{W}{4}$$

$$=\frac{3}{8}W^2+\frac{3}{4}W$$

And hence the loss rate is:

$$L=\frac{1}{\frac{3}{8}W^2+\frac{3}{4}W}$$

b)

For large enough W :

$$\frac{3}{8}W^2>>\frac{3}{4}W$$

And thus:

$$L\approx 8/3W^2 \text{ or } W\approx\sqrt{\frac{8}{3L}}$$

Using the text we will have following statement for the throughput:

$$\text{E(throughput)}=\frac{3}{4}\sqrt{\frac{8}{3L}}\cdot\frac{MSS}{RTT}$$

$$=\frac{1.22\cdot MSS}{RTT\cdot\sqrt{L}}$$

P3.46)

a) Let W denote the max window size measured in segments. Then, W*MSS/RTT = 10Mbps, as packets will be dropped if the maximum sending rate exceeds link capacity. Thus, we have W*1500*8/0.15=10*10^6, then W is about 125 segments.

b) As congestion window size varies from W/2 to W, then the average window size is 0.75W=94 (ceiling of 93.75) segments. Average throughput is 94*1500*8/0.15 =7.52Mbps.

c) 94/2 *0.15 = 7.05 seconds, as the number of RTTs (that this TCP connections needs in order to increase its window size from W/2 to W) is given by W/2. Recall the window size increases by one in each RTT.

P3.47)

Let W denote max window size. Let S denote the buffer size. For simplicity, suppose TCP sender sends data packets in a round by round fashion, with each round corresponding to a RTT. If the window size reaches W, then a loss occurs. Then the sender will cut its congestion window size by half, and waits for the ACKs for W/2 outstanding packets before it starts sending data segments again. In order to make sure the link always busying sending data, we need to let the link busy sending data in the period $W/(2*C)$ (this is the time interval where the sender is waiting for the ACKs for the W/2 outstanding packets). Thus, S/C must be no less than $W/(2*C)$, that is, $S>=W/2$.

Let Tp denote the one-way propagation delay between the sender and the receiver. When the window size reaches the minimum W/2 and the buffer is empty, we need to make sure the link is also busy sending data. Thus, we must have W/2/(2Tp)>=C, thus, $W/2>=C*2T$p. Thus:

$$S>=C*2T\text{p}.$$

P3.50)

a) The key difference between C1 and C2 is that C1's RTT is only half of that of C2. Thus C1 adjusts its window size after 50 msec, but C2 adjusts its window size after 100 msec. Assume that whenever a loss event happens, C1 receives it after 50msec and C2 receives it after 100msec. We further have the following simplified model of TCP. After each RTT, a connection determines if it should increase window size or not. For C1, we compute the average total sending rate in the link in the previous 50 msec. If that rate exceeds the link capacity, then we assume that C1 detects loss and reduces its window size. But for C2, we compute the average total sending rate in the link in the previous 100msec. If that rate exceeds the link capacity, then we assume that C2 detects loss and reduces its window size. Note that it is possible that the

average sending rate in last 50msec is higher than the link capacity, but the average sending rate in last 100msec is smaller than or equal to the link capacity, then in this case, we assume that C1 will experience loss event but C2 will not.

The following table describes the evolution of window sizes and sending rates based on the above assumptions.

	C1		C2	
Time (msec)	Window Size (num. of segments sent in next 50msec)	Average data sending rate (segments per second, =Window/0.05)	Window Size(num. of segments sent in next 100msec)	**Average data sending rate (segments per second, =Window/0.1)**
0	10	200 (in [0-50]msec]	10	**100 (in [0-50]msec)**
50	5 (decreases window size as the avg. total sending rate to the link in **last 50msec** is 300= 200+100)	100 (in [50-100]msec]		**100 (in [50-100]msec)**
100	2 (decreases window size as the avg. total sending rate to the link in **last 50msec** is 200= 100+100)	40	5 (decreases window size as the avg. total sending rate to the link in **last 100msec** is 250= (200+100)/2 + (100+100)/2)	**50**
150	1 (decreases window size as the avg. total sending rate to the link in last 50msec is 90= (40+50)	20		**50**
200	1 (no further decrease, as window size is already 1)	20	2 (decreases window size as the avg. total sending rate to the link in **last 100msec** is 80=	**20**

			(40+20)/2 + (50+50)/2)	
250	1 (no further decrease, as window size is already 1)	20		**20**
300	1 (no further decrease, as window size is already 1)	20	1 (decreases window size as the avg. total sending rate to the link in **last 100msec** is 40= (20+20)/2 + (20+20)/2)	**10**
350	2	40		**10**
400	1	20	1	**10**
450	2	40		**10**
500	1 (decreases window size as the avg. total sending rate to the link in last 50msec is 50= (40+10)	20	1	**10**
550	2	40		**10**
600	1	20	1	**10**
650	2	40		**10**
700	1	20	1	**10**
750	2	40		**10**
800	1	20	1	**10**
850	2	40		**10**
900	1	20	1	**10**
950	2	40		**10**
1000	**1**	**20**	**1**	**10**

Based on the above table, we find that after 1000 msec, C1's and C2's window sizes are 1 segment each.

b) No. In the long run, C1's bandwidth share is roughly twice as that of C2's, because C1 has shorter RTT, only half of that of C2, so C1 can adjust its window size twice as fast as C2. If we look at the above table, we can see a cycle every 200msec, e.g. from 850msec to 1000msec, inclusive. Within a cycle, the sending rate of C1 is

(40+20+40+20) = 120, which is thrice as large as the sending of C2 given by (10+10+10+10) = 40.

P3.51)

a) Similarly as in last problem, we can compute their window sizes over time in the following table. Both C1 and C2 have the same window size 2 after 2200msec.

	C1		C2	
Time (msec)	Window Size (num. of segments sent in next 100msec)	Data sending speed (segments per second, =Window/0.1)	Window Size(num. of segments sent in next 100msec)	**Data sending speed (segments per second, =Window/0.1)**
0	15	150 (in [0-100]msec]	10	**100 (in [0-100]msec)**
100	7	70	5	**50**
200	3	30	2	**20**
300	1	10	1	**10**
400	2	20	2	**20**
500	1	10	1	**10**
600	2	20	2	**20**
700	1	10	1	**10**
800	2	20	2	**20**
900	1	10	1	**10**
1000	2	20	2	**20**
1100	1	10	1	**10**
1200	2	20	2	**20**
1300	1	10	1	**10**
1400	2	20	2	**20**
1500	1	10	1	**10**
1600	2	20	2	**20**
1700	1	10	1	**10**
1800	2	20	2	**20**
1900	1	10	1	**10**
2000	2	20	2	**20**
2100	1	10	1	**10**
2200	**2**	**20**	**2**	**20**

b) Yes, this is due to the AIMD algorithm of TCP and that both connections have the same RTT.

c) Yes, this can be seen clearly from the above table. Their max window size is 2.
d) No, this synchronization won't help to improve link utilization, as these two connections act as a single connection oscillating between min and max window size. Thus, the link is not fully utilized (recall we assume this link has no buffer). One possible way to break the synchronization is to add a finite buffer to the link and randomly drop packets in the buffer before buffer overflow. This will cause different connections cut their window sizes at different times. There are many AQM (Active Queue Management) techniques to do that, such as RED (Random Early Detect), PI (Proportional and Integral AQM), AVQ (Adaptive Virtual Queue), and REM (Random Exponential Marking), etc.