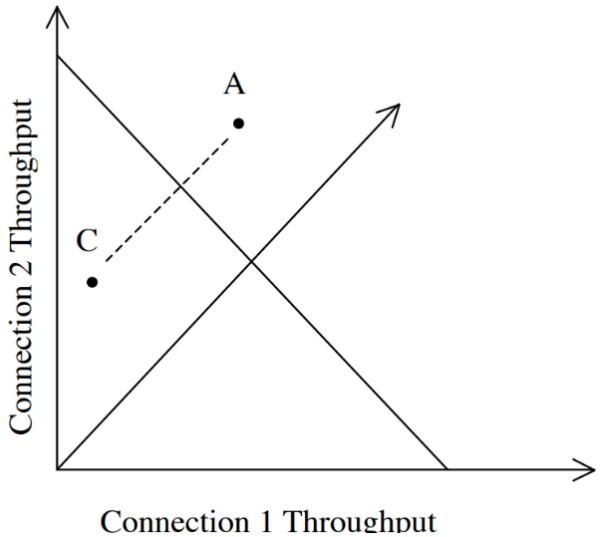
P41. Refer to Figure 3.55, which illustrates the convergence of TCP's AIMD algorithm. Suppose that instead of a multiplicative decrease, TCP decreased the window size by a constant amount. Would the resulting AIAD algorithm converge to an equal share algorithm? Justify your answer using a diagram similar to Figure 3.55.

Refer to figure below. Point A is unstable resulting in packet loss. In this case, both the flows will additively decrease their window resulting in movement along the 45- degree line (dotted line in above figure) to point A. Eventually when they reach point C, the sum of the throughputs will be less than the channel capacity and so the two flows will again additively increase their windows resulting in increase in throughput along the same dotted line. Again, this oscillation along the dotted line will continue. Hence fairness will not be achieved and connection 2 will receive an unfairly large share of the link bandwidth. Hence AIAD is not used by TCP.



P53. Consider the network described in the previous problem. Now suppose that the two TCP connections, C1 and C2, have the same RTT of 100 msec. Suppose that at time t0, C1's congestion window size is 15 segments but C2's congestion window size is 10 segments.
a. What are their congestion window sizes after 2200 msec?
b. In the long run, will these two connections get about the same share of the bandwidth of the congested link?
c. We say that two connections are synchronized, if both connections reach their maximum window sizes at the same time and reach their minimum window sizes at the same time. In the long run, will these two connections get synchronized eventually? If so, what are their maximum window sizes?
d. Will this synchronization help to improve the utilization of the shared link? Why? Sketch some idea to break this synchronization.

a)

Time is sec	Window size=No.of segments sent in next 100msec	Data sending speed=segments per second(window size/0.1)	Window size=No.of segments sent in next 100msec	Data sending speed=segments per second
0	15	150	10	100
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, they will in the long term.

c) Yes, they reach the network bandwidth at the same time and then drop at the same time.

d) If one person is slightly offput so that one backs off more and one backs off less, then the average use of the bandwidth will be higher because one is trying to send min and one is trying to send max

P56. In our discussion of TCP congestion control in Section 3.7, we implicitly assumed that the TCP sender always had data to send. Consider now the case that the TCP sender sends a large amount of data and then goes idle (since it has no more data to send) at t 1. TCP remains idle for a relatively long period of time and then wants to send more data at t 2. What are the advantages and disadvantages of having TCP use the cwnd and ssthresh values from t 1 when starting to send data at t 2? What alternative would you recommend? Why?

Advantages: If the network was NOT caught in congestion between t1 and t2 then the user is able to restore the same performance of the network at t1

Disadvantages: If the network was caught in congestion between t1 and t2, the sender will be presented

with two scenarios. Scenario one is to wait for three duplicate ACKs (Reno) and go into Fast Recovery (ssthresh= cwnd/2, cwnd = ssthresh + 3 MSS). Scenario two is to timeout and go into slow-start (ssthresh =cwnd/2, cwnd = 1 MSS). The problem with both of these scenarios is that they are still based on cwnd from t1 which is not consistent with the changes that may have occurred between t1 and t2.

I recommend to use Fast Recovery at t2, that is to send three packets and wait for three-duplicate ACKs. When the ACKs are received, proceed by setting ssthresh = cwnd/2 and cwnd = ssthresh + 3 MSS. This solution gurantees average performance regardless of congestion between t1 and t2.

P57. In this problem, we investigate whether either UDP or TCP provides a degree of end-point authentication.
a. Consider a server that receives a request within a UDP packet and responds to that request within a UDP packet (for example, as done by a DNS server). If a client with IP address X spoofs its address with address Y, where will the server send its response?
b. Suppose a server receives a SYN with IP source address Y, and after responding with a SYNACK, receives an ACK with IP source address Y with the correct acknowledgment number. Assuming the server chooses a random initial sequence number and there is no "man-in-the-middle," can the server be certain that the client is indeed at Y (and not at some other address X that is spoofing Y)?

a)
Let UDP packet is received a request to the server. This request accepts the server.
So, the IP address X is deceived with address Y and response to the address Y.
So, address Y is the matching IP address.
b)
The client address is 'Y'
The SYNACK will be send with Y's address.
TCP in the host will not send back the TCP ACK segment.
It is not possible to the attacker to send the correct sequence number.
The attacker fails even if he sends an properly timed TCP ACK segment.