Discussion #14
EE450

Sample Problems
-TCP Congestion Control

Review: TCP Congestion Control

- MaxWindow = MIN (CongestionWindow, AdvertisedWindow)
- EffectiveWindow = MaxWindow – (LastByteSent-LastByteAcked)
- Increment = MSS x (MSS / CongestionWindow)
- CongestionWindow + = Increment

Problem #4

- Consider a simple congestion-control algorithm that uses linear increase and multiplicative decrease but not slowstart, that works in units of packets rather than bytes, and that starts each connection with a congestion windows equal to one packet.
- Give a detailed sketch of this algorithm. Assume the delay is latency only, and that when a group of packets is sent, only a single ACK is returned. Plot the congestion window as a function of round-trip times for the situation in which the following packets are lost: 9, 25, 30, 38 and 50. For simplicity, assume a perfect timeout mechanism that detects a lost packet exactly 1 RTT after it is transmitted. **"Go-back-N"**

Solution

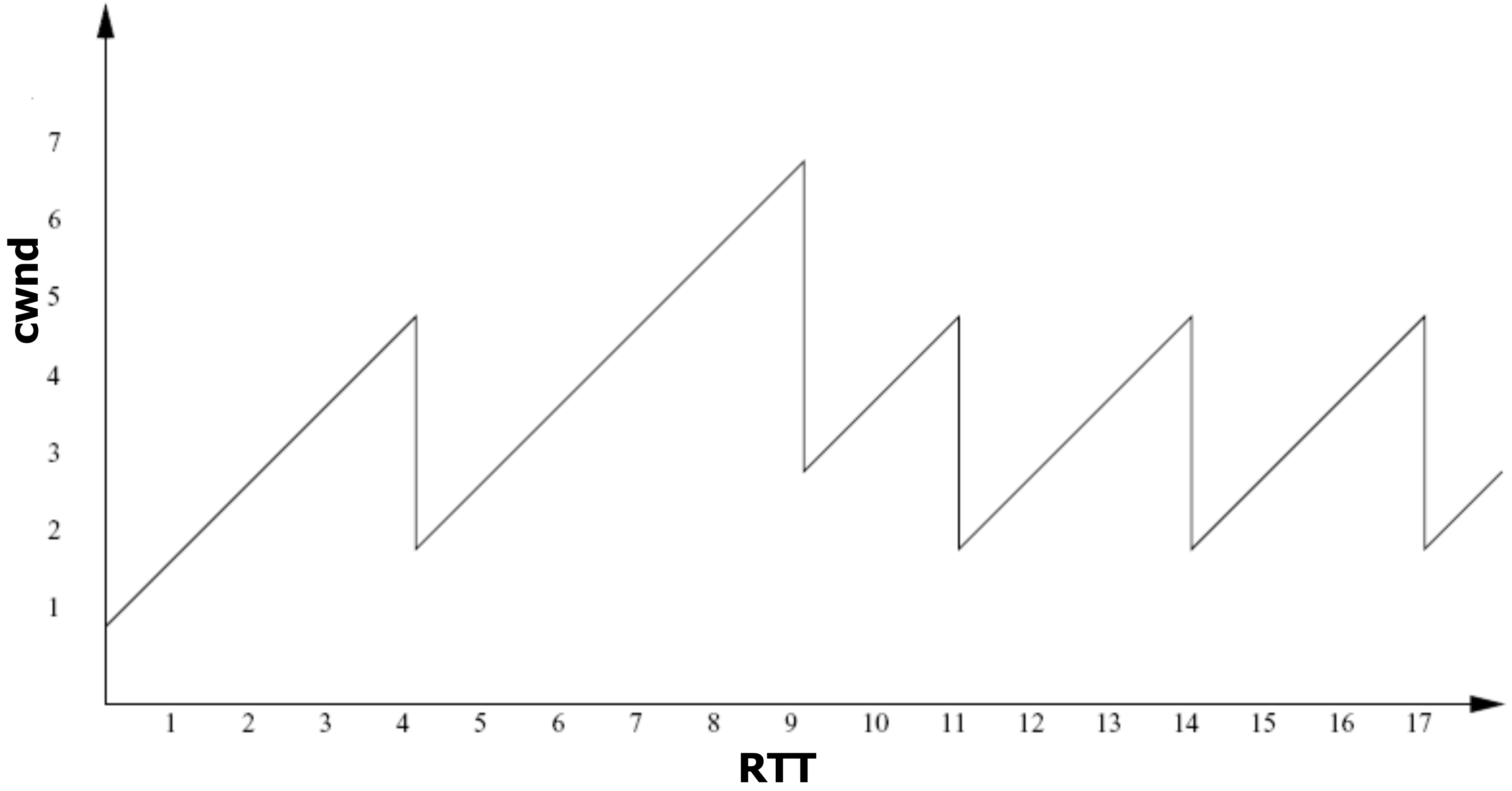
The window size is initially 1; when we get the first ACK it increases to 2. At the beginning Of the second RTT we send packets 2 and 3. When we get their ACKs we increase the window size to 3 and send packets 4, 5 and 6. When these ACKs arrive the window size becomes 4. Now, at the beginning of the 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.

Solution

The plot below shows the window size vs. RTT.



Problem#5: Description

- You are hired to design a reliable byte-stream protocol that uses a sliding window (like TCP).
- This protocol will run over a 1-Gbps network.
- The RTT of the network is 140 ms, and the maximum segment lifetime (MSL) is 60 seconds.
- How many bits would you include in the advertisedWindow and SequenceNum fields of your protocol header?

Problem#5: Solution

- The advertised window should be large enough to keep the pipe full:
  - Delay (RTT) x BW = 140 ms x 1 Gbps

    = 140 Mb = 17.5 MB of data.
  - This amount of data , i.e. 17.5 MB < 2^5 x 2^{20} bytes of data requires 25 bits (2^{25} = 33,554,432) for the AdvertisedWindow field .
- The sequence number field must not wrap around in the maximum segment lifetime, i.e. in 60 seconds.
- 60 sec x 1 Gbps/ 8 bits = 7.5 GB < 2^3 x 2^{30} bytes can be transmitted in one MSL.
- 33 bits allows a sequence space of 8.6 GB, and so will not wrap in 60 second.

Problem#6: Description

- Suppose TCP operates over a 1-Gbps link.
  - Assuming TCP could utilize the full bandwidth continuously, how long would it take the sequence number to wrap around completely?
  - Suppose an added 32-bit timestamp field increments 1000 times during the wrap-around time you found above.
  - How long would it take for the timestamp to wrap around?

Problem#6, Solution

a) 1 Gb/sec = 125 MB/sec, so the sequence numbers wrap around when we send 2^{32} B = 4 GB. This would take 4GB/ (125 MB/sec) = 32 seconds.

b) Increments 1000 times during 32 seconds means incrementing once every 32 msec, it would take about 32 x 4 x 10^9 msec, or about four years for the timestamp field to wrap.

Problem#7: Description

- Assume that TCP implements an extension that allows window sizes much larger than 64 KB. Suppose that you are using this extended TCP over a 1Gbps link with a latency of 100 ms to transfer a 10-MB file. If TCP send 1KB packets (assuming no congestion and no lost packets):

  (a) How many RTTs does it take until slowstart opens the send window to 1MB?

  (b) How many RTTs does it take to send the file?

  (c) If the time to send the file is given by the number of required RTTs multiplied by the link latency, what is the effective throughput for the transfer? What percentage of the link bandwidth is utilized?

Problem#7: Solution, Part(a)

(a) In slowstart, 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 = 1 MB.

In other words, to reach a window size of 1MB, it will take:

$$[Log_2(\frac{SendWindow}{SegmentSize})]RTT = [Log_2(\frac{1MB}{1KB})]RTT = [Log_2 1024]RTT = 10RTT$$

Problem#7: Solution

i	Time slot	Amount of Data sent	Send window in segments At **the end** of the ith RTT	Send window in bytes At **the end** of the ith RTT
1	1st RTT	2^0 x 1KB	$2^1 = 2$	2^1 x 1KB
2	2nd RTT	2^1 x 1KB	$2^2 = 4$	2^2 x 1KB
3	3rd RTT	2^2 x 1KB	$2^3 = 8$	2^3 x 1KB
4	4th RTT	2^3 x 1KB	$2^4 = 16$	2^4 x 1KB
5	5th RTT	2^4 x 1KB	$2^5 = 32$	2^5 x 1KB
6	6th RTT	2^5 x 1KB	$2^6 = 64$	2^6 x 1KB
7	7th RTT	2^6 x 1KB	$2^7 = 128$	2^7 x 1KB
8	8th RTT	2^7 x 1KB	$2^8 = 256$	2^8 x 1KB
9	9th RTT	2^8 x 1KB	$2^9 = 512$	2^9 x 1KB
10	10th RTT	2^9 x 1KB	$2^{10} = 1024$	2^{10} x 1KB

Problem#7: Solution, Part(b)

(b) Amount of data sent during the first 10 RTTs:

$$(\sum_{i=1}^{10} 2^{i-1}) \times 1KB = (2^{10} - 1) \times 1KB = (1024 - 1) \times 1KB \simeq 1MB$$

So we need to send 9MB more to completely transfer the 10MB file. Since we have not reached the capacity of the network, the slowstart continues to double the window every RTT.

Problem#7: Solution

The amount of data transferred during each RTT is well below the capacity of the link in one RTT. Therefore the file is completely transferred in 14th RTTs.

i	Time slot	Amount of Data sent
11	11th RTT	2^{10} x 1KB=1MB
12	12th RTT	2^{11} x 1KB=2MB
13	13th RTT	2^{12} x 1KB=4MB
14	14th RTT	A partial window of 2MB data

Problem#7: Solution, Part(c)

(c) It takes 14 x RTTs = 14 x 100ms = 1.4 sec to send the file.

The effective throughput is:

(10MB/1.4 sec) = 7.1MBps = 57.1Mbps.

The utilization is:

(57.1Mbps/1Gbps) = 5.7% of the link bandwidth.

Problem#8: Description

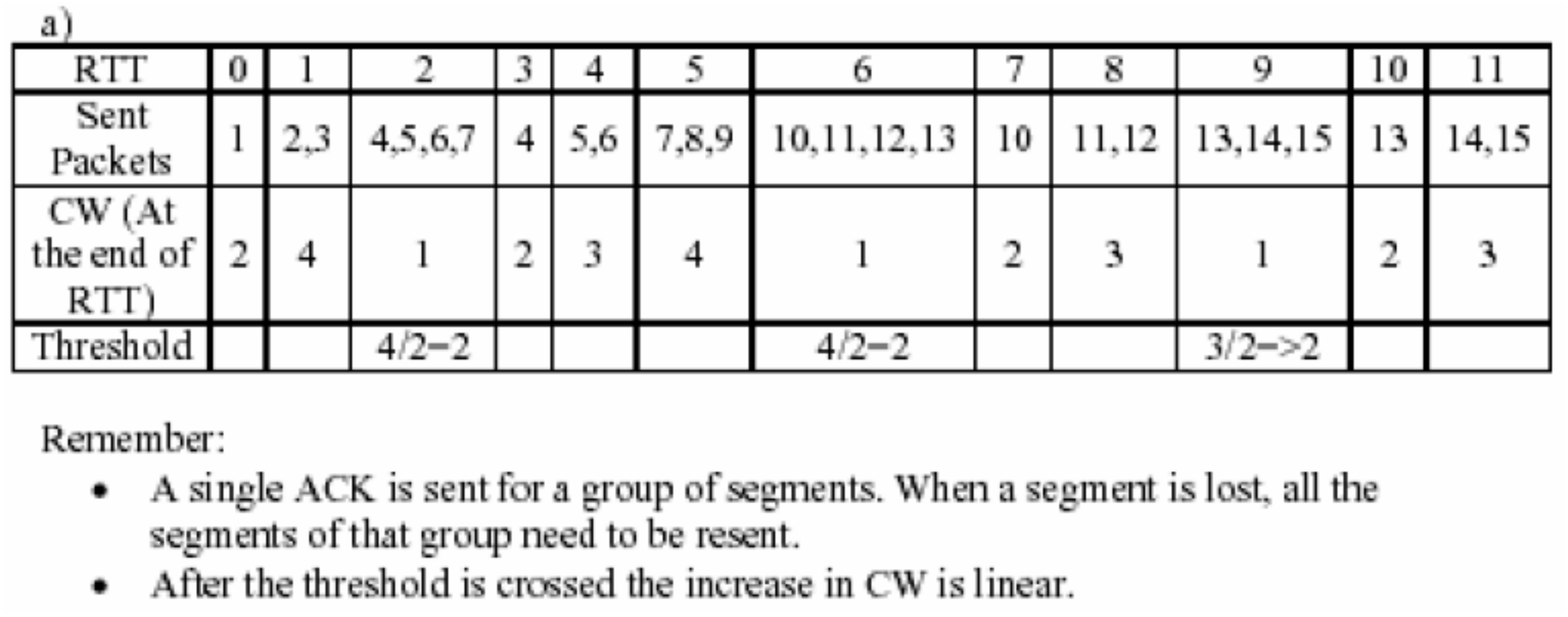
Consider the TCP congestion-control algorithm that uses slow-start but has no fast retransmit/fast recovery. The algorithm starts each connection with a window size equal to one segment. Assume the only delay is latency only, and a single ACK is sent for a group of segments. Assume a perfect timeout mechanism that detects a lost segment exactly 1 RTT after it is transmitted. Round up the fractional numbers encountered in window sizes to greater integers (like 2.5 -> 3)

a) Fill in the table below assuming segments 7, 12 and 15 are lost.

RTT	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Sent Segments										

b) Sketch the variation of congestion window with respected to time

Problem#8: Solution, Part(a)



a)

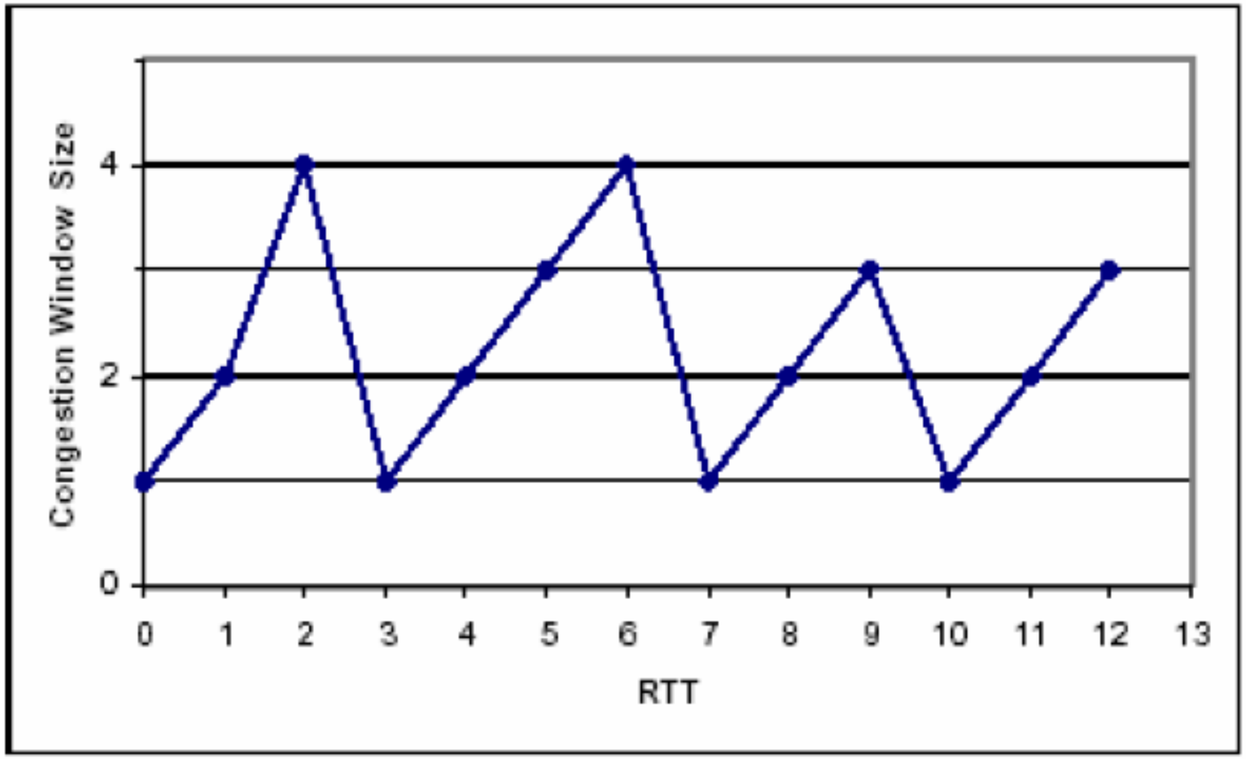
RTT	0	1	2	3	4	5	6	7	8	9	10	11
Sent Packets	1	2,3	4,5,6,7	4	5,6	7,8,9	10,11,12,13	10	11,12	13,14,15	13	14,15
CW (At the end of RTT)	2	4	1	2	3	4	1	2	3	1	2	3
Threshold			4/2=2				4/2=2			3/2->2		

Remember:

- A single ACK is sent for a group of segments. When a segment is lost, all the segments of that group need to be resent.
- After the threshold is crossed the increase in CW is linear.

Problem#8: Solution, Part(b)

b)
The graph is as follows:



Problem#9: Description

- Suppose a TCP connection, with window size 1, loses every other packet. Those that do arrive have RTT = 1 second.
- What happens? What happens to TimeOut? Do this for two cases:
  - After a packet is eventually received, we pick up where we left off, resuming with EstimatedRTT initialized to its pre-timeout value and Timeout double that.
  - After a packet is eventually received, we resume with Timeout initialized to the last exponentially backed-off value used for the timeout interval.

Problem#9: Solution, Part (a)

- If every other packet is lost, we transmit each packet twice.

  a) Let $E \geq 1$ be the value for EstimatedRTT and

  $T = 2\mathrm{x}E$ be the value for TimeOut.

  - We lose the first packet and back off TimeOut to $2\mathrm{x}T$.
  - Then when packet arrives, we resume with EstimateRTT = E and TimeOut = T.
  - In other words TimeOut doesn't change.

Problem#9: Solution, Part (b)

b) Let *T* be the value for TimeOut.

- When we transmit the packet the first time, it will be lost and we will wait time *T*.
- At this point we back off and retransmit using TimeOut=2 x *T*.
- The retransmission succeeds with an RTT of 1 sec, but we use the backed-off value of 2 x *T* for the next TimeOut.
- In other words, TimeOut doubles with each received packet. This is not Good.

Problem#10: Description

- Suppose TCP operates over a 40-Gbps link.
  - Assuming TCP could utilize the full bandwidth continuously, how long would it take the sequence number to wrap around completely?
  - Suppose an added 32-bit timestamp field increments 1000 times during the wrap-around time you found above.
  - How long would it take for the timestamp to wrap around?

Problem#10: Solution

a) 2^{32} bytes / (5 GBps) = 859 ms.

b) 1000 ticks in 859 ms is once each 859 microseconds, indicating wrap-around in 3.7 Msec or approx 43 days.