Excercise 1.1

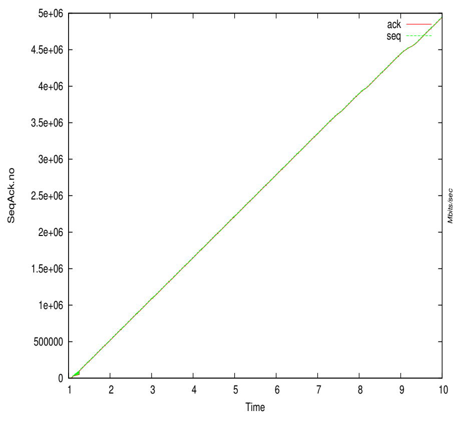


Figure 1: Time Seqeunce Graph

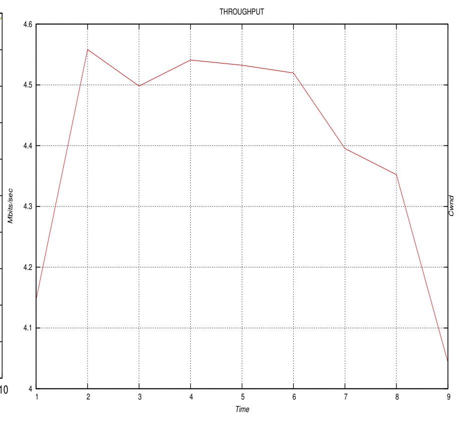


Figure 2: Throughput

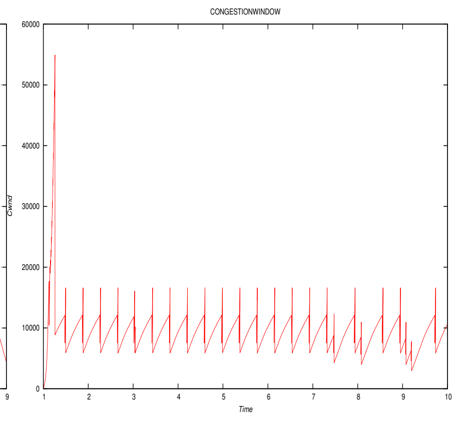


Figure 3: cwnd

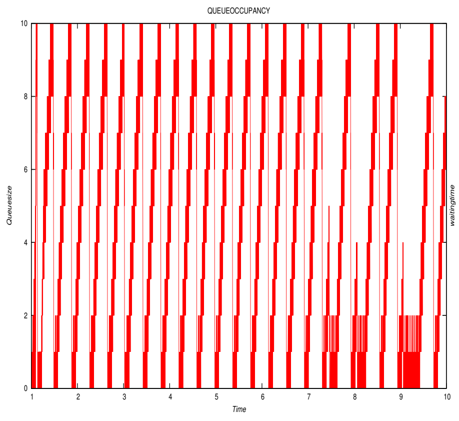


Figure 4: Queueoccupancy

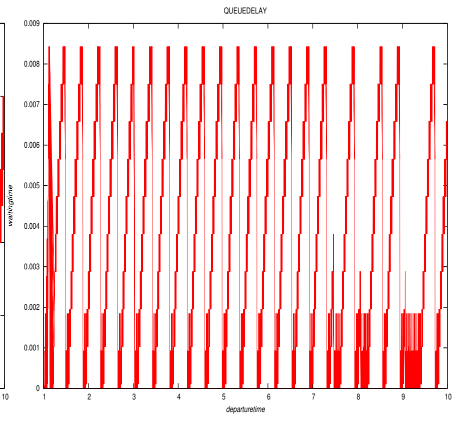


Figure 5: Waitingtime

- Here, we are using TCP new reno, it uses fast recovery when duplicate acknowledgements are received. We can see one big step and some small steps in sequence/ack no. graph. During that step, sender is receiving Dup. acks(same ack. nos.) from receiver. Accordingly, we can see that cwnd increases by 536Bytes(1 MSS) in Cwnd graph for each dupack. This is cwnd inflation. Once new ack. received, seq/ack no. increases normally in seq/ack no.graph. So, everytime when it completes fast recovery, it halves the congestion window.
- From throughput graph, Avg. throughput = 4.40 Mbits/sec Throughput is below bottleneck link (5 Mbps) because of acknowledgements, overhead transmitted with packets, and error rate of the channel.

Excercise 1.2

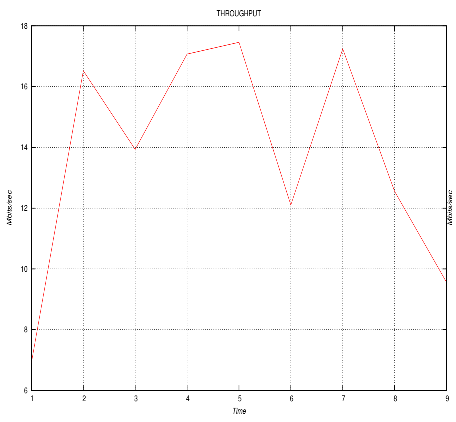


Figure 6: Average throughput for 5ms delay

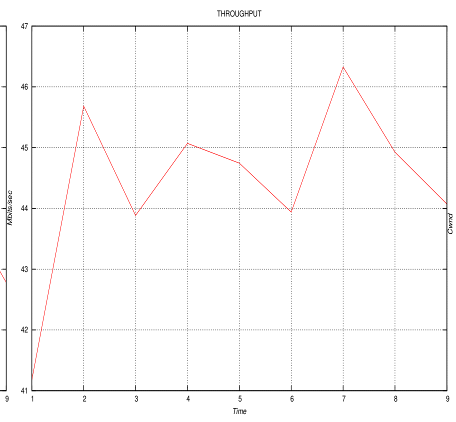


Figure 7: Average throughput for 1ms delay

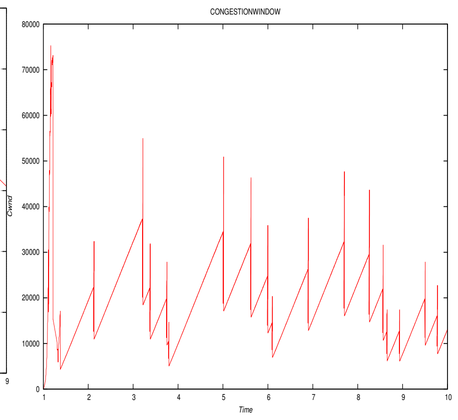


Figure 8: cwnd

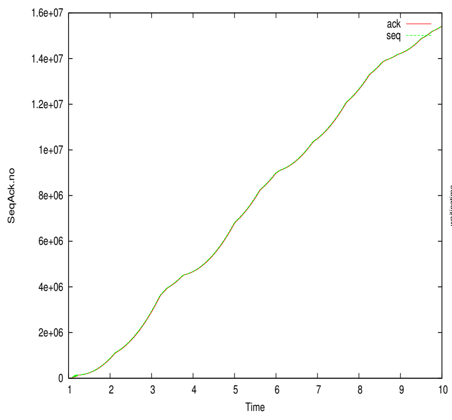


Figure 9: Time sequence graph

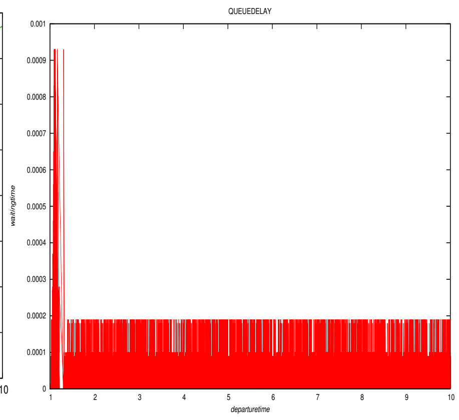


Figure 10: Waitingtime

- Average throughput = 13.70 Mbits/sec Which is way below then bottleneck link (50 Mbps). We know that to get idle value for throughput, sender's window size should be close to Bandwidh*Delay product. As sender's window size gets close to BDP throughput increases. Once it reaches BDP, from there on throughput won't increase anymore. But if sender's window is much lesser than BDP then throughput will be less as channel has more capacity to carry then what sender is sending.
- In our example, BW=50Mbps, delay=5ms =¿ BDP= 250000bits = 31250Bytes = (31250/536) = 59MSS But, here sender is sending much lesser load then BDP, so not enough packets to fill the channel. We can observe this from queue occupancy graph. After slowstart, queue occupancy never increases beyond 2.
- If we make Delay=1ms, then BDP= 50000bits = 6250Bytes = 12MSS Now, we made BDP close to sender's window size. So that channel will get filled with what sender in sending. Hence, throughput increases drastically. Here, Average throughput = 44.92 Mbits/sec, which is very much close to 50Mbps.

Excercise 1.3

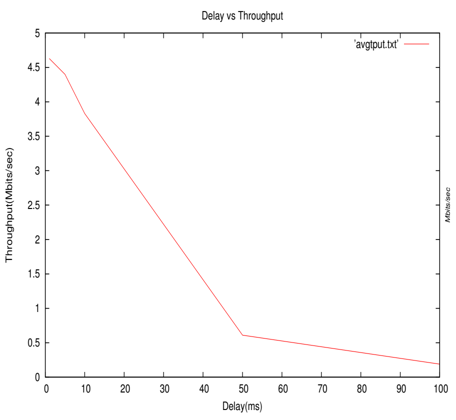


Figure 11: Delay vs Avg throughput

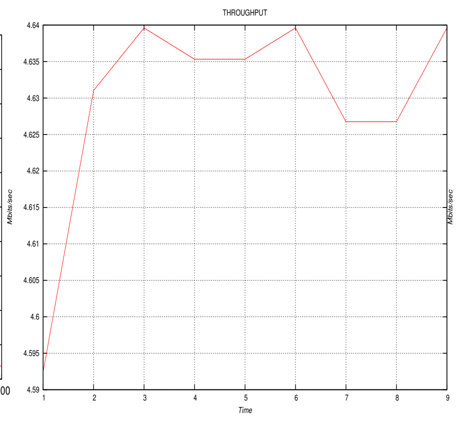


Figure 12: Throughput for delay 1ms

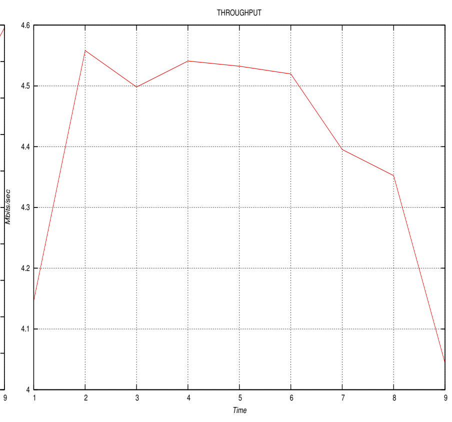


Figure 13: Throughput for delay 5ms

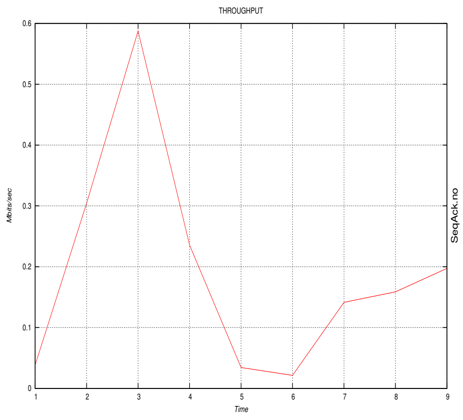


Figure 14: Throughput for delay 100ms

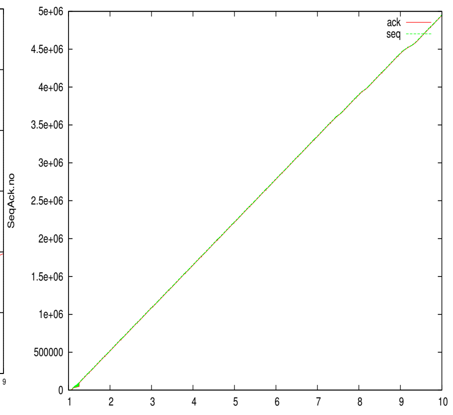


Figure 15: Time sequence graph

- Throughput of tcp decreases as delay increases.
- As we can see from graph of delay vs avgtput, when delay is below 5ms throughput is in range of 4.40 - 4.70 Mbits/sec, very close to link rate. But as we increase delay throughput falls as much as close to 0 Mbits/sec for delays above 50ms.
- Reason for this is, as link delay increases data send by sender in RTT decreases. Because RTT increases. and as we know, Throughout is inversly proportional to RTT.
- We can observe from tcp congestion window graph that as we increase the delay, gap between two cwnd updation increases. And as per delay, queue size should also be increased to match BDP, ideally equals to BDP. Otherwise, packets will get dropped from queue due to congestion and sender will receive more and more dup. acks for those packets.We can see that from seq/ack no. graph.

Excercise 1.4

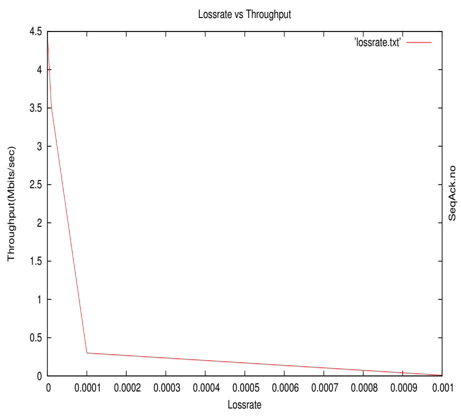


Figure 16: Lossrate vs Throughput

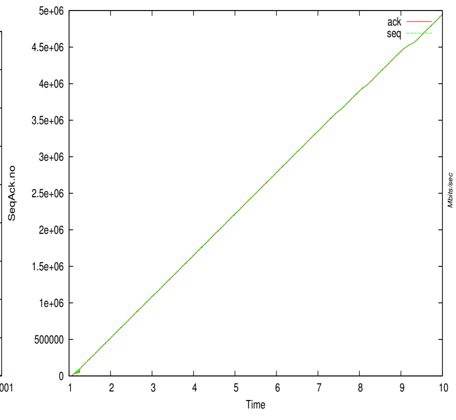


Figure 17: Time Sequence graph

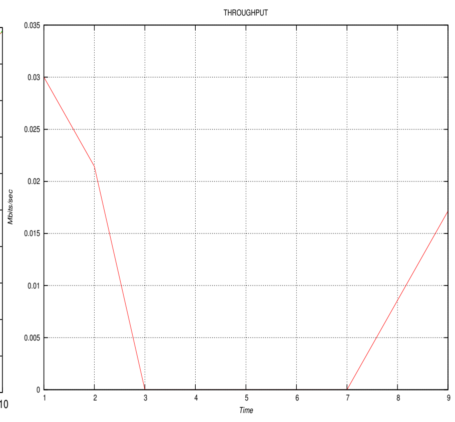


Figure 18: Throughput for loss rate 0.001

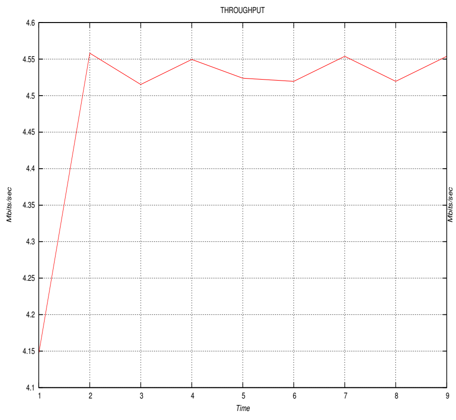


Figure 19: Throughput for loss rate 0.0000001

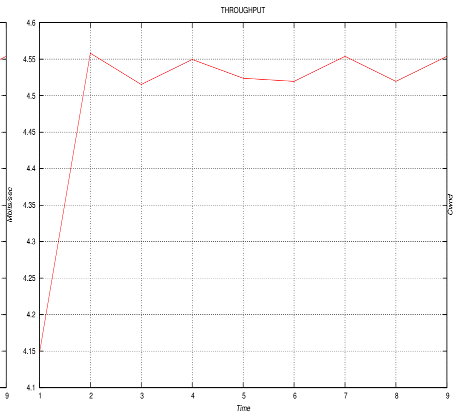


Figure 20: Throughput for loss rate 0.00000001

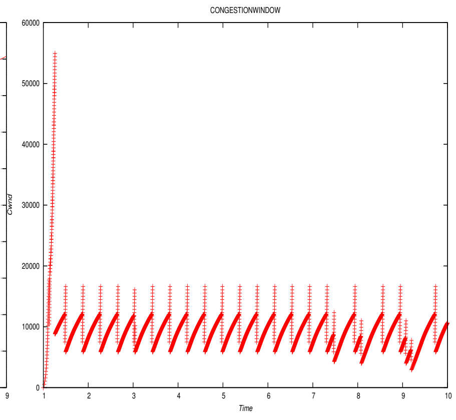


Figure 21: CWND when delay is 0.000001

- We can observe that Tcp throughput decreases as packet loss rate in creases. It is quiet clear from the graph as well as from formula for average throughput where average throughput is inversly proportional to square root(packet loss).
- From lossrate vs throughput graph, when packet loss rate is 0.000001, throughput is 4.40 Mbits/sec. When lossrate is less than that throughput will still be around 4.50 Mbits/sec. That is because, throughput can never be more than bottleneck link bandwidth (5Mbps), whatever packet lossrate is there.
- One more thing to observe is, as packet loss rate increases more and more dup. acks, retransmissions are needed. We can see that fron sequence no. /ack no. graph. And tcp will take more time in fast recovery phase. Because of that cwnd is also updated very less no. of times, which we can see from tcp cwnd graph.

Excercise 1.5

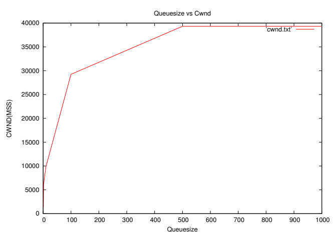


Figure 22: Queuesize vs Avg CWND

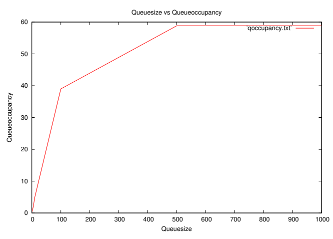


Figure 23: Queuesize vs Avg Queue Occupancy

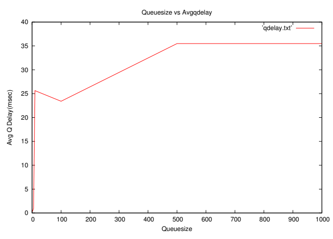


Figure 24: Queuesize vs Avg delay

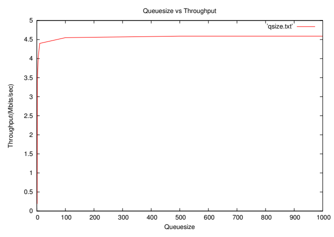


Figure 25: Queue Size vs Avg Throughput

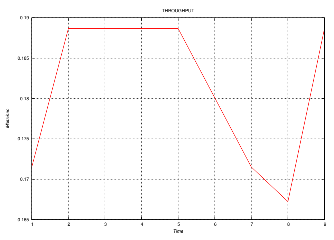


Figure 26: Throughput for Queuesize 1

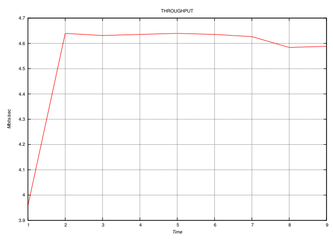


Figure 27: Throughput for Queuesize 100

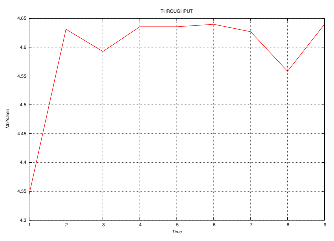


Figure 28: Throughput for Queuesize 500

Queue Length	Avg Throughput	Avg. Queueing Delay	Avg. cwnd	Avg Queue Occupancy
1	0.19	0.45	0.256	1347.69
2	3.45	0.78	0.410	6140.70
5	4.03	1.99	0.947	7650.70
10	4.40	4.84	25.63	9842.91
100	4.55	39.0	23.40	29246.29
500	4.59	58.83	35.49	39336.18
1000	4.59	58.83	35.49	39336.18

- We can observe from various graphs that as queuesize increases average queueing delay, avg queueoccupancy, avg cwnd, avg throughput increases. But after some threshold value, all the thing will be constant with more increase in queue size. For ex, throughput can be maximum upto BDP. It can't go beyond that. Like this other metrics will also be constant after some threshold value.
- Here, suitable queuesize is more than 6. After a certain value like after 10, average throughput will be almost constant. Ideally, our BDP is 6 MSS. So, once queuesize increases beyond that throughput will not change much. So, queue size should be in the range 6 - 10 (atmost 100) as per graphs we obtained.

Excercise 1.6

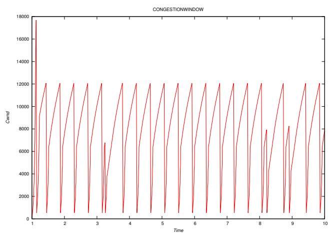


Figure 29: CWND for TCP TAHOE

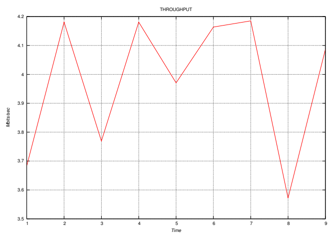


Figure 30: Avg Throughput for TCP TAHOE

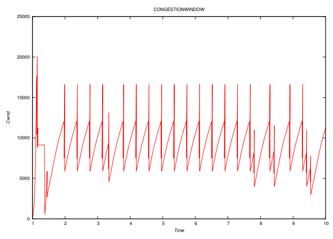


Figure 31: CWND for TCP RENO

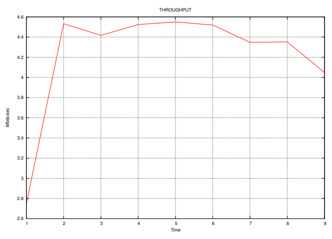


Figure 32: Avg Throughput for TCP RENO

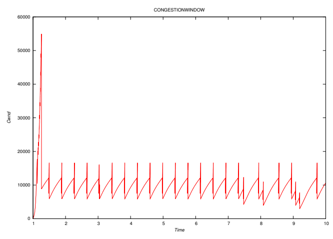


Figure 33: CWND for TCP NEW RENO

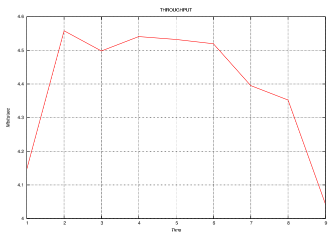


Figure 34: Avg Throughput for TCP NEW RENO

- We can compare throughput for different versions of TCP from average throughput graph. TCP tahoe has the least throughput among all the variants. It has no fast recovery mechanism. We can see that from cwnd graph. So, when packet is lost sender will wait for time out for retransmission. Thus during that time pipeline will be dry and overall throughput decreases. TCP reno has almost same throughput as TCP new reno as it implements fast recovery mechanism, we can see that in cwnd graph. Still TCP new reno has little edge as it can recover from multiple packet loss.
- In TCP tahoe's cwnd graph, cwnd starts from an initial value, then it increases exponentially (slow start) –¿ when it reaches to SSThreshold it increases additively (congestion avoidance) –¿ when time out occurs, cwnd falls to initial value and Tahoe will again start slowstart phase with SSThreshold = cwnd/2. It has no fast recovery mechanism.
- In TCP reno's cwnd graph, It is almost same as TCP new reno except in slow start phase cwnd of tcp new reno goes much higher than cwnd of reno. TCP reno also has additional fast recovery + fast retransmit mechanism than TCP tahoe. So, we can observe cwnd inflation in cwnd graph, during begining of fast recovery phase.

Excercise 2.1

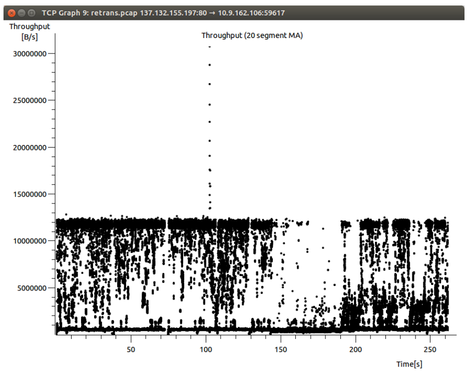
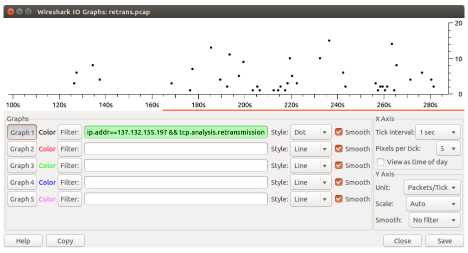


Figure 35: Avg Throughput for entire transfer

1. Average TCP throughput of entire transfer = 6.5 Mbit/sec.



Graphs

Graph 1	Color	Filter:	ip.addr==137.132.155.197 && tcp.analysis.retransmission	Style:	Dot	✓ Smooth
Graph 2	Color	Filter:		Style:	Line	✓ Smooth
Graph 3	Color	Filter:		Style:	Line	✓ Smooth
Graph 4	Color	Filter:		Style:	Line	✓ Smooth
Graph 5	Color	Filter:		Style:	Line	✓ Smooth

X Axis
Tick interval: 1 sec
Pixels per tick: 5
☐ View as time of day

Y Axis
Unit: Packets/Tick
Scale: Auto
Smooth: No filter

Figure 36: TCP retransmission rate

2. No. of retransmissions due to duplicate acknowledgements = 984
   No. of retransmissions due to retransmits = 189

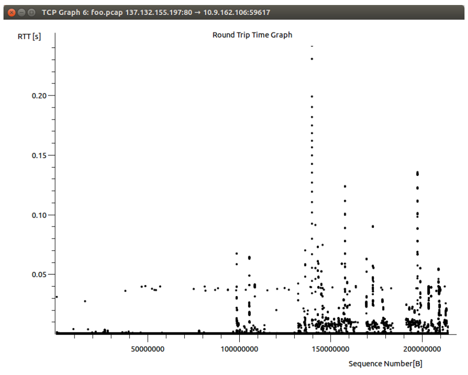


Figure 37: TCP RTT

3. Minimum RTT = 250ms
   Maximum RTT = 1us
   Queueing delay plays major part in RTT. Queueing delay is random for packet transmission every time where else transmission delay and propogation delay don't change much in every RTT.

Excercise 2.2

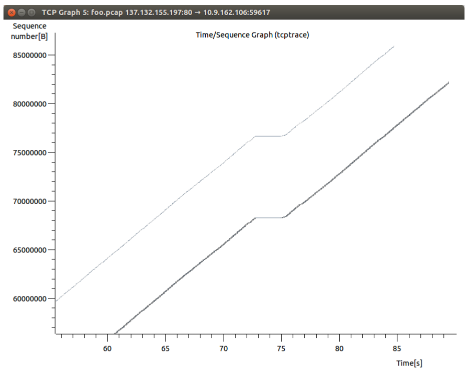
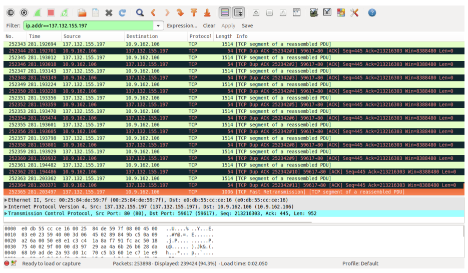


Figure 38: Time sequence graph

We can see from below screen shot that tcp uses Fast recovery mechanism, as it send a packet for every dup ack received. So, clearly sender follows the TCP Reno style. And I have not observed any sack based recovery for this transfer.



```
No.    Time       Source           Destination      Protocol Length Info
252343 281.192694 137.132.155.197  10.9.162.106     TCP      1514 [TCP segment of a reassembled PDU]
252344 281.192701 10.9.162.106     137.132.155.197  TCP        54 [TCP Dup ACK 252342#1] 59617→80 [ACK] Seq=445 Ack=213216303 Win=8388480 Len=0
252345 281.193012 137.132.155.197  10.9.162.106     TCP      1514 [TCP segment of a reassembled PDU]
252346 281.193018 10.9.162.106     137.132.155.197  TCP        54 [TCP Dup ACK 252342#2] 59617→80 [ACK] Seq=445 Ack=213216303 Win=8388480 Len=0
252347 281.193143 137.132.155.197  10.9.162.106     TCP      1514 [TCP segment of a reassembled PDU]
252348 281.193149 10.9.162.106     137.132.155.197  TCP        54 [TCP Dup ACK 252342#3] 59617→80 [ACK] Seq=445 Ack=213216303 Win=8388480 Len=0
252349 281.193224 137.132.155.197  10.9.162.106     TCP      1514 [TCP segment of a reassembled PDU]
252350 281.193228 10.9.162.106     137.132.155.197  TCP        54 [TCP Dup ACK 252342#4] 59617→80 [ACK] Seq=445 Ack=213216303 Win=8388480 Len=0
252351 281.193356 137.132.155.197  10.9.162.106     TCP      1514 [TCP segment of a reassembled PDU]
252352 281.193359 10.9.162.106     137.132.155.197  TCP        54 [TCP Dup ACK 252342#5] 59617→80 [ACK] Seq=445 Ack=213216303 Win=8388480 Len=0
252353 281.193470 137.132.155.197  10.9.162.106     TCP      1514 [TCP segment of a reassembled PDU]
252354 281.193474 10.9.162.106     137.132.155.197  TCP        54 [TCP Dup ACK 252342#6] 59617→80 [ACK] Seq=445 Ack=213216303 Win=8388480 Len=0
252355 281.193601 137.132.155.197  10.9.162.106     TCP      1514 [TCP segment of a reassembled PDU]
252356 281.193605 10.9.162.106     137.132.155.197  TCP        54 [TCP Dup ACK 252342#7] 59617→80 [ACK] Seq=445 Ack=213216303 Win=8388480 Len=0
252357 281.193798 137.132.155.197  10.9.162.106     TCP      1514 [TCP segment of a reassembled PDU]
252358 281.193801 10.9.162.106     137.132.155.197  TCP        54 [TCP Dup ACK 252342#8] 59617→80 [ACK] Seq=445 Ack=213216303 Win=8388480 Len=0
252359 281.193929 137.132.155.197  10.9.162.106     TCP      1514 [TCP segment of a reassembled PDU]
252360 281.193932 10.9.162.106     137.132.155.197  TCP        54 [TCP Dup ACK 252342#9] 59617→80 [ACK] Seq=445 Ack=213216303 Win=8388480 Len=0
252361 281.194482 137.132.155.197  10.9.162.106     TCP      1514 [TCP segment of a reassembled PDU]
252362 281.194486 10.9.162.106     137.132.155.197  TCP        54 [TCP Dup ACK 252342#10] 59617→80 [ACK] Seq=445 Ack=213216303 Win=8388480 Len=0
252363 281.203362 137.132.155.197  10.9.162.106     TCP      1514 [TCP segment of a reassembled PDU]
252364 281.203371 10.9.162.106     137.132.155.197  TCP        54 [TCP Dup ACK 252342#11] 59617→80 [ACK] Seq=445 Ack=213216303 Win=8388480 Len=0
252365 281.203497 137.132.155.197  10.9.162.106     TCP      1006 [TCP Fast Retransmission] [TCP segment of a reassembled PDU]
▶Ethernet II, Src: 00:25:84:de:59:7f (00:25:84:de:59:7f), Dst: e0:db:55:cc:ce:16 (e0:db:55:cc:ce:16)
▶Internet Protocol Version 4, Src: 137.132.155.197 (137.132.155.197), Dst: 10.9.162.106 (10.9.162.106)
▶Transmission Control Protocol, Src Port: 80 (80), Dst Port: 59617 (59617), Seq: 213216303, Ack: 445, Len: 952

0000  e0 db 55 cc ce 16 00 25  84 de 59 7f 08 00 45 00   ..U....% ..Y...E.
0010  03 e0 23 59 40 00 3d 06  45 02 89 84 9b c5 0a 09   ..#Y@.=. E.......
0020  a2 6a 00 50 e8 e1 c3 c4  1a 8a f7 91 fc ac 50 18   .j.P.... ......P.
0030  75 40 02 9f 00 00 d3 97  29 aa 4a 6b 26 b6 28 da   u@...... ).Jk&.(.
0040  68 b9 ad de 2a 93 d0 1c  70 c5 b3 60 1e c7 1e e9   h...*... p..`....

Ready to load or capture    Packets: 253898 · Displayed: 239424 (94.3%) · Load time: 0:02.050    Profile: Default
```

Figure 39: Fast Recovery