

CS 494/594 Homework 3 (Winter 2022)

Instructor: Dr. Nirupama Bulusu

Due Date: 2/22/2022 11:59 pm PST

For 594 students only: Transport Layer Reviews

Review **only one** of the following three papers:

- Dina Katabi, Mark Handley, and Charlie Rohrs, "Congestion control for high bandwidth-delay product networks", ACM SIGCOMM 2002.  
<http://conferences.sigcomm.org/sigcomm/2002/papers/xcp.pdf>
- Keith Winstein and Hari Balakrishnan, "TCP ex machina: computer-generated congestion control", ACM SIGCOMM 2013.  
<https://dl.acm.org/doi/10.1145/2486001.2486020>

### Paper Review

TCP ex-machina: Computer -generated congestion control

By – Parth Parashar

This paper tells us about the end-to-end congestion control while using multi-user network. The authors have developed an algorithm that generates congestion-control, the algorithm is called as Remy which runs at the endpoints. Congestion control is a fundamental problem in a multi-user computer networks, now the question that addresses: When should an endpoint transmit each packet of data?. The answer would be to transfer the packet whenever there is capacity to carry the packets. The subnets and link layer are typically telling us how TCP performs over them. The TCP assumes that the packet losses are due to congestion and reduces its transmission rate in response.

The congestion control over heterogeneous packet-switched networks has been an active area of research for Ramakrishnan and Jain's DECBIT scheme and Jacobson's TCP Tahoe and as well as Reno algorithms. The end-to-end algorithms typically compute a congestion window as well as the round-trip time using. Due to the congestion inferred from the packet loss or, in some cases, rising delays, the sender reduces its window; conversely, when no congestion is perceived, the sender increases its window.

The different ways to change the window size is by linear method, additive increase / multiplicative decrease which converges to high utilization and a fair allocation of throughputs, under some simplifying assumptions. In this paper the author compares Remy's algorithms with several end-to-end schemes. TCP congestion control was not designed with an explicit optimization goal in mind, but instead allows overall network behavior to emerge from its rules.

The congestion control is treated as a problem of distributed decision making under uncertainty. If all the nodes knew in advance the network topology and capacity, and the schedule of each node's present and future offered load. The author approach hinges on being able to evaluate quantitatively the merit of any congestion control algorithm, and search for the best algorithm for a given network model and objective function.

The author treats the network as having been drawn from a stochastic generative process. The author assumes that the network is Markovian. Currently the author parameterizes the networks on the three axes: the speed of bottleneck links, the propagation delay of the network paths and

the degree of multiplexing. The author assumes that the senders have no control over the paths taken by their packets to receiver.

As the conclusion the author in this paper asks whether the design of distributed congestion control algorithms for heterogeneous and dynamic networks can be done by satisfying the assumptions are entitled to have and the policy they ought to achieve, and letting computers work out the details of the per-endpoint mechanisms.

The author developed and evaluated Remy, a program that designs end-to-end congestion-control algorithm to human-supplied specifications. This algorithm outputs handily outperforms the best-known techniques, including the ones that require intrusive in-network changes, in scenarios where network parameters varied over one or two orders of magnitude. As per the result of the author's experiment using the program or algorithm indicates that there is no existing single congestion control method that is the best in all situations. A computer-generated approach that maximizes an explicit function of the throughput and delay to generate algorithms may be the right way forward for the networking community

1. (20 points) Pipelining

Consider an idealized case of two hosts, one located in the United States and the other located in Brazil. The speed of light round-trip propagation delay between the two end systems,  $RTT$ , is approximately 200 milliseconds. Suppose that they are connected by a channel with a transmission rate of  $R$ , of 1 Gbps ( $10^9$  bits per second). Given a pipelined protocol with a pipeline of  $N$  packets, assuming no packet loss, how big would the window size have to be for the channel utilization to be greater than 95 percent? Assume that the size of the data packet is 4096 bytes, including both header fields and data.

**Answer: -**

Ans) According to the question, we can see that

$$\begin{aligned} RTT &= 200 \text{ ms} \\ TR(R) &= 1 \text{ Gbps} = 10^9 \text{ bits per second} \\ L &= 4096 \text{ bytes} = 32,768 \text{ Bits} \end{aligned}$$

The time taken to transfer the packets is  $L/R$

$$= \frac{32768}{10^9} = 0.032768 \text{ milliseconds}$$

Now, Since from the question, we know that the utilization is greater than 95%.

$$\Rightarrow \left[ (N \times \frac{L}{R}) / (RTT + \frac{L}{R}) \right] > 0.95$$

$$\Rightarrow \left[ (n\text{-packets} \times \frac{L}{R}) / (RTT + \frac{L}{R}) \right] > \frac{95}{100}$$

$$\Rightarrow \left[ (N \times 0.032768) / (200.032768) \right] > \frac{95}{100}$$

$$\Rightarrow (N \times 0.032768) > (0.95 \times 200.032768)$$

$$\Rightarrow N > \frac{(0.95 \times 200.032768)}{0.032768}$$



$$\therefore N > 1900.03113 / 0.032768$$

$$\text{NoPackets} > 0.95^* 200.032768 / 0.032768$$

$$\Rightarrow N\text{-packets} > 0.95^* 200.032768 / 0.032768$$

$$\Rightarrow N\text{-packets} > 5799.28984375$$

$\therefore$  The window size will be enough to hold min 5800 packets.

$$2P.0 < (RTT + RTT) / (C/N \times H) \Rightarrow$$

$$P < [(RTT + RTT) / (C/N \times H)] \Rightarrow$$

$$P < [(200.032768) / (500.032768 \times H)] \Rightarrow$$

$$(200.032768 \times 2P.0) < (500.032768 \times H) \Rightarrow$$

$$820.131072 < (500.032768 \times H) \Rightarrow$$

Suppose a TCP Reno sender (congestion avoidance, fast retransmit, fast recovery) has an *ssthresh* = 32 and a *cwnd* = 4. The sender has no outstanding unacknowledged segments and 350 more segments left to transmit. In addition, the receiver it is sending to advertises an empty socket buffer that can hold up to 35 segments in each acknowledgement it sends.

Excluding the TCP FIN handshake at the end of the transfer, how many more roundtrips does it take for the transfer to finish? Assume there is no packet loss. For each round-trip, show how many segments are transmitted.

Answer: -

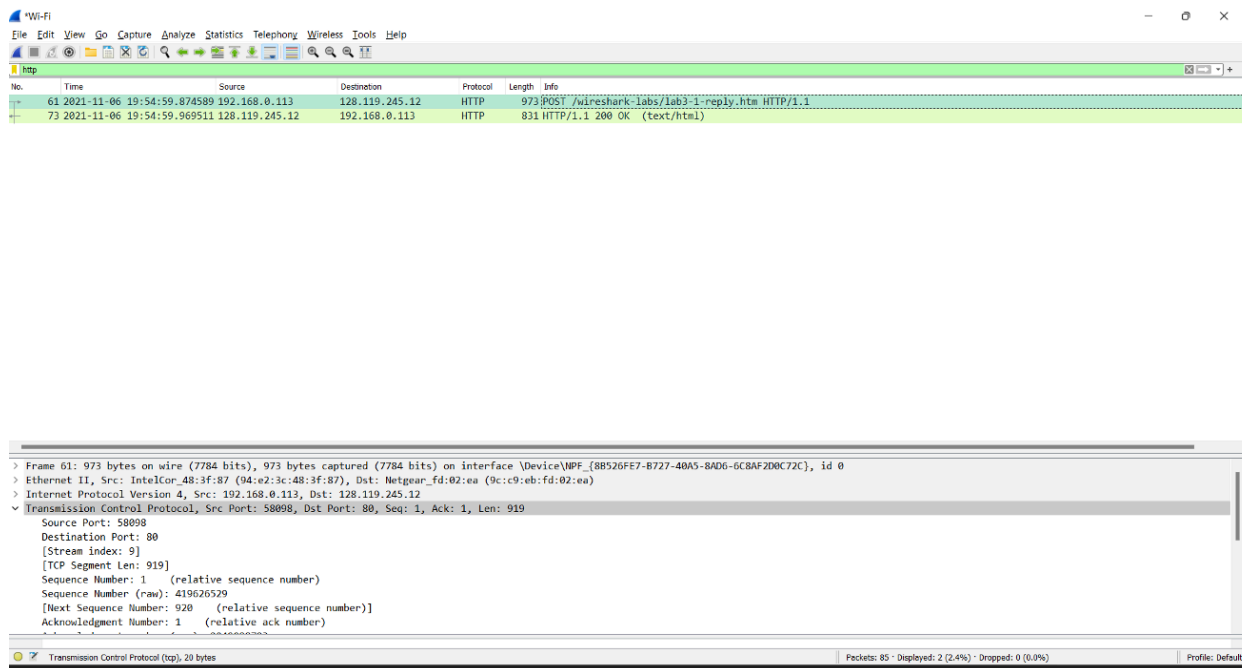
Round Trip	TCP Phase	Cwnd	rwnd	#Segments transmitted	Which segments transmitted
1	TCP Slow Start	4	35	4	1-4
2	TCP Slow Start	8	35	8	5-12
3	TCP Slow Start	16	35	16	13-28
4	TCP Slow Start Size = ssthresh	32	35	32	29-60
5	Congestion avoidance (Additive increase)	33	35	33	61-93
6	Congestion avoidance (Additive increase)	34	35	34	94-127
7	Congestion avoidance (Additive increase)	35	35	35	128-162
8	Congestion avoidance (Additive increase)	36	35	35	163-197
9	Congestion avoidance (Additive increase)	37	35	35	198-232
10	Congestion avoidance (Additive increase)	38	35	35	233-267

11	Congestion avoidance (Additive increase)	39	35	35	268-302
12	Congestion avoidance (Additive increase)	40	35	35	303-337
13	Congestion avoidance (Additive increase)	41	35	13	338-350

### 3. (40 points) Wireshark Lab: TCP

**Ans)**

The IP address is 192.168.0.113 and the port number is 58094



**2) What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?**

**ANS)** The IP Address is 192.119.245.12 and the port number is 80

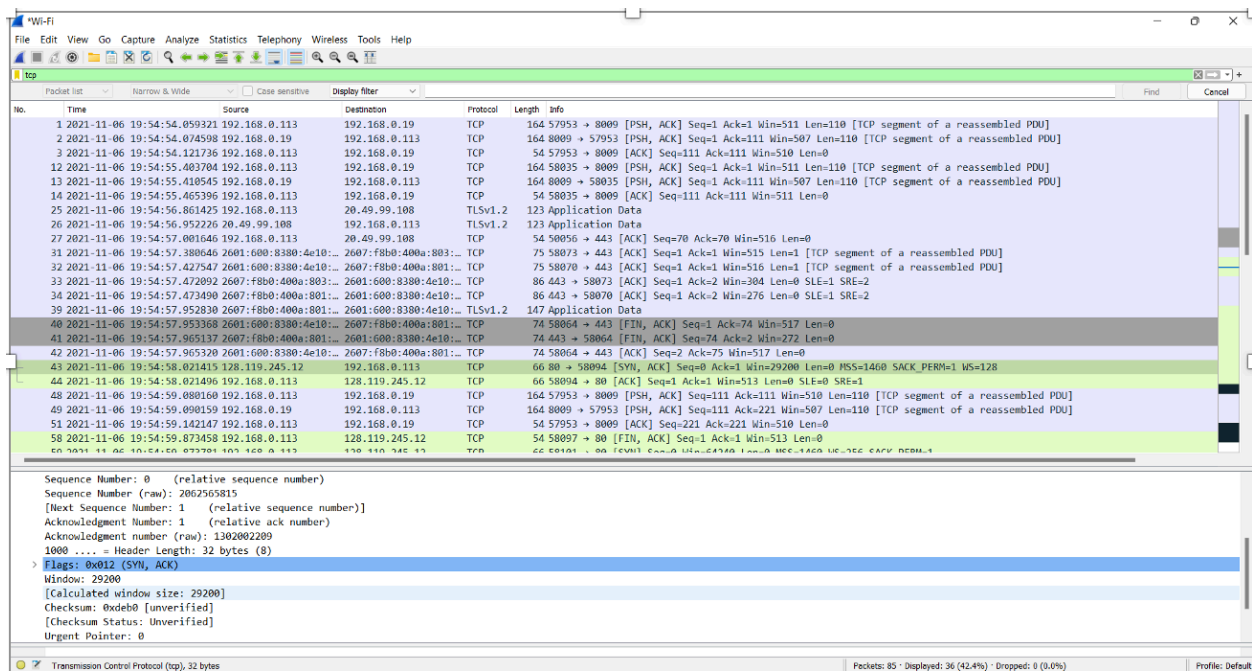
**3) What is the IP address and TCP port number used by your client computer (source) to transfer the file to gaia.cs.umass.edu?**

**ANS)** The IP Address is 192.168.0.113 and the TCP Source Port number is 58098

**4) What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?**

**ANS)** The SYN segment is indicated as 0

**5) What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?**



**6) What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you'll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a "POST" within its DATA field.**

Source Port: 58098  
Destination Port: 80  
[Stream Index: 3]  
[TCP Segment Len: 919]  
Sequence Number: 1 (relative sequence number)  
Sequence Number (raw): 419626529  
[Next Sequence Number: 920 (relative sequence number)]  
Acknowledgment Number: 1 (relative ack number)  
Acknowledgment number (raw): 2040028793  
0101 .... = Header Length: 20 bytes (5)  
> Flags: 0x018 (PSH, ACK)

0000 3c c9 4b fd 02 ea 94 e2 3c 48 3f 87 08 00 45 00 ..... <H>...E  
0010 03 bf 8c 2c 40 00 06 00 00 c9 a8 00 71 80 72 ...|B... ..-q-w  
0020 f5 9c e2 f2 00 50 19 02 fe 21 7a 21 b2 b9 50 18 ...P... :|L:-P  
0030 02 01 3a 4f 00 00 50 4f 53 54 20 2f 77 69 72 65 ...:O:-PO S! /win  
0040 73 68 61 72 6b 2d 6c 61 62 73 2f 6c 61 62 33 2d ...shark-la bs/lab3-  
0050 31 2d 72 65 70 6c 79 2e 68 74 6d 20 48 54 54 50 ...l.reply.htm HTTP  
0060 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 67 61 09 61 .../1:1-the str: gaia  
0070 2e 63 73 2e 75 6d 61 73 75 2e 65 64 75 0d 0a 42 ...cs.umass.edu-c  
0080 6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 70 2d ...connectio n: keep  
0090 61 6c 69 70 65 0d 0a 43 0f 6e 74 65 6e 74 2d 4c ...alive-c ontent-l  
00a0 65 6e 67 74 68 3a 20 31 38 38 0d 0a 43 61 63 68 ...length: 1 88--Cach  
00b0 65 2d 43 6f 6e 74 72 6f 6c 3a 20 6d 61 78 2d 61 ...e-Contro l: max-a  
00c0 67 65 2d 30 0d 0a 55 70 67 72 61 64 65 2d 49 6a ...ge-0--Up grade-In  
00d0 73 65 63 75 72 65 2d 52 65 71 75 65 73 74 73 3a ...secure-B equests:  
00e0 20 31 0d 0a 4f 72 69 67 69 6e 3a 20 68 74 74 70 ...1--Orig in: http  
00f0 3a 2f 2f 67 61 69 61 2e 63 73 2e 75 6d 61 73 73 ...//gaia. cs.umass  
0100 2e 65 64 75 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79 ...edu--Co ntent-Ty  
0110 70 65 3a 20 6d 75 6c 74 69 70 61 72 74 2f 66 61 ...pe: mult ipart/fo

The sequence number is 1

7) Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection (including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the EstimatedRTT value (see Section 3.5.3, page 242 in text) after the receipt of each ACK? Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation on page 242 for all subsequent segments.

Note: Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the “listing of captured packets” window that is being sent from the client to the gaia.cs.umass.edu server. Then select: Statistics->TCP Stream Graph->Round Trip Time Graph.

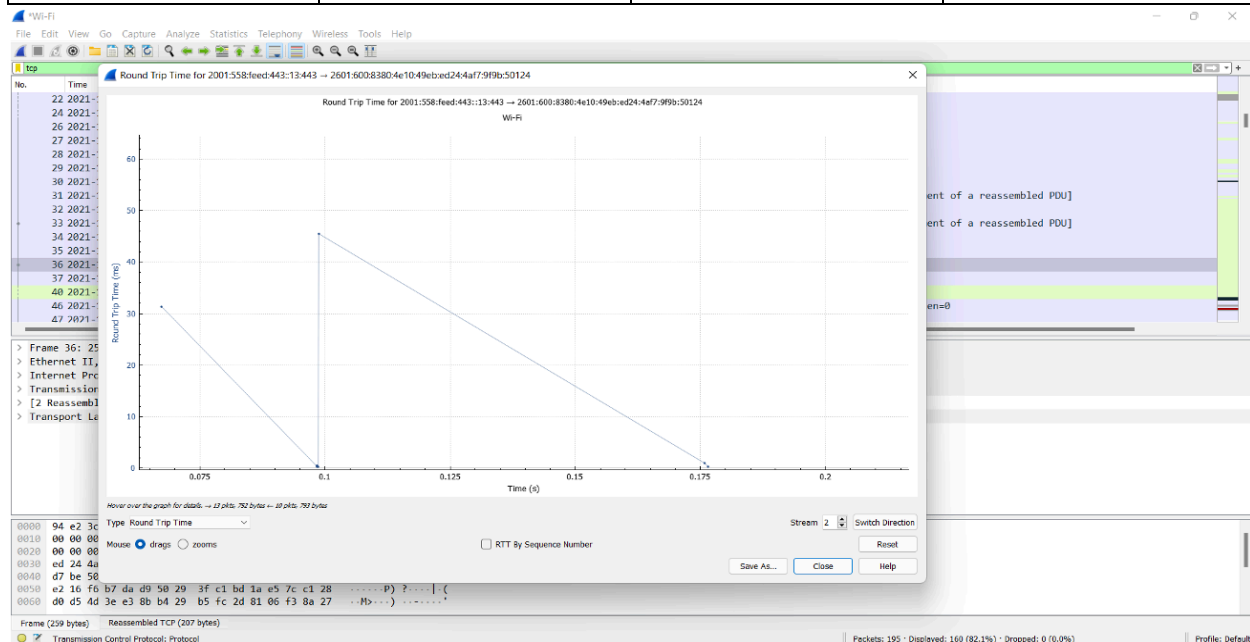
**ANS)**

The segments 1,2,3,5,7 and 8 are the first six segments.

Segment No.	Sent	ACK received	RTT Values
1	15.394058	15.431784	0.037726



2	15.431899	15.439434	0.007535
3	15.433249	15.950503	0.517254
5	15.916967	16.202343	0.285376
7	16.129955	16.255145	0.125190
8	17.005997	17.006162	0.000165



**8) What is the length of each of the first six TCP segments?**

**ANS)** The length of first six TCP segments are 164 and 54 bytes

**9)What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?**

**ANS)** The minimum amount of buffer space is 243 and whereas the maximum amount of buffer space is 64240. There was no throttle observed.

The screenshot shows the Wireshark interface with a packet list filter of 'tcp'. The packet list shows several TCP segments. The selected packet (Packet 1) is a TCP segment with the following details:

- Sequence Number: 0 (relative sequence number)
- Sequence Number (raw): 262006261
- [Next Sequence Number: 1 (relative sequence number)]
- Acknowledgment Number: 0
- Acknowledgment number (raw): 0
- 1000 .... = Header Length: 32 bytes (8)
- Flags: 0x002 (SYN)
- Window: 64240
- [Calculated window size: 64240]
- Checksum: 0x36c4 [unverified]
- [Checksum Status: Unverified]

The packet bytes are displayed in hexadecimal and ASCII. The status bar at the bottom indicates: Packets: 85 · Displayed: 36 (42.4%) · Dropped: 0 (0.0%)

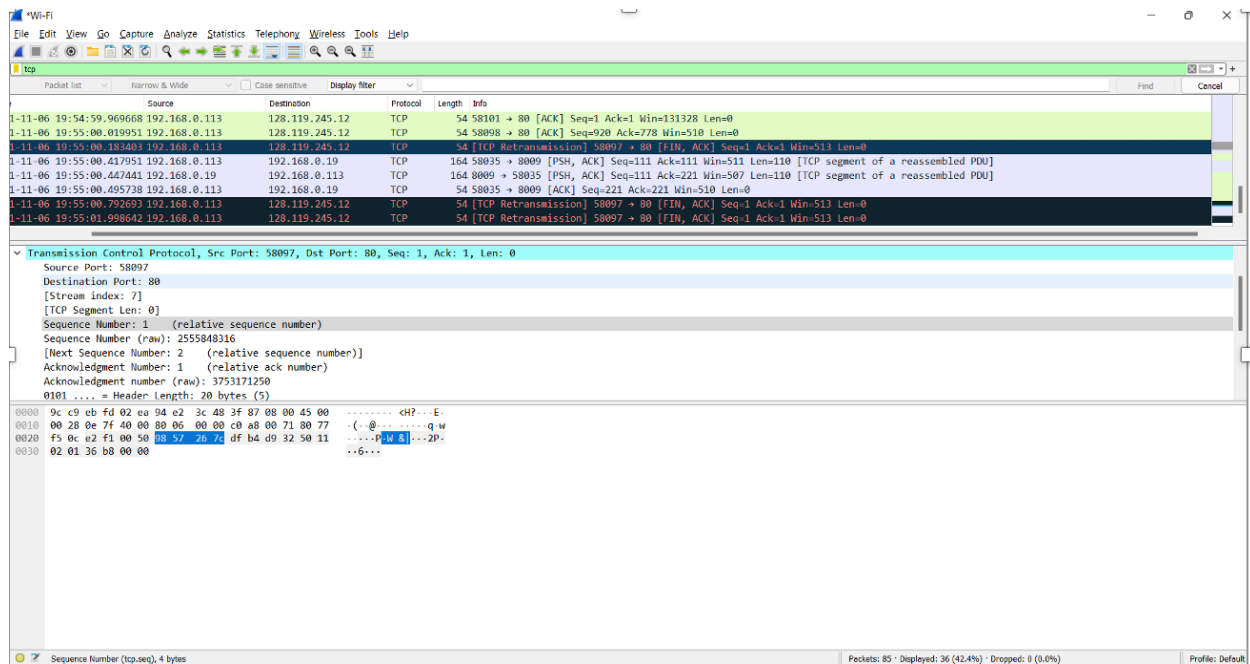
The screenshot shows the Wireshark interface with a packet list filter of 'tcp'. The packet list shows several TCP segments. The selected packet (Packet 2) is a TCP segment with the following details:

- Sequence Number: 1 (relative sequence number)
- Sequence Number (raw): 2049028793
- [Next Sequence Number: 1 (relative sequence number)]
- Acknowledgment Number: 920 (relative ack number)
- Acknowledgment number (raw): 419627448
- 0101 .... = Header Length: 20 bytes (5)
- Flags: 0x010 (ACK)
- Window: 243
- [Calculated window size: 243]
- [Window size scaling factor: -1 (unknown)]
- Checksum: 0x4dd6 [unverified]

The packet bytes are displayed in hexadecimal and ASCII. The status bar at the bottom indicates: Packets: 85 · Displayed: 36 (42.4%) · Dropped: 0 (0.0%)

**10) Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?**

**ANS)** Yes, there is retransmission segments in the trace file



**11. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment (see Table 3.2 on page 250 in the text).**

**ANS)** A sender often sends a large number of segments back-to-back, if one segment is lost, there will likely be many back-to-back duplicate ACKs. If the TCP sender receives three duplicate ACKs for the same data, it takes this as an indication that the segment following the segment that has been ACKed three times has been lost.

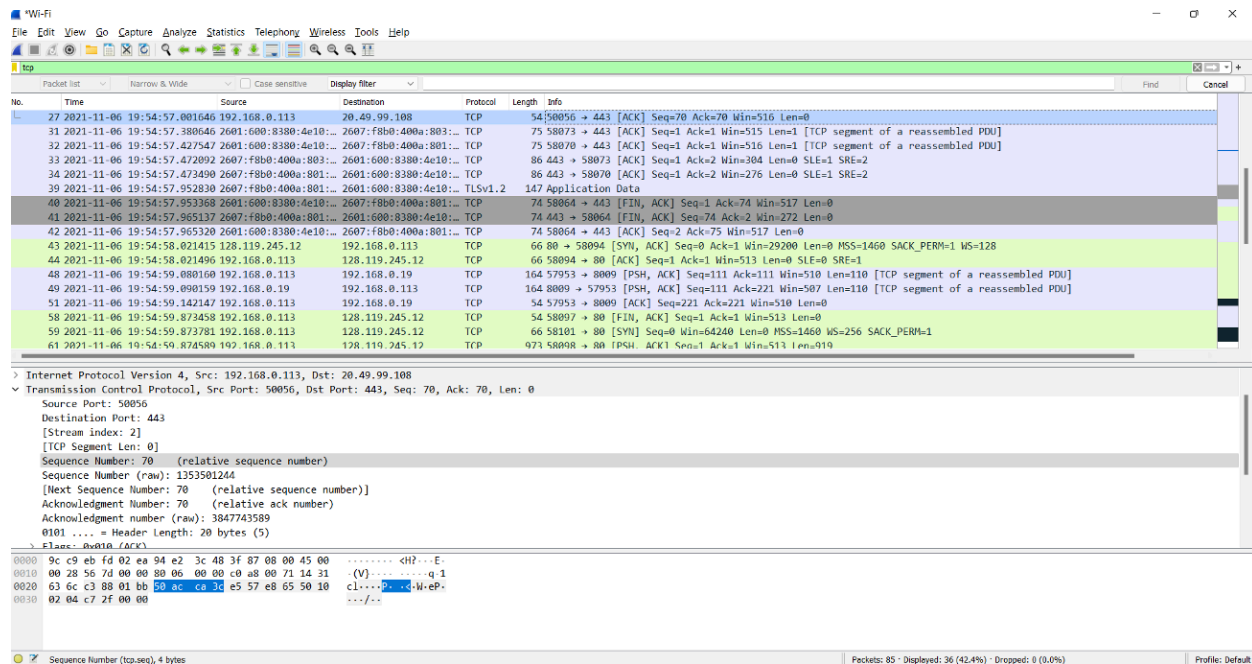
**12. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.**

**ANS)**

Sequence Number is: 1353501244 byte

Time for segment = 3.676751 sec

Throughput =  $1353501244 / 3.676751$   
= 38812426 byte/sec



Wireshark packet capture showing TCP segments. The top pane displays a list of packets with columns for No., Time, Source, Destination, Protocol, Length, and Info. The bottom pane shows the details of a selected segment (Seq=70, Len=0).

Packet list (filtered by 'tcp'):

No.	Time	Source	Destination	Protocol	Length	Info
27	2021-11-06 19:54:57.001646	192.168.0.113	20.49.99.108	TCP	54	50056 → 443 [ACK] Seq=70 Ack=70 Win=516 Len=0
31	2021-11-06 19:54:57.380646	2601:600:8380:4e10::	2607:f8b0:400a:803::	TCP	75	58073 → 443 [ACK] Seq=1 Ack=1 Win=515 Len=1 [TCP segment of a reassembled PDU]
32	2021-11-06 19:54:57.427547	2601:600:8380:4e10::	2607:f8b0:400a:803::	TCP	75	58070 → 443 [ACK] Seq=1 Ack=1 Win=516 Len=1 [TCP segment of a reassembled PDU]
33	2021-11-06 19:54:57.472092	2601:600:8380:4e10::	2607:f8b0:400a:803::	TCP	86	443 → 58073 [ACK] Seq=1 Ack=2 Win=304 Len=0 SLE=1 SRE=2
34	2021-11-06 19:54:57.473490	2601:600:8380:4e10::	2607:f8b0:400a:803::	TCP	86	443 → 58070 [ACK] Seq=1 Ack=2 Win=276 Len=0 SLE=1 SRE=2
39	2021-11-06 19:54:57.952830	2601:600:8380:4e10::	2607:f8b0:400a:803::	TLSv1.2	147	Application Data
40	2021-11-06 19:54:57.953368	2601:600:8380:4e10::	2607:f8b0:400a:803::	TCP	74	58064 → 443 [FIN, ACK] Seq=1 Ack=74 Win=517 Len=0
41	2021-11-06 19:54:57.965137	2601:600:8380:4e10::	2607:f8b0:400a:803::	TCP	74	443 → 58064 [FIN, ACK] Seq=74 Ack=2 Win=272 Len=0
42	2021-11-06 19:54:57.965320	2601:600:8380:4e10::	2607:f8b0:400a:803::	TCP	74	58064 → 443 [ACK] Seq=2 Ack=75 Win=517 Len=0
43	2021-11-06 19:54:58.021415	128.119.245.12	192.168.0.113	TCP	66	80 → 58094 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
44	2021-11-06 19:54:58.021496	128.119.245.12	192.168.0.113	TCP	66	58094 → 80 [ACK] Seq=1 Ack=1 Win=513 Len=0 SLE=0 SRE=1
48	2021-11-06 19:54:59.080160	192.168.0.19	192.168.0.113	TCP	164	57953 → 8009 [PSH, ACK] Seq=111 Ack=111 Win=510 Len=110 [TCP segment of a reassembled PDU]
49	2021-11-06 19:54:59.090159	192.168.0.19	192.168.0.113	TCP	164	8009 → 57953 [PSH, ACK] Seq=111 Ack=221 Win=507 Len=110 [TCP segment of a reassembled PDU]
51	2021-11-06 19:54:59.142147	192.168.0.19	192.168.0.113	TCP	54	57953 → 8009 [ACK] Seq=221 Ack=221 Win=510 Len=0
58	2021-11-06 19:54:59.873458	128.119.245.12	192.168.0.113	TCP	54	58097 → 80 [FIN, ACK] Seq=1 Ack=1 Win=513 Len=0
59	2021-11-06 19:54:59.873791	128.119.245.12	192.168.0.113	TCP	66	58101 → 80 [SYN] Seq=0 Win=54240 Len=0 MSS=1460 WS=256 SACK_PERM=1
61	2021-11-06 19:54:59.874589	192.168.0.113	128.119.245.12	TCP	973	58098 → 80 [PSH, ACK] Seq=1 Ack=1 Win=513 Len=919

Details of selected packet (No. 43):

- Internet Protocol Version 4, Src: 192.168.0.113, Dst: 20.49.99.108
- Transmission Control Protocol, Src Port: 50056, Dst Port: 443, Seq: 70, Ack: 70, Len: 0
  - Source Port: 50056
  - Destination Port: 443
  - [Stream index: 2]
  - [TCP Segment Len: 0]
  - Sequence Number: 70 (relative sequence number)
  - Sequence Number (raw): 1353501244
  - [Next Sequence Number: 70 (relative sequence number)]
  - Acknowledgment Number: 70 (relative ack number)
  - Acknowledgment number (raw): 3847743589
  - 0101 ... = Header Length: 20 bytes (5)
  - Flags: 0x010 (ACK)

Sequence Number (tcp.seq), 4 bytes

Packets: 85 · Displayed: 36 (42.4%) · Dropped: 0 (0.0%)

Profile: Default

**13) Use the Time-Sequence-Graph(Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify where TCP's slow start phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we've studied in the text**

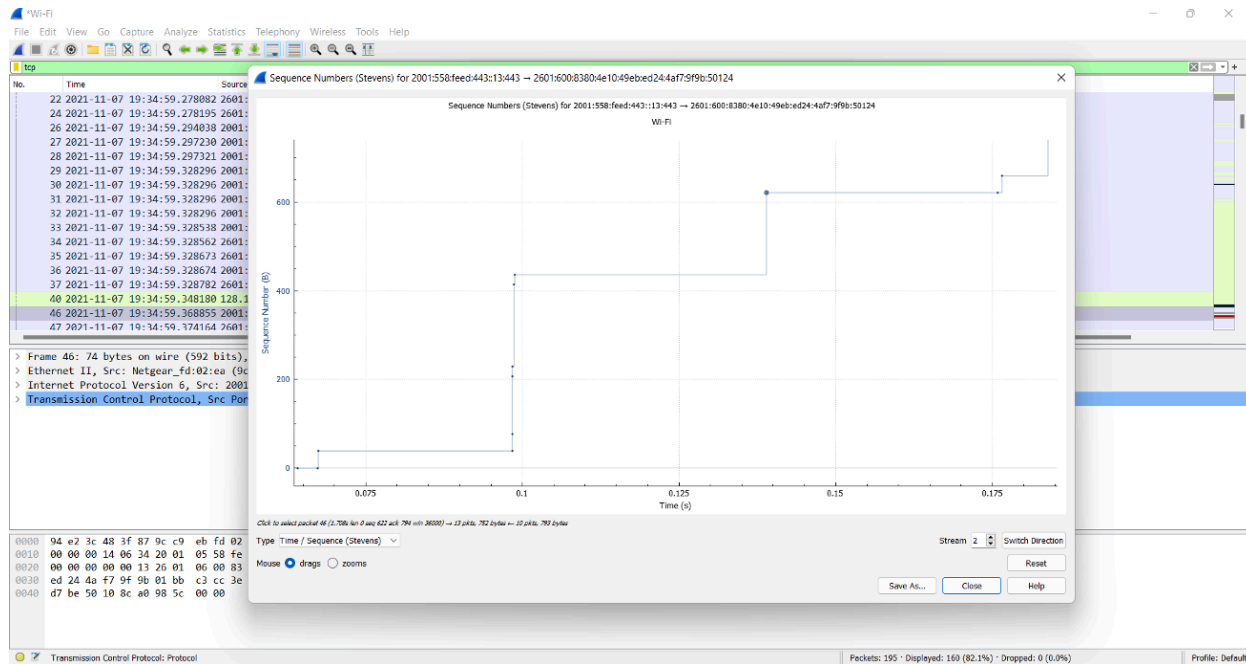
**ANS)**

From the graph mentioned in the wireshark.pdf, we can tell that TCP connection starts from the 0<sup>th</sup> second at the beginning of the graph from 0 and ends at 0.15 seconds precisely. Congestion avoidance starts at 0.4 secs as it reduces the quantity send.

To decide the finish of slow start and the start of the blockage evasion, we take a glance at how clog window estimate responds to the entry of ACKs. On an ACK entry, if the clog window measure expands, TCP sender remains in the slow start stage. In the clog evasion stage, the blockage window measure increments at  $1/(\text{current\_congestion\_window\_size})$ . Seeing the difference in the clog window upon the entry of ACKs, we can discover the conditions of the TCP sender.

TCP senders ought to pursue the AIMD algorithm so when they distinguish parcel misfortune their sending window size should drop down. In our model, when the TCP sender conveys information, even before the finish of slow start stage, the transmission is over a direct result of little size of information.





**14) Answer each of two questions above for the trace that you have gathered when you transferred a file from your computer to gaia.cs.umass.edu**

**ANS)** Answered in the previous question