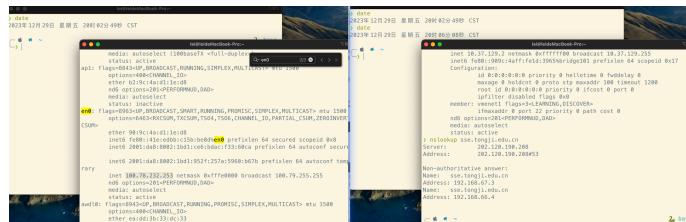


1 Q1

1.1 Q1.1

The IP address used by my own computer is `100.78.232.253`. For IPv6 address, my own computer is `2001:da8:8002:1bd1:952f:257a:5960:b67b`.

The IP address of sse.tongji.edu.cn is `192.168.67.3` or `192.168.66.4`.



1.2 Q1.2

After launching additional applications like QQ and WeChat on my computer that can access the Internet, I started capturing network traffic again while keeping the previously accessed website open. After a few seconds, I stopped the capture.

first.pcapng						
Time	Time	Source	Destination	Protocol	Length	Info
frame.time 1 2023-12-29 20:41:09.409473		2001:da8:8002:1bd1:952f:257a:5960:b67b	2404:6800:4012:4::200a	TCP	98	63161 → 443 [SYN] Seq=0 Win=65535 L
2 2023-12-29 20:41:09.464937		100.78.232.253	142.251.43.10	TCP	78	63162 → 443 [SYN] Seq=0 Win=65535 L
3 2023-12-29 20:41:09.713191		100.78.232.253	142.251.43.10	TCP	78	63163 → 443 [SYN] Seq=0 Win=65535 L
4 2023-12-29 20:41:09.748769		2001:da8:8002:1bd1:952f:257a:5960:b67b	2407:b380:8000:c:103:74:50:133	TCP	98	63173 → 80 [SYN] Seq=0 Win=65535 L
5 2023-12-29 20:41:09.785449		2407:b380:8000:c:103:74:50:133	2001:da8:8002:1bd1:952f:257a:5960:b67b	TCP	94	80 → 63173 [SYN, ACK] Seq=1 Ack=1 Win=27
6 2023-12-29 20:41:09.785560		2001:da8:8002:1bd1:952f:257a:5960:b67b	2407:b380:8000:c:103:74:50:133	TCP	86	63173 → 88 [ACK] Seq=1 Ack=1 Win=27
7 2023-12-29 20:41:09.786501		2001:da8:8002:1bd1:952f:257a:5960:b67b	2407:b380:8000:c:103:74:50:133	HTTP	757	GET /search?q=&pos=-1&id=1E27E861A
8 2023-12-29 20:41:09.830284		2407:b380:8000:c:103:74:50:133	2001:da8:8002:1bd1:952f:257a:59..	TCP	86	80 → 63173 [ACK] Seq=1 Ack=672 Win=27
9 2023-12-29 20:41:09.830285		2407:b380:8000:c:103:74:50:133	2001:da8:8002:1bd1:952f:257a:59..	HTTP..	1156	HTTP/1.1 200
10 2023-12-29 20:41:09.830430		2001:da8:8002:1bd1:952f:257a:5960:b67b	2407:b380:8000:c:103:74:50:133	TCP	86	63173 → 88 [ACK] Seq=672 Ack=1071 Win=27
11 2023-12-29 20:41:10.008300		100.78.232.253	1.1.1.1	ICMP	58	Echo (ping) request id=0x8c39, seq=1
12 2023-12-29 20:41:10.138210		Apple_d1:e:d8	Broadcast	ARP	42	Who has 192.168.1.1? Tell 192.168.1.1
13 2023-12-29 20:41:10.138269		Parallels_a4:51:91	Broadcast	ARP	42	Who has 192.168.1.1? Tell 192.168.1.1
14 2023-12-29 20:41:10.277734		1.1.1.1	100.78.232.253	ICMP	58	Echo (ping) reply id=0x8c39, seq=1
15 2023-12-29 20:41:10.463233		100.78.232.253	239.255.255.250	SSDP	218	M-SEARCH * HTTP/1.1
16 2023-12-29 20:41:10.463240		10.211.55.2	239.255.255.250	SSDP	218	M-SEARCH * HTTP/1.1
17 2023-12-29 20:41:10.463263		10.37.129.2	239.255.255.250	SSDP	218	M-SEARCH * HTTP/1.1
18 2023-12-29 20:41:10.667860		100.78.232.253	120.92.107.7	TLSv..	136	Application Data
19 2023-12-29 20:41:10.668059		100.78.232.253	120.92.107.7	TLSv..	501	Application Data
20 2023-12-29 20:41:10.699547		120.92.107.7	100.78.232.253	TCP	66	443 → 63077 [ACK] Seq=1 Ack=530 Win=27

From the packet trace, several connections can be observed. To focus on two specific connections, I applied filters in Wireshark as follows:

```
1 | ip.dst == 100.78.232.253 and dns.qry.name ==
  "sse.tongji.edu.cn" and not dns.qry.type == 28
```

first.pcapng						
Time	Time	Source	Destination	Protocol	Length	Info
frame.time 5 2023-12-29 20:41:11.405882		202.120.190.208	100.78.232.253	DNS	109	Standard query response 0x0684 A sse.tongji.edu.cn
38 2023-12-29 20:41:11.412614		202.120.190.208	100.78.232.253	DNS	122	Standard query response 0xdd37 HTTPS ss...

The Wireshark filter is set to: `ip.dst == 100.78.232.253` to filter packets destined for IP `100.78.232.253`, `dns.qry.name == "sse.tongji.edu.cn"` for DNS queries to `"sse.tongji.edu.cn"`, and `not dns.qry.type == 28` to exclude AAAA (IPv6) DNS queries.

2 Q2

2.1 Q2.1

a. Use the following command to filter in Wireshark,:;

1 | ipv6.dst==2001:da8:b8:67:192:168:67:3

The screenshot shows a Wireshark capture window titled "first.pcapng". A single packet is selected, which is a TCP SYN segment. The details pane shows the following information:

- Source: 2001:da8:8002:1bd1:952f:257a:5960:b67b
- Destination: 2001:da8:b8:67:192:168:67:3
- Protocol: TCP
- Length: 98
- Info: 63175 → 443 [SYN] Seq=0 Win=65535 L

The "Sequence Number (raw)" field is highlighted with a red box and contains the value **4167028726**.

The sequence number of the TCP SYN segment that is used to initiate the TCP connection between the source and destination is **4167028726**.

b. A SYN segment is identified when the "flags" field in the TCP header is set to **0x002**, corresponding to the **SYN** flag bit being set to **1**.

The screenshot shows the same Wireshark capture window. The "Flags" section of the TCP header is highlighted with a red box. An arrow points to the "Syn" entry, which is marked with a red box and shows the value **1**.

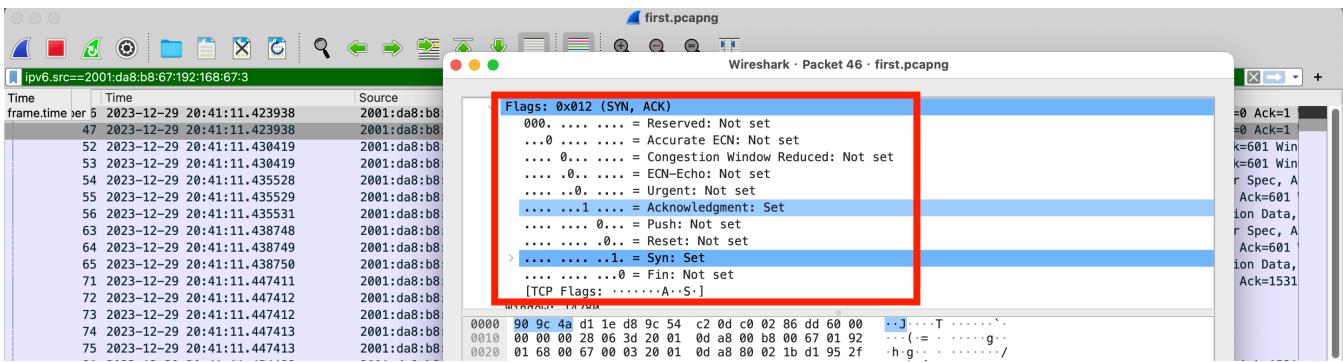
2.2 Q2.2

a. The sequence number of the SYNACK segment sent by destination to the source in reply to the SYN is **3855327051**.

The screenshot shows a Wireshark capture window. A single packet is selected, which is a TCP SYNACK segment. The details pane shows the following information:

- Source Port: 443
- Destination Port: 63175
- [Stream index: 6]
- [Conversation completeness: Incomplete, DATA (15)]
- [TCP Segment Len: 0]
- Sequence Number: 0 (relative sequence number)
- Sequence Number (raw): 3855327051
- (Next Sequence Number: 1 (relative sequence number))
- Acknowledgment Number: 1 (relative ack number)
- Acknowledgment number (raw): 4167028727

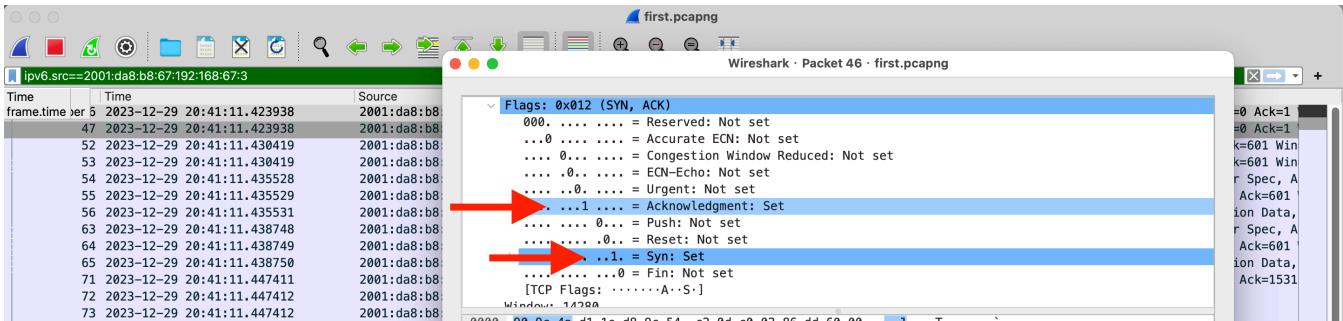
b. The value of the Acknowledgement filed in the SYNACK segment is below:



- c. The destination determines the value in the acknowledgment field by adding one to the sequence number received in the TCP segment from the source.

$$\text{Acknowledgement} = \text{sequence number} + 1$$

- d. A segment is identified as a SYNACK segment based on specific bits set in the TCP header's flags field. In this case, the flags value is set to **0x012**, which corresponds to both the **Acknowledgment** and **SYN** flags being set to **1**.



2.3 Q2.3

The TCP "three-way handshake" is an essential and non-reducible process for establishing a connection. It involves SYN, SYN-ACK, and ACK. This process is crucial for synchronizing sequence numbers for reliable data transmission, negotiating connection parameters, preventing issues from delayed packets of old connections, and ensuring security and reliability. Any reduction in these steps could compromise these key aspects.

2.4 Q2.4

I pick segment No.11 to analysis.

- a. Sent time: **2023-06-30 14:29:20.848409**

Destination Time	Source	Destination	Protocol	Length	Info
2023-06-30 14:29:28.844633	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [ACK] Seq=1 Ack=1 Win=501 Len=1424
2 2023-06-30 14:29:28.844658	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [PSH, ACK] Seq=1425 Ack=1 Win=501
3 2023-06-30 14:29:28.844669	100.75.250.48	146.56.196.163	TCP	54	50555 → 8081 [ACK] Seq=1 Ack=2849 Win=8332 Len=1
4 2023-06-30 14:29:28.844681	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [ACK] Seq=2849 Ack=1 Win=501 Len=1
5 2023-06-30 14:29:28.844692	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [PSH, ACK] Seq=4273 Ack=1 Win=501
6 2023-06-30 14:29:28.844698	100.75.250.48	146.56.196.163	TCP	54	50555 → 8081 [ACK] Seq=1 Ack=5697 Win=8332 Len=1
7 2023-06-30 14:29:28.844711	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [ACK] Seq=5697 Ack=1 Win=501 Len=1
8 2023-06-30 14:29:28.844720	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [PSH, ACK] Seq=7121 Ack=1 Win=501
9 2023-06-30 14:29:28.844727	100.75.250.48	146.56.196.163	TCP	54	50555 → 8081 [ACK] Seq=1 Ack=8545 Win=8332 Len=1
10 2023-06-30 14:29:28.844735	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [ACK] Seq=8545 Ack=1 Win=501 Len=1
11 2023-06-30 14:29:28.848489	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [PSH, ACK] Seq=9969 Ack=1 Win=501
12 2023-06-30 14:29:28.848496	100.75.250.48	146.56.196.163	TCP	54	50555 → 8081 [ACK] Seq=1 Ack=11393 Win=8332 Len=1
13 2023-06-30 14:29:28.848548	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [ACK] Seq=11393 Ack=1 Win=501 Len=1
14 2023-06-30 14:29:28.848681	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [PSH, ACK] Seq=12817 Ack=1 Win=501
15 2023-06-30 14:29:28.848702	100.75.250.48	146.56.196.163	TCP	54	50555 → 8081 [ACK] Seq=1 Ack=14241 Win=8332 Len=1
16 2023-06-30 14:29:28.848869	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [ACK] Seq=14241 Ack=1 Win=501 Len=1
17 2023-06-30 14:29:28.849046	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [PSH, ACK] Seq=15665 Ack=1 Win=501
18 2023-06-30 14:29:28.849061	100.75.250.48	146.56.196.163	TCP	54	50555 → 8081 [ACK] Seq=1 Ack=17089 Win=8332 Len=1
19 2023-06-30 14:29:28.849225	146.56.196.163	100.75.250.48	TCP	1478	8081 → 50555 [ACK] Seq=17089 Ack=1 Win=501 Len=1

b. For segment No.11 , the Sequence Number is 9969.

No.	Time	Source Port	Destination Port	Stream index:	Conversation completeness:	[TCP Segment Len: 1424]	Sequence Number:	Sequence Number (raw):	[Next Sequence Number: 11393 (relative sequence number)]	Acknowledgment Number:	Acknowledgment number (raw):	Flags:	Header Length: 20 bytes (5)	ACK] Seq=1 Ack=1 Win=501 Len=1424	PSH, ACK] Seq=1425 Ack=1 Win=501	ACK] Seq=2849 Ack=1 Win=501 Len=1	PSH, ACK] Seq=1393 Ack=1 Win=501	ACK] Seq=5697 Ack=1 Win=8332 Len=1	PSH, ACK] Seq=7121 Ack=1 Win=501	ACK] Seq=1 Ack=8545 Win=8332 Len=1	ACK] Seq=8545 Ack=1 Win=501 Len=1	PSH, ACK] Seq=9969 Ack=1 Win=501	ACK] Seq=1 Ack=11393 Win=8332 Len=1	ACK] Seq=11393 Ack=1 Win=501 Len=0	PSH, ACK] Seq=12817 Ack=1 Win=501	ACK] Seq=1 Ack=14241 Win=8332 Len=1
1	2023-06-30 14:29:20.844633																									
2	2023-06-30 14:29:20.844658																									
3	2023-06-30 14:29:20.844669																									
4	2023-06-30 14:29:20.844681																									
5	2023-06-30 14:29:20.844692																									
6	2023-06-30 14:29:20.844698																									
7	2023-06-30 14:29:20.844711																									
8	2023-06-30 14:29:20.844720																									
9	2023-06-30 14:29:20.848489																									
10	2023-06-30 14:29:20.848496																									
11	2023-06-30 14:29:20.848499																									
12	2023-06-30 14:29:20.848548																									
13	2023-06-30 14:29:20.848548																									
14	2023-06-30 14:29:20.848681																									
15	2023-06-30 14:29:20.848702																									
16	2023-06-30 14:29:20.848869																									
17	2023-06-30 14:29:20.849046																									
18	2023-06-30 14:29:20.849061																									
19	2023-06-30 14:29:20.849225																									

$$Ack = Sequence\ Number + Length\ of\ TCP\ Segment\ Data$$

Hence, $Ack = 9969 + 1424 = 11393$

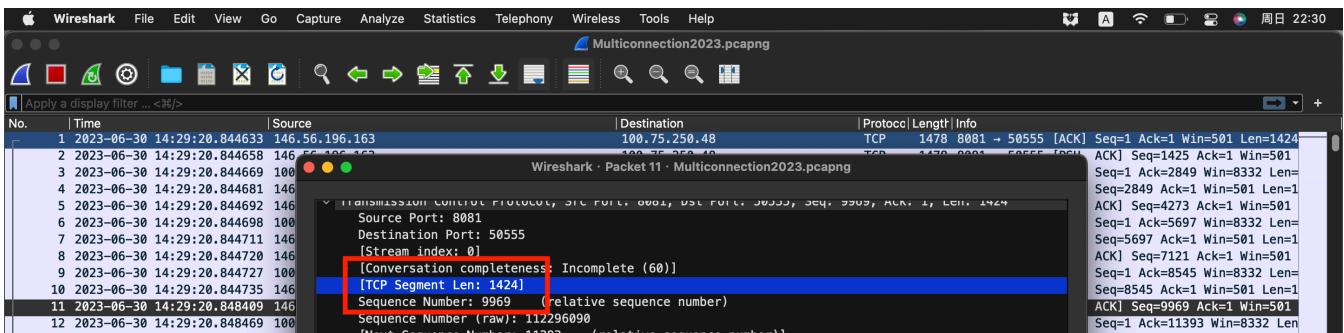
To filter this in Wireshark, use the following command:

```
1 | tcp.ack == 11393
```

No.	Time	Source	Destination	Protocol	Length	Info
12	2023-06-30 14:29:20.848469	100.75.250.48	146.56.196.163	TCP	54	50555 → 8081 [ACK] Seq=1 Ack=11393 Win=8332 Len=0
32	2023-06-30 14:29:20.852463	100.75.250.48	146.56.196.163	TCP	54	50564 → 8081 [ACK] Seq=1 Ack=11393 Win=4149 Len=0
117	2023-06-30 14:29:20.862646	100.75.250.48	146.56.196.163	TCP	54	50556 → 8081 [ACK] Seq=1 Ack=11393 Win=4171 Len=0
177	2023-06-30 14:29:20.878477	100.75.250.48	146.56.196.163	TCP	54	50568 → 8081 [ACK] Seq=1 Ack=11393 Win=4149 Len=0
281	2023-06-30 14:29:20.890371	100.75.250.48	146.56.196.163	TCP	54	50562 → 8081 [ACK] Seq=1 Ack=11393 Win=4160 Len=0
341	2023-06-30 14:29:20.918519	100.75.250.48	146.56.196.163	TCP	54	50563 → 8081 [ACK] Seq=1 Ack=11393 Win=4149 Len=0
937	2023-06-30 14:41:56.522516	100.75.250.48	146.56.196.163	TCP	54	51822 → 8081 [ACK] Seq=398 Ack=11393 Win=131328 Len=0
937..	2023-06-30 14:41:56.524394	100.75.250.48	146.56.196.163	TCP	54	51828 → 8081 [ACK] Seq=398 Ack=11393 Win=131328 Len=0
937..	2023-06-30 14:41:56.530355	100.75.250.48	146.56.196.163	TCP	54	51826 → 8081 [ACK] Seq=398 Ack=11393 Win=131328 Len=0
938..	2023-06-30 14:41:56.531269	100.75.250.48	146.56.196.163	TCP	54	51831 → 8081 [ACK] Seq=397 Ack=11393 Win=132352 Len=0
938..	2023-06-30 14:41:56.544136	100.75.250.48	146.56.196.163	TCP	54	51824 → 8081 [ACK] Seq=398 Ack=11393 Win=131328 Len=0
938..	2023-06-30 14:41:56.544940	100.75.250.48	146.56.196.163	TCP	54	51829 → 8081 [ACK] Seq=398 Ack=11393 Win=131328 Len=0

The ACK for this segment was received at [2023-06-30 14:29:20.848469](https://www.timeanddate.com/time/zone/america/los_angeles?year=2023&month=6&day=30&hour=14&minute=29&second=20).

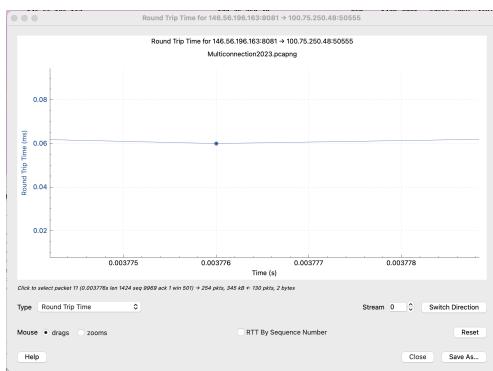
c. The length of the segment is **1424**.



d. RTT (Round-Trip Time) is the time difference between sending a TCP segment and receiving its ACK.

$$\text{RTT} = 2023-06-30\ 14:29:20.848469 - 2023-06-30\ 14:29:20.848409 = 0.000060s = 0.06ms$$

Using Wireshark, go to [Statistics → TCP Stream Graph → Round Trip Time Graph](#) and select segment 11, we can see the result is 0.06ms.



So the RTT value for this segment is 0.06ms.

2.5 Q2.5

a. First, apply the following filter in Wireshark

```
1 | ip.addr==100.75.250.48 && ip.addr==120.233.43.93
```

Then, navigate to [Statistics → TCP Stream Graphs → Time Sequence \(tcptrace\)](#) to generate a tcptrace graph. Based on TA's suggestion, set the Stream to 110. The resulting graph provides valuable insights:



- **Green Line (Send Window Limit):** This line represents the sender's perception of the receiver's window limit. When the blue line approaches the green line, it suggests that the sender has reached what it believes to be the limit of the receiver's window, and it must wait for a new ACK to send more data.
- **Blue Line (Send Sequence Number):** This line grows over time, indicating that packets are being sent. A halt in the growth of the blue line signifies a temporary pause in data transmission by the sender.
- **Yellow Line (Acknowledgment Sequence Number):** The growth of this line indicates that the receiver has received and acknowledged the packets. If the growth of the yellow line pauses, it may mean that the sender is waiting for more acknowledgments to continue sending data.

In Wireshark, clicking on `Switch Direction` allows us to switch between the two directions of the TCP connection, highlighting the full-duplex communication nature of TCP/IP.

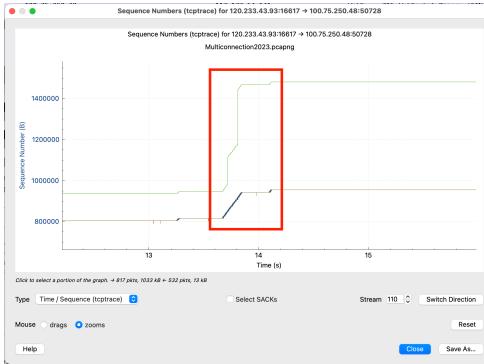
The full-duplex communication: the channel allows simultaneous data transmission in both directions. In the TCP/IP model, this means the client can send data to the server while the server simultaneously sends data to the client, with each direction independently managed and controlled.

Example: Full-duplex communication is common in daily life, with telephone conversations being a classic example. When two people talk over the phone, full-duplex communication allows them to speak and listen at the same time without taking turns. The key here is that both parties can send and receive information simultaneously.

b. The lack of receiver buffer space throttles the sender. This is the flow control mechanism of TCP, which ensures that The sender will not send data that exceeds the receiver's buffer capacity.

Within the tcptrace graph, a diminishing gap between the blue line (sender's sequence numbers) and the yellow line (receiver's acknowledgments) indicates timely ACKs from the receiver, suggesting swift data handling without noticeable delays. A shrinking or stagnant gap between the green line (representative of the receiver's window limit) and the blue line could signify a full receiver buffer, necessitating the sender to halt transmissions until the receiver processes the incoming data and sends a new ACK to update its window size, thereby granting the sender additional space for data transmission.

For a nuanced analysis, I zoomed into a specific part of the graph.



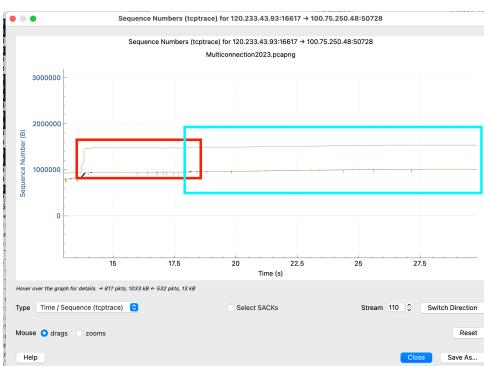
Observe the horizontal stretches within the red box in the graph. These could signify that the sender has paused sending data due to not receiving timely acknowledgments, likely due to receiver buffer restrictions.

c. By analyzing the tcptrace graph, we can evaluate the "smooth" of a TCP connection.

In the tcptrace graph, the horizontal segments of the blue line indicate times when the sender has stopped sending new data, likely due to awaiting acknowledgments from the receiver, thus temporarily halting data transmission.

If these horizontal segments occur frequently and last for extended periods, they may indicate network congestion. The TCP protocol, upon detecting congestion, would reduce its sending rate, causing a stagnation in the sequence numbers.

Brief upward movements between these horizontal segments are indicative of retransmissions by the sender, usually in response to delayed ACKs or perceived packet losses.



The graph exhibits multiple horizontal stretches, which could signal potential network congestion, punctuated by very brief increases in sequence numbers, suggesting retransmissions.

These patterns suggest that the TCP connection may have experienced a certain degree of network congestion during the observed period.

The packet list for the corresponding timeframe supports the same conclusion.

No.	Time	Source	Destination	Protocol	Length
67295	2023-06-30 14:30:43.995900	146.56.196.163	100.75.250.48	TCP	56
67296	2023-06-30 14:30:43.995922	100.75.250.48	146.56.196.163	TCP	54
67297	2023-06-30 14:30:44.032347	100.75.250.48	120.233.43.93	TCP	54
67298	2023-06-30 14:30:44.032366	100.75.250.48	1.12.12.12	TCP	138
67299	2023-06-30 14:30:44.109028	MercuryComm_6f:1b:74	Broadcast	ARP	42
67300	2023-06-30 14:30:44.109072	MercuryComm_6f:1b:74	Broadcast	ARP	42
67301	2023-06-30 14:30:44.139639	100.75.250.48	118.195.149.247	TCP	54
67302	2023-06-30 14:30:44.139640	100.75.250.48	118.195.149.247	TCP	54
67303	2023-06-30 14:30:44.139738	100.75.250.48	118.195.149.247	TCP	54
67304	2023-06-30 14:30:44.142559	120.233.43.93	100.75.250.48	TCP	1514
67305	2023-06-30 14:30:44.170755	100.75.250.48	117.177.54.141	TCP	176
67306	2023-06-30 14:30:44.185860	100.75.250.48	8.8.4.4	TCP	293
67307	2023-06-30 14:30:44.185897	100.75.250.48	120.233.43.93	TCP	54
67308	2023-06-30 14:30:44.185996	MercuryComm_6f:1b:74	Broadcast	ARP	42
67309	2023-06-30 14:30:44.279879	100.75.250.48	118.195.149.247	TCP	54
67310	2023-06-30 14:30:44.279197	100.75.250.48	118.195.149.247	TCP	54
67311	2023-06-30 14:30:44.303397	120.233.43.93	100.75.250.48	TCP	1514
67312	2023-06-30 14:30:44.356423	100.75.250.48	120.233.43.93	TCP	54
67313	2023-06-30 14:30:44.356427	100.75.250.48	117.177.54.141	TCP	113
67314	2023-06-30 14:30:44.371770	100.75.250.48	22.2.0.0	IP	171

d. When a TCP connection requires retransmission, typical actions include packet loss detection, retransmitting lost segments, implementing congestion control measures, adjusting the retransmission timeout (RTO), and temporarily reducing the sending rate.

Common causes of transmission failure are network congestion, unreliable network links, improper timeout configuration, hardware issues, and high latency or jitter.

To mitigate these issues, it's advised to implement Quality of Service (QoS), enhance network reliability, optimize timeout settings, maintain hardware, and manage latency and jitter through effective routing and traffic shaping.

2.6 Q2.6

To calculate the throughput, follow these basic steps:

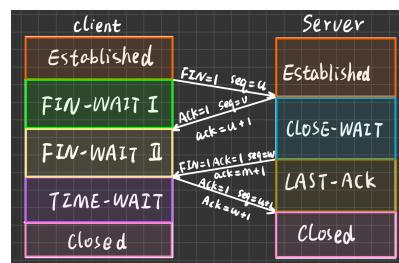
- Determine Data Volume and Time Frame:** Start by identifying the total amount of data transferred over the TCP connection (in bytes) and the time taken for this transfer.
- Collect Data:** Capture TCP packets with Wireshark, noting the amount of data transferred and the timestamps of each packet.
- Calculate Data Volume:** Analyze the captured packets to sum up the payload size (excluding the TCP header) of all TCP packets within the target time frame.
- Measure Time Interval:** Identify the start and end times of the period you're analyzing.
- Compute Throughput:** Divide the total amount of data from step 3 by the total time from step 4.
$$\text{Throughput} = \frac{\text{TotalTransmissionBytes}}{\text{TotalTime}}$$

Additionally, Wireshark offers a direct way to visualize throughput with its built-in graph feature. You can access it through [Statistics -> TCP Stream Graph -> Throughput](#), which generates a graph illustrating the rate of data transfer over time.

2.7 Q2.7

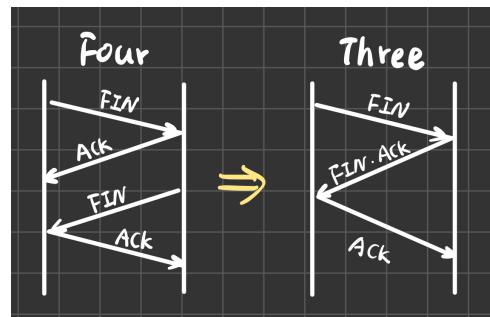
Four-way Handshake Process(Assuming the client initiates the closure):

1. **First Handshake (FIN from Client):** The client sends a FIN (Finish) flag to indicate no more data is sent but can still receive.
2. **Second Handshake (ACK from Server):** The server acknowledges with an ACK flag, indicating receipt of the client's termination request.
3. **Third Handshake (FIN from Server):** After sending all data, the server sends a FIN flag, indicating it has no more data.
4. **Fourth Handshake (ACK from Client):** The client sends an ACK flag in response to the server's FIN and enters TIME_WAIT state to ensure its ACK is received.



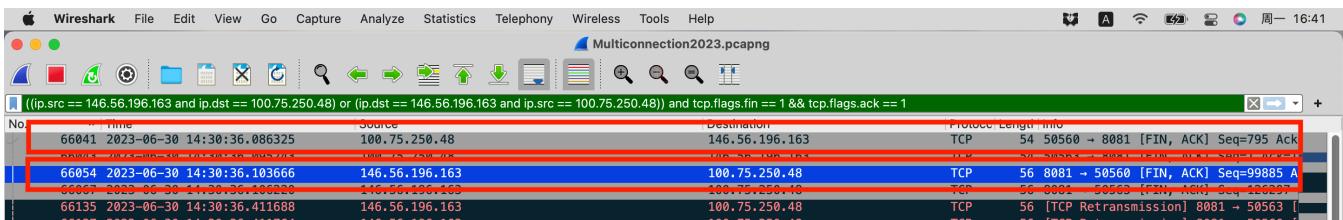
2.8 Q2.8

In certain scenarios, it is indeed possible for the TCP four-way handshake process to appear as a three-way handshake.



, Use the following filter in Wireshark to identify such occurrences:

```
1 | ((ip.src == 146.56.196.163 and ip.dst == 100.75.250.48) or
| (ip.dst == 146.56.196.163 and ip.src == 100.75.250.48))
```



In this case, the second and third steps of the handshake are merged into one, making it appear as though a three-way handshake has occurred.

Such a three-way handshake scenario may arise when the passive closing party (e.g., the server in the illustration) has "no data to send" and "the TCP delayed acknowledgment mechanism is enabled" during the handshake process. The delay in sending ACKs, triggered after receiving the initial FIN, occurs because the conditions for immediate acknowledgment are not met. During this delay, if the application also decides to close the connection and has no more data to send, it triggers the sending of a FIN, which then gets combined with the previous ACK and sent together.

3 Feedback

Using Wireshark to capture and analyze network packets has not only deepened my understanding of how the TCP/IP protocol works but also enhanced my skills in troubleshooting network issues. Through hands-on experience, I was able to observe the transmission process of data packets and understand how TCP maintains reliable data transmission under various network conditions.

The experiment was detailed and complex, presenting many new challenges in terms of concepts and operations, which required me to do extensive research and learning. I realized that there is a significant gap between theoretical knowledge and practical application.

The difficulty of the experiments, especially parts 2.4 and 2.5, was quite high. My unfamiliarity with Wireshark led me to spend a lot of time searching for information online. In total, I spent over 16 hours on this project, with the majority of that time dedicated to familiarizing myself with the various functions of Wireshark and searching for relevant tutorials. During this process, I found that learning to use filters effectively and utilizing Wireshark to generate related charts were particularly challenging.

Based on my experience, I would suggest allocating some time in the classroom for demonstrating and explaining the basic use of Wireshark. This would not only help beginners to grasp the fundamental operations of this powerful tool more quickly but also improve the efficiency and depth of understanding of the experiments. Useful link: [1](#) [2](#) [3](#) [4](#) [5](#)