IK2218 Protocols and Principles of the Internet

Lab 1 ARP, IP, UDP, TCP

Team member: Xianqing Zeng, Yuhang Lin, Saly Al Masri, Alice Wijitchakhorn

Note: During this lab, the IP address of Host A(sender) is 192.168.0.22, and Host B(receiver) is 192.168.0.20.

10 Basic operation of TCP

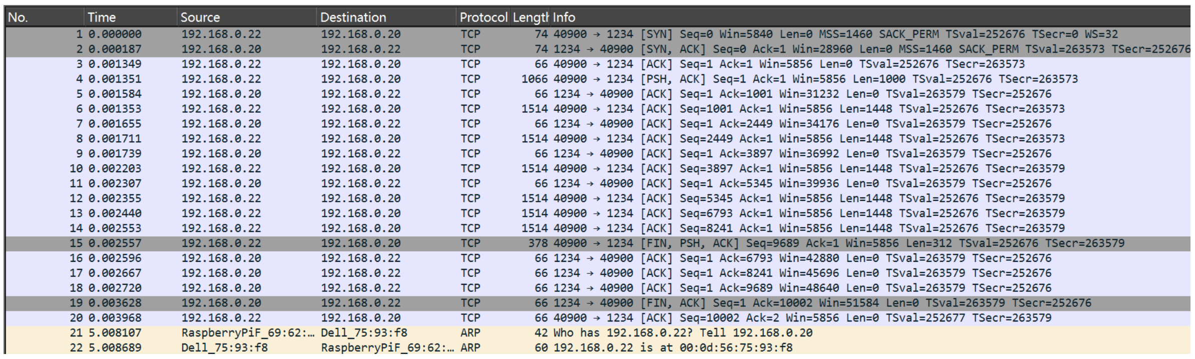
1) How many packets are transmitted in total (count both directions)?
   In total, **22 packets** are transmitted seen in the following screenshot. Out of these, there are 20 TCP packets and 2 ARP packets.

2) What is the range of the sequence numbers used by the sender (Host A)?
   The sequence number range is **0 to 10002**.

3) How many packets do not carry a data payload?
   Packets 1, 2, 3, 5, 7, 9, 11, 16, 17, 18, 19, 20, 21 and 22 (all of these are indicated `Len=0`. Therefore, **14 packets** do not carry a data payload.

4) What is the total number of bytes transmitted in the recorded transfer? From those, calculate the amount of user data that was transmitted?
   According to the `statistics` in Wireshark, the total number of bytes transmitted is **11438**. Among them, there are 8 packets carrying user data, which sum up to $1000 + 1448 * 6 + 312 = 10000$ bytes.

5) Compare the total amount of data transmitted in the TCP data transfer to that of a UDP data transfer. Which of the protocols is more efficient in terms of overhead? What is the efficiency in percentage for these two protocols?
   TCP: $10000/11438 = 87.43\%$
   UDP: $10000/10760 = 92.94\%$
   In comparison, UDP is more efficient in terms of efficiency.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.22	192.168.0.20	TCP	74	40900 → 1234 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM TSval=252676 TSecr=0 WS=32
2	0.000187	192.168.0.20	192.168.0.22	TCP	74	1234 → 40900 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM TSval=263573 TSecr=252676
3	0.001349	192.168.0.22	192.168.0.20	TCP	66	40900 → 1234 [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval=252676 TSecr=263573
4	0.001351	192.168.0.22	192.168.0.20	TCP	1066	40900 → 1234 [PSH, ACK] Seq=1 Ack=1 Win=5856 Len=1000 TSval=252676 TSecr=263573
5	0.001584	192.168.0.20	192.168.0.22	TCP	66	1234 → 40900 [ACK] Seq=1 Ack=1001 Win=31232 Len=0 TSval=263579 TSecr=252676
6	0.001353	192.168.0.22	192.168.0.20	TCP	1514	40900 → 1234 [ACK] Seq=1001 Ack=1 Win=5856 Len=1448 TSval=252676 TSecr=263573
7	0.001655	192.168.0.20	192.168.0.22	TCP	66	1234 → 40900 [ACK] Seq=1 Ack=2449 Win=34176 Len=0 TSval=263579 TSecr=252676
8	0.001711	192.168.0.22	192.168.0.20	TCP	1514	40900 → 1234 [ACK] Seq=2449 Ack=1 Win=5856 Len=1448 TSval=252676 TSecr=263573
9	0.001739	192.168.0.20	192.168.0.22	TCP	66	1234 → 40900 [ACK] Seq=1 Ack=3897 Win=36992 Len=0 TSval=263579 TSecr=252676
10	0.002203	192.168.0.22	192.168.0.20	TCP	1514	40900 → 1234 [ACK] Seq=3897 Ack=1 Win=5856 Len=1448 TSval=252676 TSecr=263579
11	0.002307	192.168.0.20	192.168.0.22	TCP	66	1234 → 40900 [ACK] Seq=1 Ack=5345 Win=39936 Len=0 TSval=263579 TSecr=252676
12	0.002355	192.168.0.22	192.168.0.20	TCP	1514	40900 → 1234 [ACK] Seq=5345 Ack=1 Win=5856 Len=1448 TSval=252676 TSecr=263579
13	0.002440	192.168.0.22	192.168.0.20	TCP	1514	40900 → 1234 [ACK] Seq=6793 Ack=1 Win=5856 Len=1448 TSval=252676 TSecr=263579
14	0.002553	192.168.0.22	192.168.0.20	TCP	1514	40900 → 1234 [ACK] Seq=8241 Ack=1 Win=5856 Len=1448 TSval=252676 TSecr=263579
15	0.002557	192.168.0.22	192.168.0.20	TCP	378	40900 → 1234 [FIN, PSH, ACK] Seq=9689 Ack=1 Win=5856 Len=312 TSval=252676 TSecr=263579
16	0.002596	192.168.0.20	192.168.0.22	TCP	66	1234 → 40900 [ACK] Seq=1 Ack=6793 Win=42880 Len=0 TSval=263579 TSecr=252676
17	0.002667	192.168.0.20	192.168.0.22	TCP	66	1234 → 40900 [ACK] Seq=1 Ack=8241 Win=45696 Len=0 TSval=263579 TSecr=252676
18	0.002720	192.168.0.20	192.168.0.22	TCP	66	1234 → 40900 [ACK] Seq=1 Ack=9689 Win=48640 Len=0 TSval=263579 TSecr=252676
19	0.003628	192.168.0.20	192.168.0.22	TCP	66	1234 → 40900 [FIN, ACK] Seq=1 Ack=10002 Win=51584 Len=0 TSval=263579 TSecr=252676
20	0.003968	192.168.0.22	192.168.0.20	TCP	66	40900 → 1234 [ACK] Seq=10002 Ack=2 Win=5856 Len=0 TSval=252677 TSecr=263579
21	5.008107	RaspberryPiF_69:62:…	Dell_75:93:f8	ARP	42	Who has 192.168.0.22? Tell 192.168.0.20
22	5.008689	Dell_75:93:f8	RaspberryPiF_69:62:…	ARP	60	192.168.0.22 is at 00:0d:56:75:93:f8

11 TCP connection management

11.1 Connection establishment and termination

1) Which packets constitute the three-way handshake? Which flags are set in the headers of these packets?
   The first three packets (**1, 2, and 3**) constitute the TCP three-way handshake, and the flags are set to be **`SYN`**, **`SYN, ACK`**, **`ACK`** respectively.

2) What are the initial sequence numbers used by the client and the server, respectively?
   According to the `Seq` and `Ack` number, the initial sequence numbers used by the client and the server are both **0**.

3) Which packet contains the first application data?
   The first application data is sent in **Packet 15**, which is a TELNET packet with 22 bytes of data.

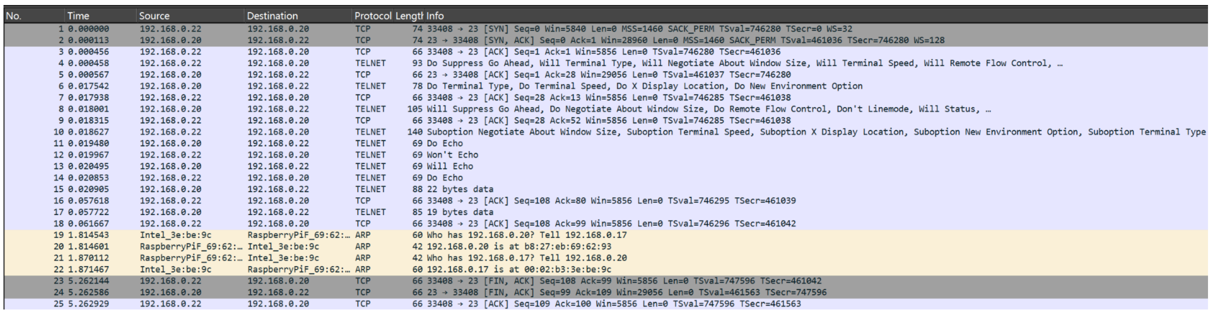
4) What are the initial window sizes for the client and for the server?
   Client (Host A): The initial window size is **5840** (seen in Packet 1, SYN).
   Server (Host B): The initial window size is **28960** (seen in Packet 2, SYN-ACK).

5) How long does it roughly take to open the TCP connection?
   The TCP connection is established by the three-way handshake, starting with the SYN (Packet 1) at **0.000000 seconds** and completing with the ACK (Packet 3) at **0.000456 seconds**. Therefore, it takes approximately **0.000456 seconds** to open the TCP connection.

6) Which packets are involved in closing the connection?
   Packet 23: The client sends a FIN, ACK to initiate the connection termination.
   Packet 24: The server responds with a FIN, ACK.
   Packet 25: The client sends an ACK to acknowledge the server's FIN.

7) Which flags are set in these packets?
   Packet 23: FIN, ACK
   Packet 24: FIN, ACK

Packet 25: ACK

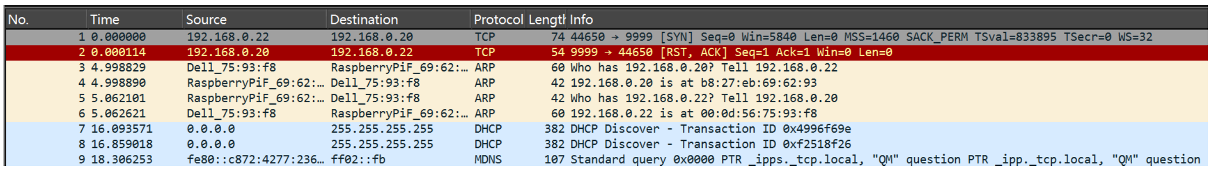


```
No.  Time      Source             Destination        Protocol Length Info
   1 0.000000  192.168.0.22       192.168.0.20       TCP       74 33408 → 23 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM TSval=746280 TSecr=0 WS=32
   2 0.000113  192.168.0.20       192.168.0.22       TCP       74 23 → 33408 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM TSval=461036 TSecr=746280 WS=128
   3 0.000456  192.168.0.22       192.168.0.20       TCP       66 33408 → 23 [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval=746280 TSecr=461036
   4 0.000458  192.168.0.22       192.168.0.20       TELNET    93 Do Suppress Go Ahead, Will Terminal Type, Will Negotiate About Window Size, Will Terminal Speed, Will Remote Flow Control, …
   5 0.000567  192.168.0.20       192.168.0.22       TCP       66 23 → 33408 [ACK] Seq=1 Ack=28 Win=29056 Len=0 TSval=461037 TSecr=746280
   6 0.017542  192.168.0.20       192.168.0.22       TELNET    78 Do Terminal Type, Do Terminal Speed, Do X Display Location, Do New Environment Option
   7 0.017938  192.168.0.22       192.168.0.20       TCP       66 33408 → 23 [ACK] Seq=28 Ack=13 Win=5856 Len=0 TSval=746285 TSecr=461038
   8 0.018001  192.168.0.20       192.168.0.22       TELNET   105 Will Suppress Go Ahead, Do Negotiate About Window Size, Do Remote Flow Control, Don't Linemode, Will Status, …
   9 0.018315  192.168.0.22       192.168.0.20       TCP       66 33408 → 23 [ACK] Seq=28 Ack=52 Win=5856 Len=0 TSval=746285 TSecr=461038
  10 0.018627  192.168.0.22       192.168.0.20       TELNET   140 Suboption Negotiate About Window Size, Suboption Terminal Speed, Suboption X Display Location, Suboption New Environment Option, Suboption Terminal Type
  11 0.019480  192.168.0.20       192.168.0.22       TELNET    69 Do Echo
  12 0.019967  192.168.0.22       192.168.0.20       TELNET    69 Won't Echo
  13 0.020495  192.168.0.20       192.168.0.22       TELNET    69 Will Echo
  14 0.020853  192.168.0.22       192.168.0.20       TELNET    69 Do Echo
  15 0.020905  192.168.0.20       192.168.0.22       TELNET    88 22 bytes data
  16 0.057618  192.168.0.22       192.168.0.20       TCP       66 33408 → 23 [ACK] Seq=108 Ack=80 Win=5856 Len=0 TSval=746295 TSecr=461039
  17 0.057722  192.168.0.20       192.168.0.22       TELNET    85 19 bytes data
  18 0.061667  192.168.0.22       192.168.0.20       TCP       66 33408 → 23 [ACK] Seq=108 Ack=99 Win=5856 Len=0 TSval=746296 TSecr=461042
  19 1.814543  Intel_3e:be:9c     RaspberryPiF_69:62:… ARP     60 Who has 192.168.0.20? Tell 192.168.0.17
  20 1.814601  RaspberryPiF_69:62:… Intel_3e:be:9c   ARP       42 192.168.0.20 is at b8:27:eb:69:62:93
  21 1.870112  RaspberryPiF_69:62:… Intel_3e:be:9c   ARP       42 Who has 192.168.0.17? Tell 192.168.0.20
  22 1.871467  Intel_3e:be:9c     RaspberryPiF_69:62:… ARP     60 192.168.0.17 is at 00:02:b3:3e:be:9c
  23 5.262144  192.168.0.22       192.168.0.20       TCP       66 33408 → 23 [FIN, ACK] Seq=108 Ack=99 Win=5856 Len=0 TSval=747596 TSecr=461042
  24 5.262586  192.168.0.20       192.168.0.22       TCP       66 23 → 33408 [FIN, ACK] Seq=99 Ack=109 Win=29056 Len=0 TSval=461563 TSecr=747596
  25 5.262929  192.168.0.22       192.168.0.20       TCP       66 33408 → 23 [ACK] Seq=109 Ack=100 Win=5856 Len=0 TSval=747596 TSecr=461563
```

11.2 Connecting to a non-existing port

1) How does the server host (Host B) close the connection?
   The server host (Host B) closes the connection by sending a RST, ACK packet, and **RST (Reset)** flag indicates that the port the client is trying to connect to is either closed or does not exist.

2) How long does the process of ending the connection take?
   The process of ending the connection takes approximately **0.000114** seconds (pkg2 -pkg1).



```
No.  Time       Source                 Destination            Protocol Length Info
   1 0.000000   192.168.0.22           192.168.0.20           TCP       74 44650 → 9999 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM TSval=833895 TSecr=0 WS=32
   2 0.000114   192.168.0.20           192.168.0.22           TCP       54 9999 → 44650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   3 4.998829   Dell_75:93:f8          RaspberryPiF_69:62:…   ARP       60 Who has 192.168.0.20? Tell 192.168.0.22
   4 4.998890   RaspberryPiF_69:62:…   Dell_75:93:f8          ARP       42 192.168.0.20 is at b8:27:eb:69:62:93
   5 5.062101   RaspberryPiF_69:62:…   Dell_75:93:f8          ARP       42 Who has 192.168.0.22? Tell 192.168.0.20
   6 5.062621   Dell_75:93:f8          RaspberryPiF_69:62:…   ARP       60 192.168.0.22 is at 00:0d:56:75:93:f8
   7 16.093571  0.0.0.0                255.255.255.255        DHCP     382 DHCP Discover - Transaction ID 0x4996f69e
   8 16.859018  0.0.0.0                255.255.255.255        DHCP     382 DHCP Discover - Transaction ID 0xf2518f26
   9 18.306253  fe80::c872:4277:236…   ff02::fb               MDNS     107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
```

12 TCP data transfer

12.1 Interactive application

1) Describe the payload of each packet.

   From the TCP packets in the file 12_1, specifically frame 4 (`0.150932s`), we can see the **TCP Segment Len** as **27 bytes**. This corresponds to the payload (telnet options) that are being sent. To view the actual content, you would inspect the **TCP payload** field under each frame in Wireshark, where small character-based payloads are typical in telnet communications. Here, frame 4 contains data related to terminal negotiations.

   In general

   - **Len = 0**: Indicates packets without data, usually part of the **TCP handshake** (e.g., SYN, ACK) or pure **ACKs** acknowledging received data.

- **Len = 1**: Represents **control characters** (e.g., backspace, line feed) that are often sent separately in telnet.
- **Len = 3**: Small **data segments**, likely representing individual characters typed during the telnet session, or responses echoing each typed character back.
- **Larger Segment Lengths (e.g., 27 or 74)**: These represent more substantial **data packets**, such as **telnet option negotiations** or the bundling of multiple characters into one packet for efficiency.

2) Explain why you do not see four packets per typed character.

Three Packets Instead of Four:

- Normally, we expect four packets: send, ACK, echo, ACK. However, in this capture, there are only three packets per character exchange:
  - Frame 4 (sending character)
  - Frame 5 (server ACK)
  - Frame 6 (server response with echo)
- The reason for seeing three packets instead of four is **delayed ACK**, the client is holding off on sending the acknowledgment right after the echo (which would normally be the fourth packet). Instead, TCP combines this acknowledgment with the next data transmission from the client or waits for the **delayed ACK timer** to expire before sending an ACK.

3) When the client receives the echo, it waits a certain time before sending the ACK. Why? How long is the delay?

To calculate the delay we need to to calculate the difference in time between when the data has been sent and when is the ACK sent, the average time is 150 ms, I will go through some examples to show the values:

Frame 18:

- **Data Sent**: Time = 2.492555 seconds (from 192.168.0.20 to 192.168.0.22)
- **Acknowledgment**: Frame 19, Time = 2.643180 seconds (from 192.168.0.22 to 192.168.0.20)

Delay: 2.643180−2.492555=0.150625 seconds

Frame 48:

- **Data Sent**: Time = 7.496842 seconds
- **Acknowledgment**: Frame 49, Time = 7.647270 seconds

Delay: 7.647270 - 7.496842= 0.150428 seconds

Frame 74:

- **Data Sent**: Time = 24.594591 seconds
- **Acknowledgment**: Frame 75, 24.745405 seconds

Delay: 24.745405 - 24.594591= 0.150 814 seconds

4) In the segments that carry characters, what window size is advertised by the telnet client and by the server? Does the window size vary as the connection proceeds?

   In the segments carrying characters, the Telnet client advertises a window size of 5856 bytes, as seen in frames 4 and 10, allowing it to receive more data before needing an acknowledgment. The server, on the other hand, advertises a window size of 29056 bytes in frames 62 and 66, indicating it can handle a larger amount of incoming data. The window size varies slightly as the connection proceeds. The client starts with a window size of 5840 bytes, which increases slightly to 5856 bytes, showing stable behavior. Meanwhile, the server's window size grows from 28960 bytes to 29056 bytes, remaining large and consistent throughout the connection.

5) Do you observe a difference in the transmission of segment payloads and ACKs?

   There is a clear difference between the transmission of segment payloads and acknowledgments (ACKs) in the Telnet capture. The segments carrying Telnet data vary in size, with some frames transmitting 1 byte of data, while others transmit 3, 8, or even 12 bytes, reflecting the small chunks of data typically sent by Telnet. In contrast, the ACK segments consistently contain no data (payload length of 0), and their purpose is solely to acknowledge the receipt of the previous segments. The timing between the data transmissions and the ACKs is prompt, indicating efficient flow control between the sender and receiver.

12.2 Bulk transfer

1) How often does the receiver send ACKs? Can you see a rule on how TCP sends ACKs?
   Based on the analysis of the frames and the TCP sequence and acknowledgment numbers, we can conclude that the receiver sends ACKs for

every **two full-sized segments** received, which aligns with TCP’s **delayed acknowledgment** mechanism.

The receiver sends ACKs after receiving two full-sized segments of **1448 bytes**. For example, in **Frame 5**, the receiver sends an ACK for sequence number **1001**, confirming receipt of the first segment. In **Frame 6**, the sender transmits 1448 bytes, and in **Frame 7**, the receiver acknowledges sequence **2449**. Similarly, after another 1448 bytes in **Frame 8**, the receiver sends an ACK for **3897** in **Frame 9**. This pattern aligns with TCP's **delayed acknowledgment** mechanism, where the receiver waits for two full segments before sending an ACK, improving efficiency by reducing the number of ACKs.

2) How many bytes of data does a receiver acknowledge in a typical ACK?

   The receiver acknowledges the number of bytes cumulatively, typically acknowledging **1448 bytes** per segment, which is the maximum segment size (MSS) for TCP. For example, **Frame 5** acknowledges sequence number **1001** after receiving 1448 bytes (1 MSS). Similarly, in **Frame 7**, the receiver acknowledges up to **2449** bytes after receiving another 1448 bytes (another MSS). If multiple segments are received, the acknowledgment number increases accordingly, always reflecting the total bytes received.

3) How does the window size vary during the session?
   The window size changes throughout the session, increasing as the receiver processes and acknowledges more data. For example, in **Frame 1**, the window size is **5840 bytes**, and in **Frame 5**, it slightly increases to **5856 bytes**. By **Frame 9**, it grows significantly to **36992 bytes** as more data is buffered and the receiver can accept more. This trend continues as the session progresses, reflecting the receiver's ability to handle additional data as its buffer space frees up.

4) Select any ACK packet in the Wireshark trace and note its acknowledgement number. Find the original segment in the Wireshark output. How long did it take from the transmission until it was ACKed?

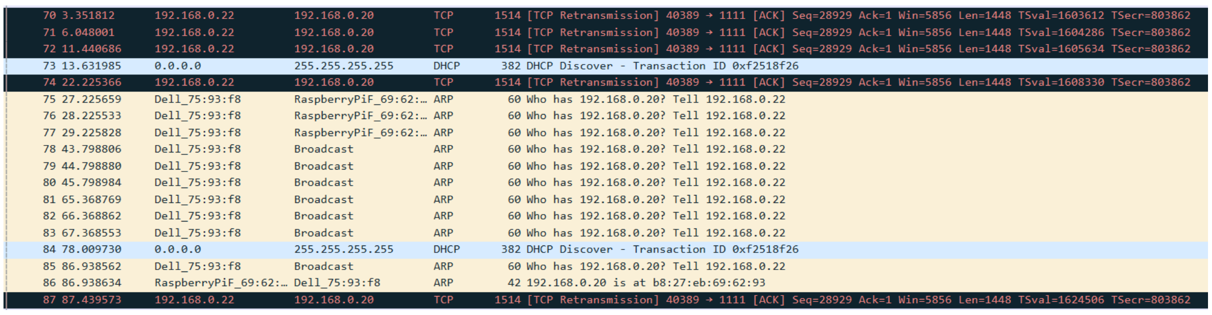
   In **Frame 5**, the acknowledgment number is **1001**, acknowledging the segment sent in **Frame 4**. The time difference between these frames is **0.000057 seconds**, which reflects local processing delay, not network delay. Similarly, in **Frame 7**, the acknowledgment number is **2449**, acknowledging the data sent in **Frame 6**, with a time difference of **0.000168 seconds**. While these differences seem very short, the **150 ms network delay** would be observed in round-trip times when the packets actually cross the network between hosts.

5) Does the TCP sender generally transmit the maximum number of bytes as allowed by the receiver?
 Yes, the sender generally transmits close to the maximum number of bytes allowed by the receiver, which is typically **1448 bytes** (the maximum segment size for Ethernet). For example, in frames 6, 8, and 10, the sender transmits **1448 bytes** each time, matching the MSS for this connection.

13 TCP retransmissions

1) How many packets are transmitted at retransmission timeout?

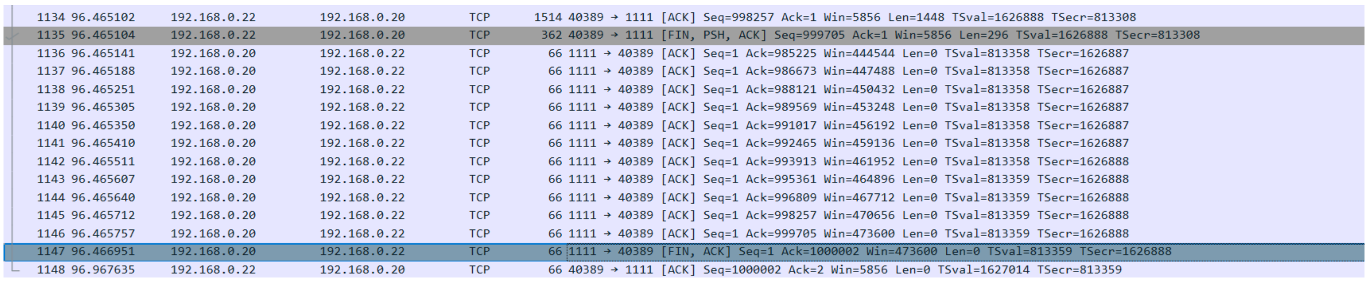
   The retransmission timeout led to the repeated transmission of the same packet with sequence number **28929**. In the trace, this packet was retransmitted five times as indicated in frames 70, 71, 72, 74 and 87 all labeled as **TCP Retransmission**. These retransmissions occurred because the sender did not receive an acknowledgment for the original packet, so it attempted to send the same data again at each timeout. In total, **five packets were transmitted** during the retransmission timeout for this sequence number.



```
70 3.351812    192.168.0.22         192.168.0.20         TCP   1514 [TCP Retransmission] 40389 → 1111 [ACK] Seq=28929 Ack=1 Win=5856 Len=1448 TSval=1603612 TSecr=803862
71 6.048001    192.168.0.22         192.168.0.20         TCP   1514 [TCP Retransmission] 40389 → 1111 [ACK] Seq=28929 Ack=1 Win=5856 Len=1448 TSval=1604286 TSecr=803862
72 11.440686   192.168.0.22         192.168.0.20         TCP   1514 [TCP Retransmission] 40389 → 1111 [ACK] Seq=28929 Ack=1 Win=5856 Len=1448 TSval=1605634 TSecr=803862
73 13.631985   0.0.0.0              255.255.255.255      DHCP   382 DHCP Discover - Transaction ID 0xf2518f26
74 22.225366   192.168.0.22         192.168.0.20         TCP   1514 [TCP Retransmission] 40389 → 1111 [ACK] Seq=28929 Ack=1 Win=5856 Len=1448 TSval=1608330 TSecr=803862
75 27.225659   Dell_75:93:f8        RaspberryPiF_69:62:… ARP     60 Who has 192.168.0.20? Tell 192.168.0.22
76 28.225533   Dell_75:93:f8        RaspberryPiF_69:62:… ARP     60 Who has 192.168.0.20? Tell 192.168.0.22
77 29.225828   Dell_75:93:f8        RaspberryPiF_69:62:… ARP     60 Who has 192.168.0.20? Tell 192.168.0.22
78 43.798806   Dell_75:93:f8        Broadcast            ARP     60 Who has 192.168.0.20? Tell 192.168.0.22
79 44.798880   Dell_75:93:f8        Broadcast            ARP     60 Who has 192.168.0.20? Tell 192.168.0.22
80 45.798984   Dell_75:93:f8        Broadcast            ARP     60 Who has 192.168.0.20? Tell 192.168.0.22
81 65.368769   Dell_75:93:f8        Broadcast            ARP     60 Who has 192.168.0.20? Tell 192.168.0.22
82 66.368862   Dell_75:93:f8        Broadcast            ARP     60 Who has 192.168.0.20? Tell 192.168.0.22
83 67.368553   Dell_75:93:f8        Broadcast            ARP     60 Who has 192.168.0.20? Tell 192.168.0.22
84 78.009730   0.0.0.0              255.255.255.255      DHCP   382 DHCP Discover - Transaction ID 0xf2518f26
85 86.938562   Dell_75:93:f8        Broadcast            ARP     60 Who has 192.168.0.20? Tell 192.168.0.22
86 86.938634   RaspberryPiF_69:62:… Dell_75:93:f8        ARP     42 192.168.0.20 is at b8:27:eb:69:62:93
87 87.439573   192.168.0.22         192.168.0.20         TCP   1514 [TCP Retransmission] 40389 → 1111 [ACK] Seq=28929 Ack=1 Win=5856 Len=1448 TSval=1624506 TSecr=803862
```

2) Do the retransmissions end at some point?
   The retransmissions eventually stop, as shown by the exchange of **FIN** and **ACK** packets between the hosts, indicating that the transmission completed successfully once network stability was restored. In Frame 1135, **192.168.0.22** sends a **FIN, PSH, ACK**, signaling the end of data transmission. **192.168.0.20** responds with a **FIN, ACK** in Frame 1147, confirming receipt and readiness to close the connection. The **ACK** frames between these **FIN**

packets (Frames 1136-1146) acknowledge previously received data, ensuring both sides are synchronized before the connection is closed.



```
1134 96.465102   192.168.0.22   192.168.0.20   TCP   1514 40389 → 1111 [ACK] Seq=998257 Ack=1 Win=5856 Len=1448 TSval=1626888 TSecr=813308
1135 96.465104   192.168.0.22   192.168.0.20   TCP    362 40389 → 1111 [FIN, PSH, ACK] Seq=999705 Ack=1 Win=5856 Len=296 TSval=1626888 TSecr=813308
1136 96.465141   192.168.0.20   192.168.0.22   TCP     66 1111 → 40389 [ACK] Seq=1 Ack=985225 Win=444544 Len=0 TSval=813358 TSecr=1626887
1137 96.465188   192.168.0.20   192.168.0.22   TCP     66 1111 → 40389 [ACK] Seq=1 Ack=986673 Win=447488 Len=0 TSval=813358 TSecr=1626887
1138 96.465251   192.168.0.20   192.168.0.22   TCP     66 1111 → 40389 [ACK] Seq=1 Ack=988121 Win=450432 Len=0 TSval=813358 TSecr=1626887
1139 96.465305   192.168.0.20   192.168.0.22   TCP     66 1111 → 40389 [ACK] Seq=1 Ack=989569 Win=453248 Len=0 TSval=813358 TSecr=1626887
1140 96.465350   192.168.0.20   192.168.0.22   TCP     66 1111 → 40389 [ACK] Seq=1 Ack=991017 Win=456192 Len=0 TSval=813358 TSecr=1626887
1141 96.465410   192.168.0.20   192.168.0.22   TCP     66 1111 → 40389 [ACK] Seq=1 Ack=992465 Win=459136 Len=0 TSval=813358 TSecr=1626887
1142 96.465511   192.168.0.20   192.168.0.22   TCP     66 1111 → 40389 [ACK] Seq=1 Ack=993913 Win=461952 Len=0 TSval=813358 TSecr=1626888
1143 96.465607   192.168.0.20   192.168.0.22   TCP     66 1111 → 40389 [ACK] Seq=1 Ack=995361 Win=464896 Len=0 TSval=813359 TSecr=1626888
1144 96.465640   192.168.0.20   192.168.0.22   TCP     66 1111 → 40389 [ACK] Seq=1 Ack=996809 Win=467712 Len=0 TSval=813359 TSecr=1626888
1145 96.465712   192.168.0.20   192.168.0.22   TCP     66 1111 → 40389 [ACK] Seq=1 Ack=998257 Win=470656 Len=0 TSval=813359 TSecr=1626888
1146 96.465757   192.168.0.20   192.168.0.22   TCP     66 1111 → 40389 [ACK] Seq=1 Ack=999705 Win=473600 Len=0 TSval=813359 TSecr=1626888
1147 96.466951   192.168.0.20   192.168.0.22   TCP     66 1111 → 40389 [FIN, ACK] Seq=1 Ack=1000002 Win=473600 Len=0 TSval=813359 TSecr=1626888
1148 96.967635   192.168.0.22   192.168.0.20   TCP     66 40389 → 1111 [ACK] Seq=1000002 Ack=2 Win=5856 Len=0 TSval=1627014 TSecr=813359
```

14 TCP congestion control

1) Try to observe periods when TCP sender is in slow start phase and when the sender switches to congestion avoidance. Verify if the congestion window follows the rule of the slow-start phase.

   In the initial stages of transmission, TCP enters the slow start phase, where the congestion window (cwnd) grows exponentially as ACKs are received, leading to rapid increases in sequence numbers. For example, sequence numbers increase quickly in Frames 3, 5, and 7, each time by 1448 bytes, reflecting the exponential growth characteristic of slow start.

   Packet loss is detected around Frame 41 through repeated ACKs for sequence 26033, leading to a fast retransmission in Frame 57. This triggers a switch to congestion avoidance, where the growth of the cwnd becomes more linear. In Frames 61 and onward, the sender's pace slows down, showing smaller, measured increases in sequence numbers, which reflects the more conservative approach of the congestion avoidance phase.

2) Can you find occurrences of fast recovery?

   Fast recovery occurs when TCP detects packet loss through three duplicate ACKs instead of a timeout, allowing it to adjust its transmission rate without fully returning to slow start. In the trace, Frame 45 shows the first duplicate ACK for sequence number 26033, followed by multiple duplicate ACKs in Frames 46-56. This signals packet loss. By Frame 57, the sender triggers a "TCP Fast Retransmission" for the missing segment, entering the fast recovery phase. During this phase, the congestion window is reduced but not reset entirely, allowing transmission to continue at a slower pace. From Frame 61 onward, the sender resumes sending data, with sequence numbers increasing more gradually by 1448 bytes, indicating a controlled recovery from the packet loss.