Group Activity

Activity 1

Archit: Imagine we're moving kitchen items quickly because the truck's arriving soon. We'd just throw everything into boxes without much care—this is like UDP, where speed matters more than reliability.

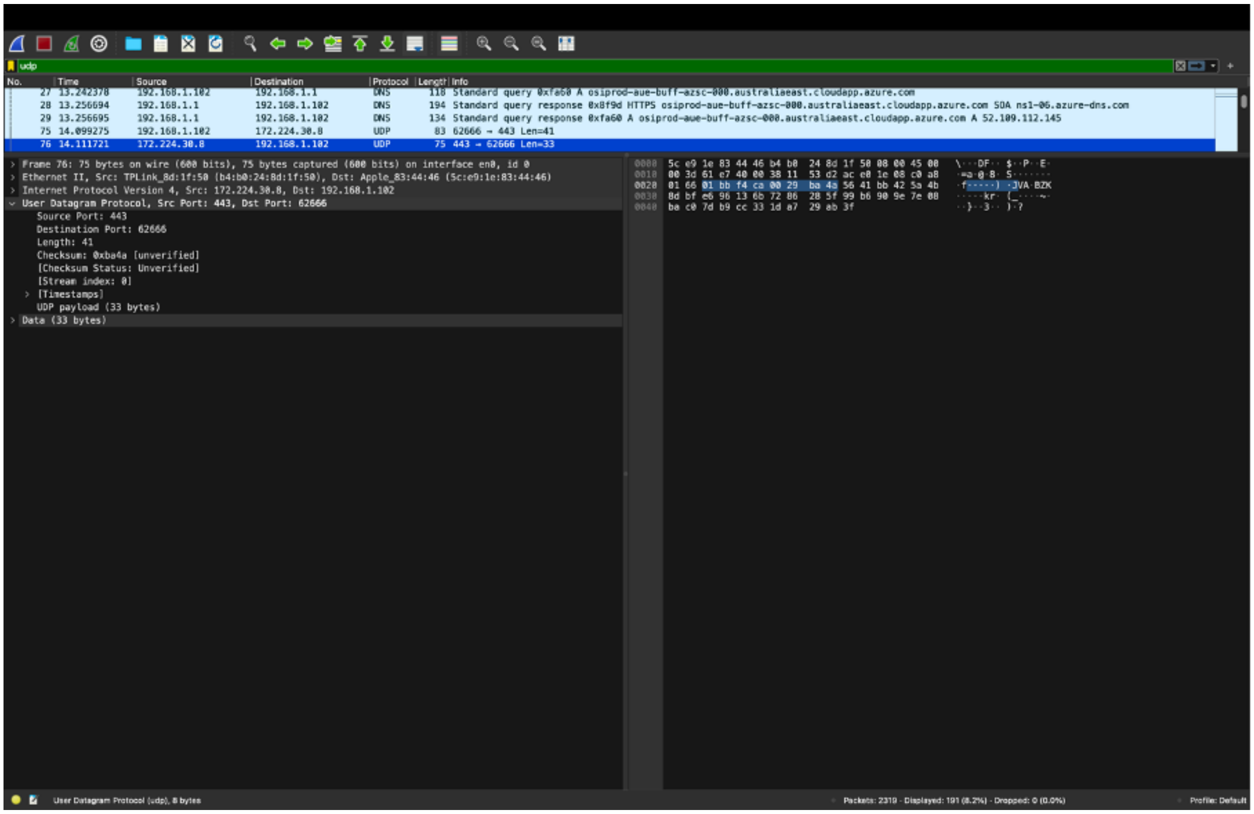
Pranika: Exactly, UDP is fast but doesn't guarantee that everything arrives safely or in order, like streaming videos.

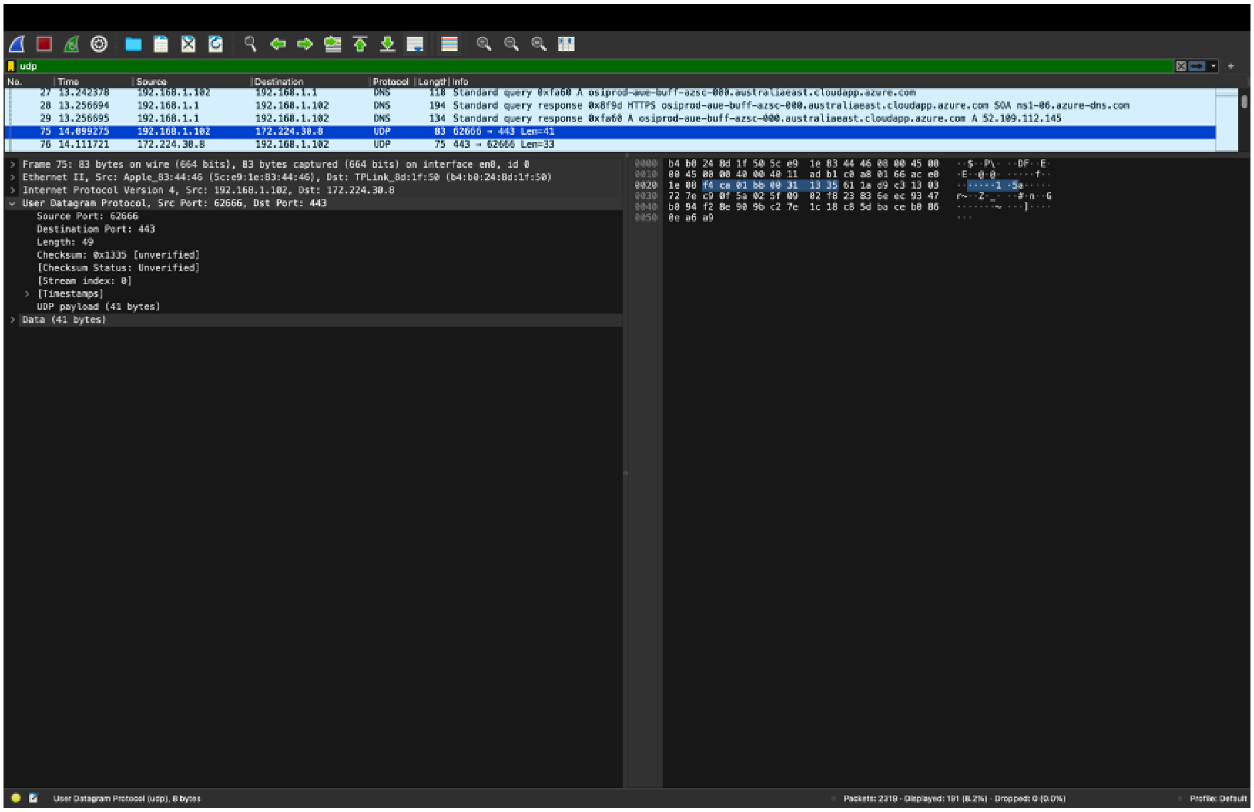
Gitanshi: But if we have more time and care about the order and safety of each item, we'd use TCP—carefully packing and labeling everything.

Jasveena: Yes, TCP ensures everything arrives intact and in the right order, like downloading large files.

Nadia: So, UDP for speed, TCP for reliability. Different scenarios, different protocols!

Activity 2





UDP Header Fields:

- Fields: Source Port, Destination Port, Length, Checksum.
- Each Field Length: 2 bytes.
- Total Header Length: 8 bytes.

Length Field:

Indicates: Total length of UDP header + payload.

- Screenshot 1: Length = 49 bytes (8-byte header + 41-byte payload).
- Screenshot 2: Length = 41 bytes (8-byte header + 33-byte payload).

Maximum UDP Payload:

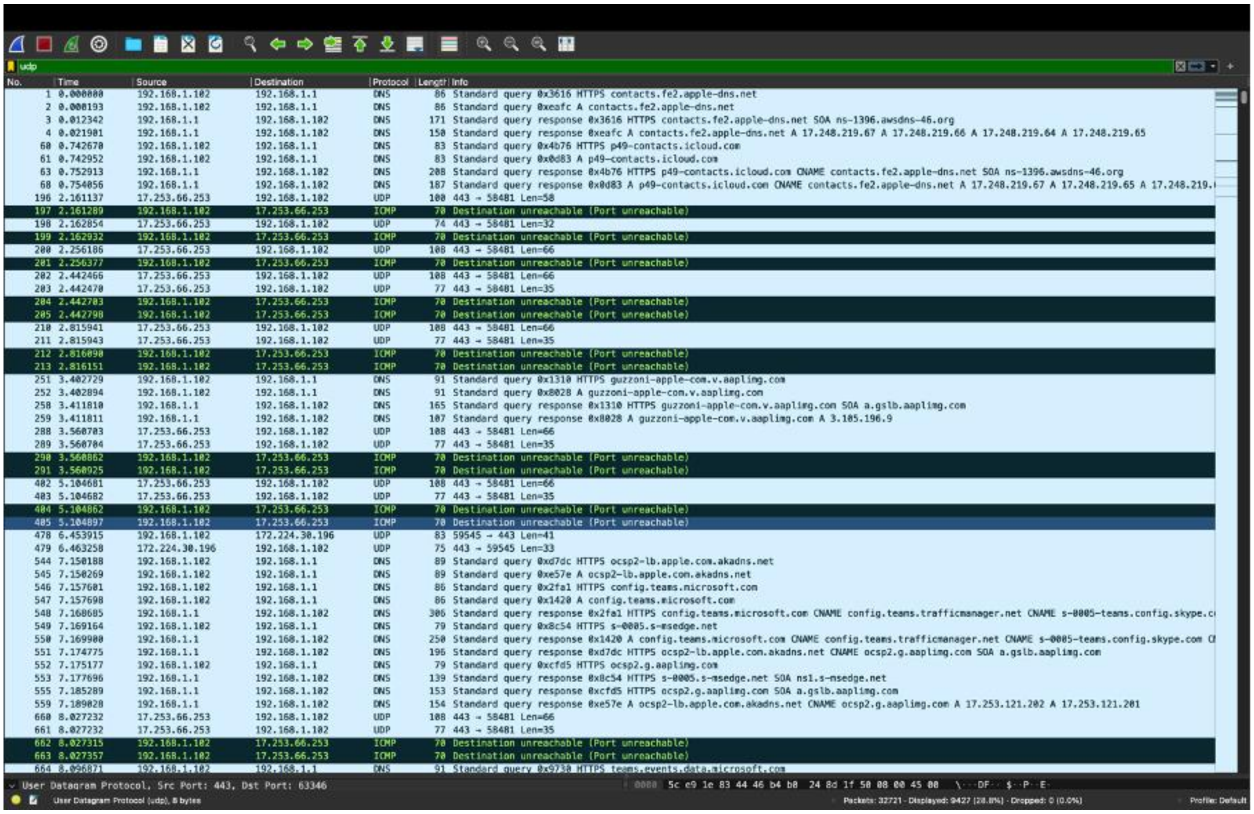
- Theoretical Max: 65,507 bytes.
- UDP Protocol Number:
- Number: 17.

Consecutive UDP Packets:

- Relationship: First packet (source port 62666, destination port 443) is likely a request; second packet (source port 443, destination port 62666) is the response.

Port Numbers Across UDP Packets:

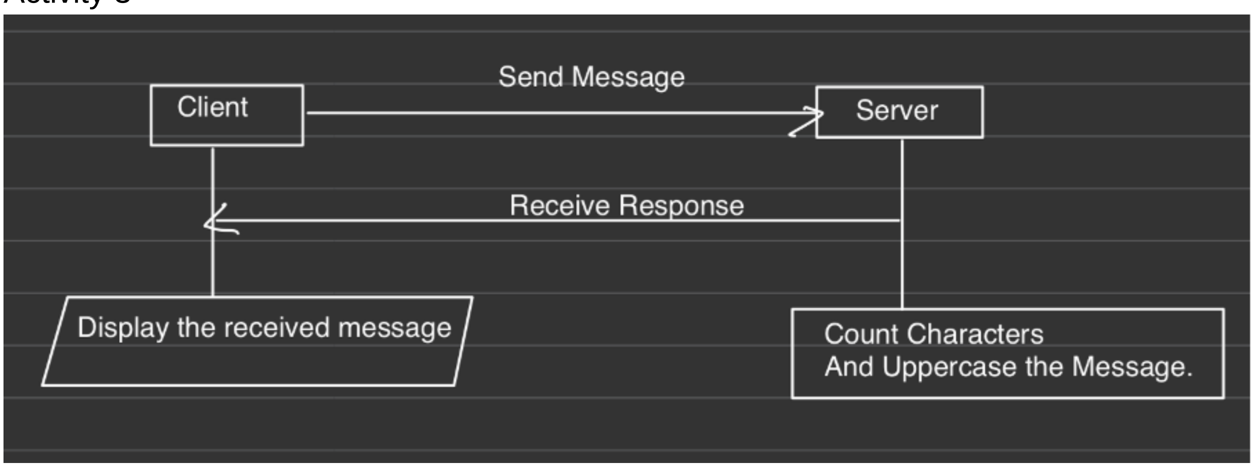
- Varies: Different applications/sessions use different port numbers.

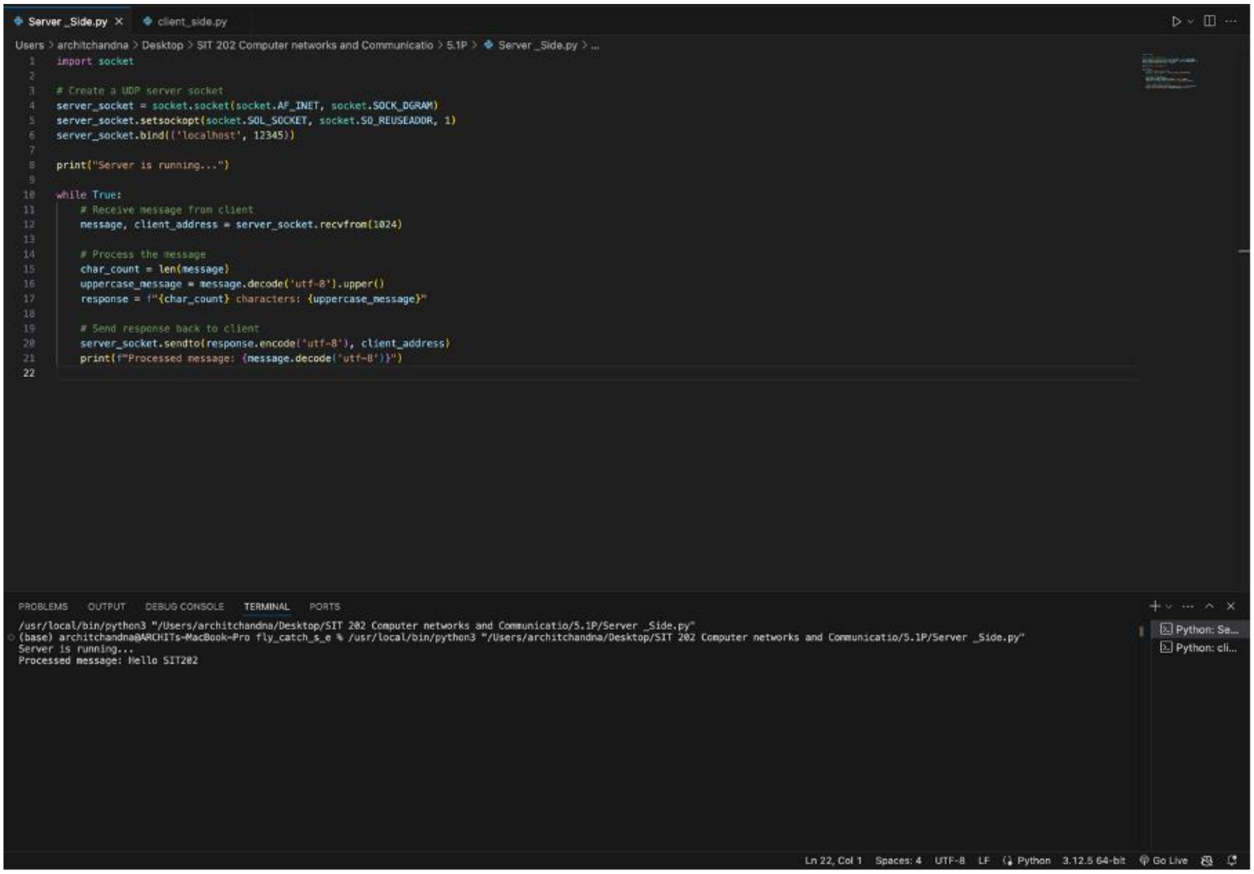


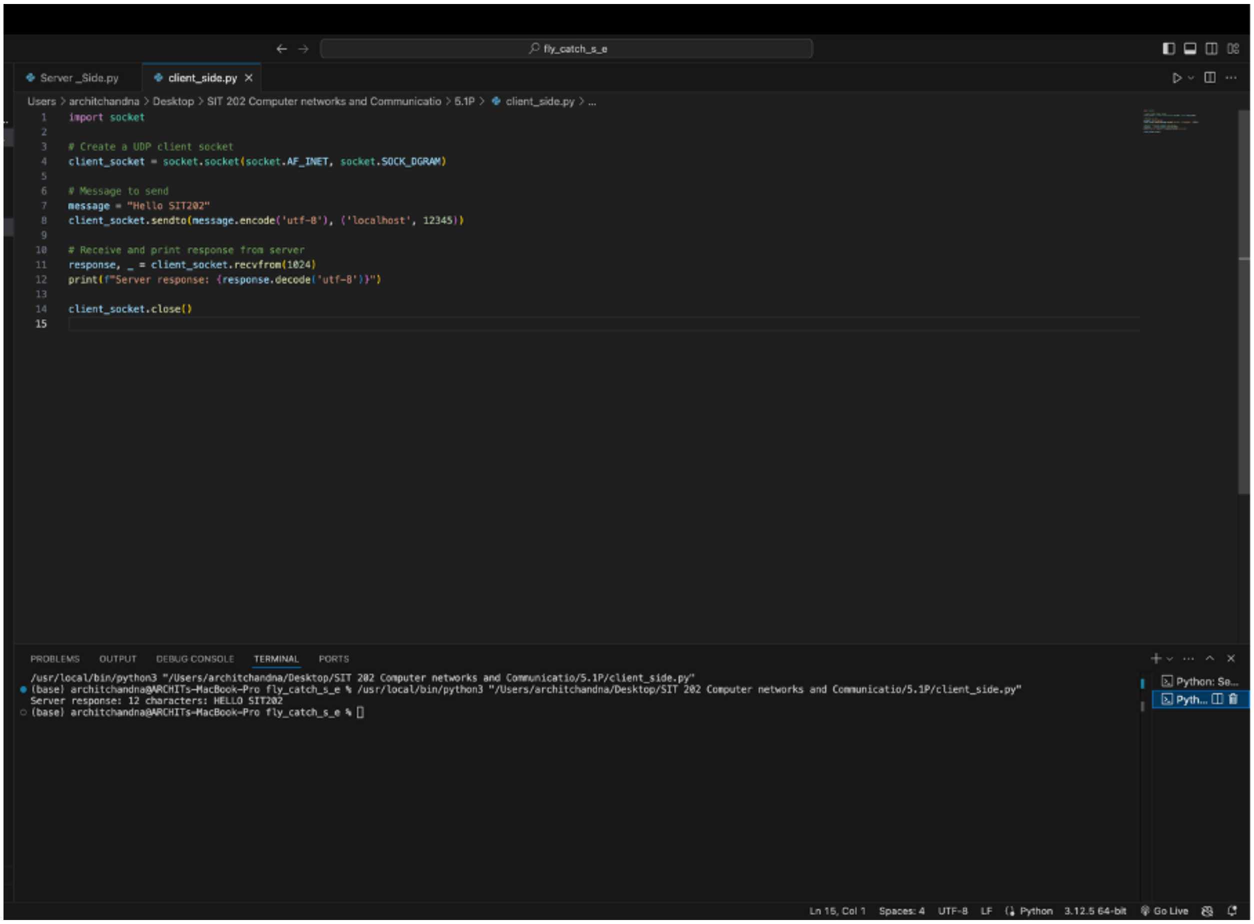
- Observation: UDP packets captured use different port numbers.
- Example: Ports like 58481, 59545, and 63346 are used.
- Explanation: Different applications and sessions use different ports to manage separate data streams.

This confirms that UDP packets from various applications like MSTeams and web browsing don't share the same port numbers.

Activity 3




Server _Side.py
client_side.py
Users > architchandna > Desktop > SIT 202 Computer networks and Communicatio > 5.1P > Server _Side.py > ...
import socket
# Create a UDP server socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind(('localhost', 12345))
print("Server is running...")
while True:
# Receive message from client
message, client_address = server_socket.recvfrom(1024)
# Process the message
char_count = len(message)
uppercase_message = message.decode('utf-8').upper()
response = f"{char_count} characters: {uppercase_message}"
# Send response back to client
server_socket.sendto(response.encode('utf-8'), client_address)
print(f"Processed message: {message.decode('utf-8')}")
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
/usr/local/bin/python3 "/Users/architchandna/Desktop/SIT 202 Computer networks and Communicatio/5.1P/Server _Side.py"
(base) architchandna@ARCHITs-MacBook-Pro fly_catch_s_e % /usr/local/bin/python3 "/Users/architchandna/Desktop/SIT 202 Computer networks and Communicatio/5.1P/Server _Side.py"
Server is running...
Processed message: Hello SIT202
Python: Se...
Python: cli...
Ln 22, Col 1
Spaces: 4
UTF-8
LF
Python
3.12.5 64-bit
Go Live



QUESTION:

- Why do we need transport layer protocols?
- Can the Internet survive without transport layer protocols?

ANSWER:

Why Do We Need Transport Layer Protocols?

Transport layer protocols are essential because they:

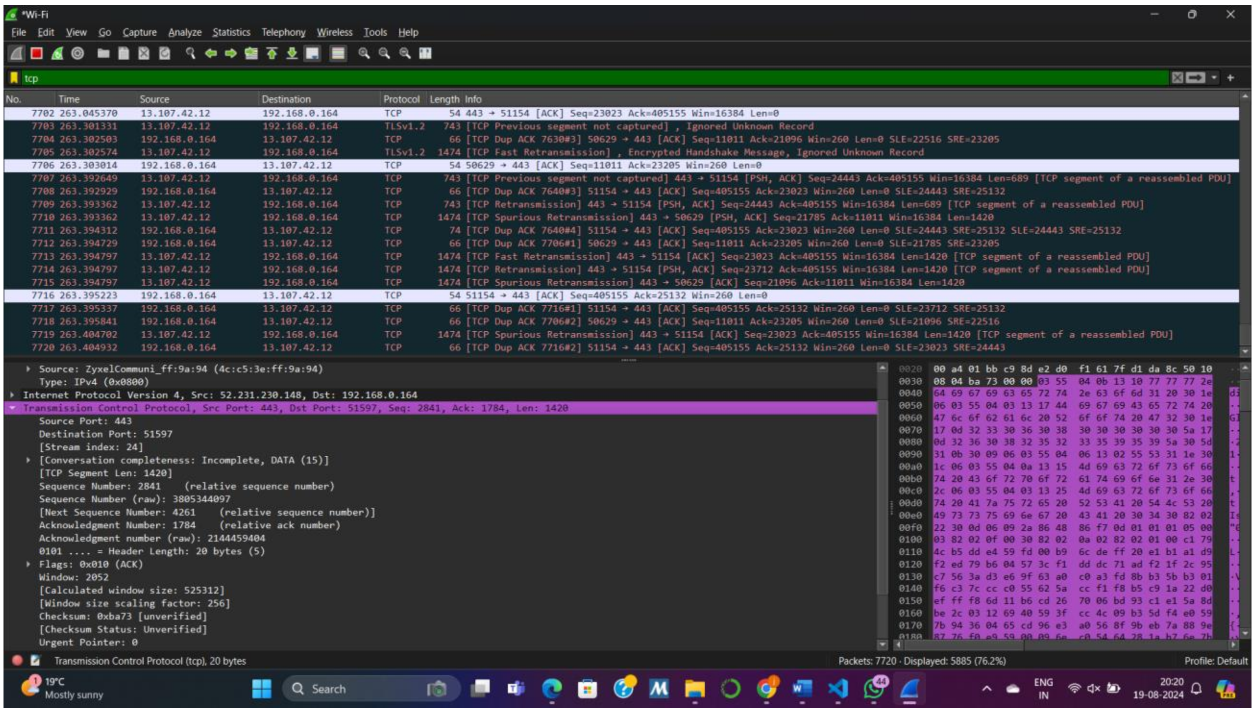
1. Ensure Reliable Communication: They guarantee that data arrives correctly and in the right order.
2. Handle Errors: They detect and correct errors during transmission.
3. Control Data Flow: They prevent data overflow by managing the rate of transmission.
4. Manage Connections: They establish, maintain, and terminate communication sessions.

5. Segment Data: They break data into manageable pieces and reassemble it at the destination.
6. Direct Data: They use port numbers to route data to the correct application.

Can the Internet Survive Without Transport Layer Protocols?

No, the Internet cannot function effectively without transport layer protocols. They provide the necessary mechanisms for reliable, error-free data transfer, flow control, and proper routing of data to applications. Without them, communication would be unreliable and inefficient.

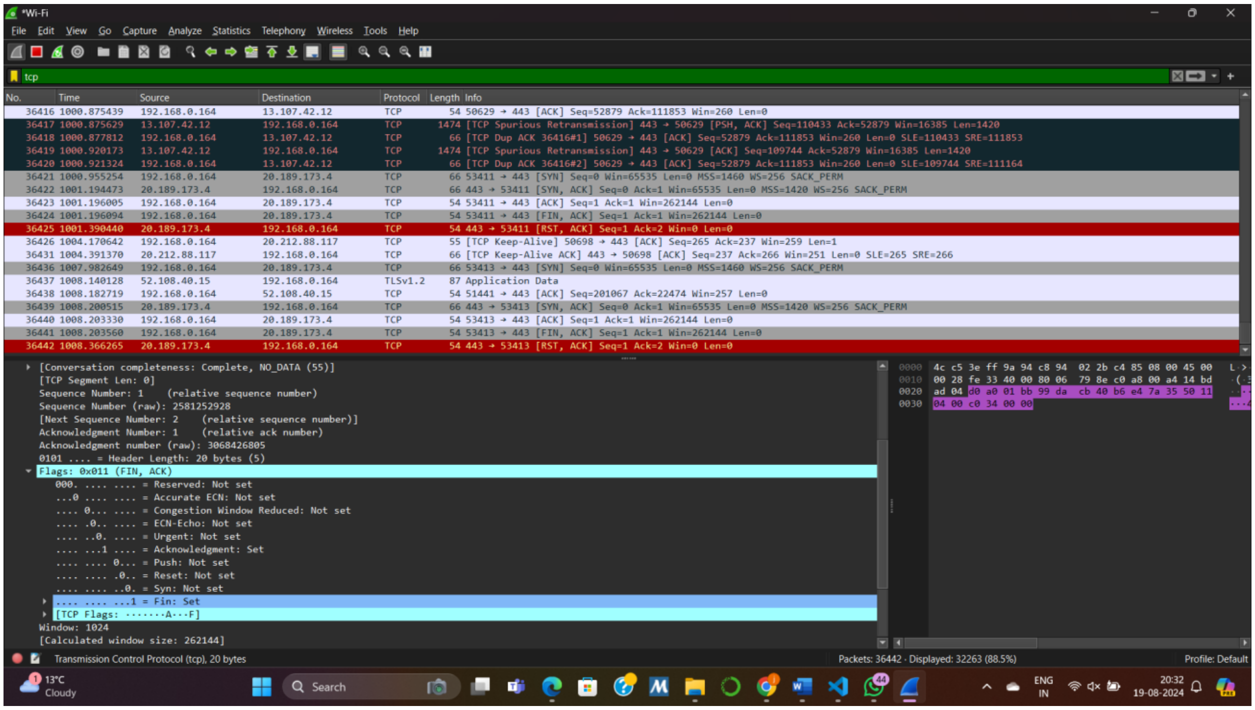
QUESTION:
Explore different TCP segments in a Wireshark packet capture and identify different header fields used in TCP segments. Note down your answer before you continue.



Source port: 443
Destination port: 51597
Sequence number: 2841
Acknowledgement number: 1784
Header lengths: 20 bytes
Flags: 0x010
Window: 2052
Checksum : oxba73
Urgent pointer : 0

QUESTION:

In a Wireshark Packet capture of a network application that uses TCP as the transport layer protocol, can you identify "three-way handshake" and relevant TCP segments with SYN, SYN/ACK, ACK flags are set to 1?



HANDWRITTEN NOTES:

Services provided by the transport layer:-

☆ **Transport layer responsibilities:-**

=> Establishes logical communication channel between application processes on different end systems/hosts.

☆ **Contrast with Network layer:-**

=> Ensures connectivity within the same or across different networks.

☆ **Transport layer Protocols:-**

=> Facilated by transport layer protocols.

☆ **Protocols used for Internet Applications:-**

=> Two main transport layer protocols
- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)

☆ Transport Protocol actions in End systems

Sender:-
- Breaks application messages into segments.
- Passes segments to network layer

Reciever
- Reassembles segments into messages
- Passes messages to application layer

MULTIPLEXING

- Combines data from multiple application processes on a single host
- Creates a unified data stream for transmission.

DEMULTIPLEXING:

- Separates incoming data into distinct streams.
- Delivers data to the correct application processes on the destination host.

Sockets :-

- Process sends/receives messages to/from its socket
- Socket analogous to door
  - ☆ sending process shoves message outdoor
  - ☆ sending process relies on transport infrastructure on other side of door to deliver messages to socket at receiving process
  - ☆ two sockets involved: one on each side.

HOW DEMULTIPLEXING WORKS IN HOST

Host receives IP datagrams

Host uses IPaddress & port number to direct segment to appropriate socket

CONNECTIONLESS DEMULTIPLEXING :-

- When creating socket, must specify host-local port
- When creating datagram to send into UDP socket, must specify
- when receiving host receives UDP segment
- IP/UDP ~~port~~ datagrams with same dest port # but different source IP addresses and/or source port numbers will be directed to same socket at receiving host

UDP – USER DATAGRAM PROTOCOL :-

- "no frills", "bare bones" Internet Transport Protocol.
- "best effort" service, UDP, segments maybe :-
  - ☆ lost
  - ☆ delivered out-of-order to app.
- Connectionless :-
  - ☆ no handshaking between UDP sender, receiver.
  - ☆ each UDP segment handled independently of others.

UDP - User Datagram Protocol :- uses

- DNS
- streaming multimedia apps
- SNMP
- HTTP/3

TCP - Transmission control protocol :-

1) Connection Oriented Protocol :-

- Establish Connection
- Point to Point

Initial RFC : RFC 793
Refinements : 1122, 1323, 2018, 2581

=> Handshake => agreement on parameters for communication.

Data Transfer
- Send Buffer
- Segmentation

TCP sequence Numbers, ACKs

Sequence Numbers : byte stream "number" of first byte in segment's data.

ACKs : ~~Seg~~ • Seq # of next byte expected from other side
• Cumulative ACK

Control Flags :-

URG: Urgent pointer – used to identify incoming data as "urgent".

ACK: Acknowledgement – to indicate the acknowledgment of the receipt of data.

PHS: Push: to indicate to push currently-buffered data.

RST: Reset: is used to indicate TCP hasn't been able to properly recover from misinformed segments.

SYN: Synchronise: is used when first establishing a TCP connection

FIN: Finish: used to indicate that the connection can be closed.

TCP Sender:

event: data received from application:-

* create segment with seq #
* seq # is byte-stream number of first data byte in segment
* start timer if not already running.

event: timeout
* retransmit segment that caused timeout
* restart timer

event: ACK received
* if ACK acknowledges previously unACKed segments

AQR Protocols Flowchart

AUTOMATIC REPEAT REQUEST PROTOCOLS

* Purpose :- Ensure reliable communication
* Components :- Error detection, acknowledgement receipt, and retransmission.

KEY ARQ ALGORITHMS

* Stop & wait
* Process : Send one packet, wait for acknowledgment, then send the next.
* Issue : Inefficient channel utilization due to idle time.

GO-BACK-N

* Process : Send multiple packets without waiting for individual acknowledgements, constrained by window size.
* Issue : Requires retransmission of multiple packets if an error occurs.

~~* Selective~~

SELECTIVE REPEAT

* Process :- Retransmit Packets with errors only.
* Advantage :- Minimizes unnecessary retransmission improving efficiency.