

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.

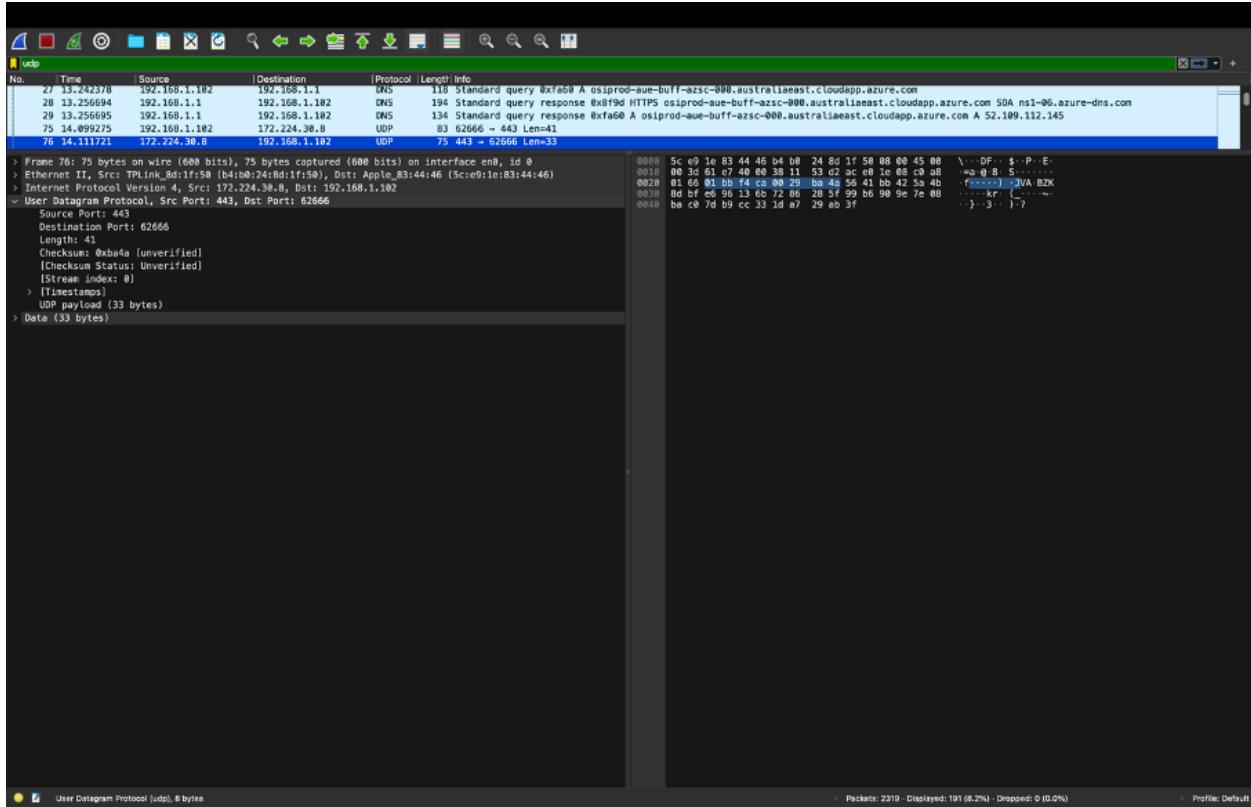
Pranika: Exactly, UDP is fast but doesn't guarantee that everything arrives safely or in order, like streaming videos.

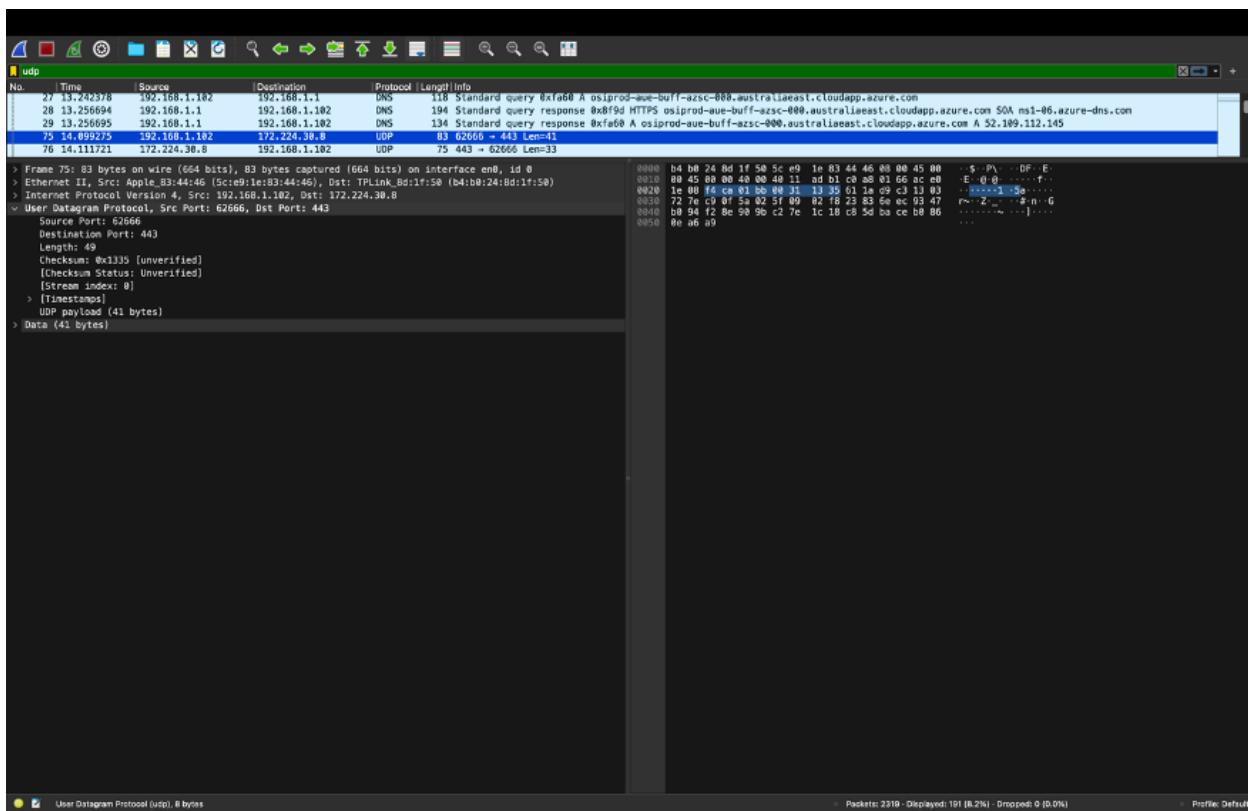
Gitanshi: But if we have more time and care about the order and safety of each item, we'd use TCP—carefully packing and labeling everything.

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:

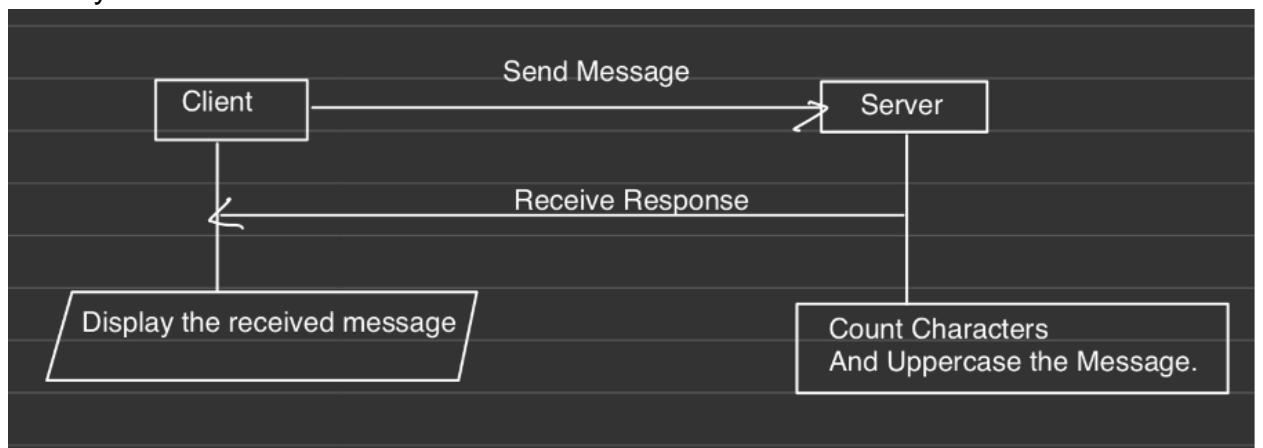
- Varies: Different applications/sessions use different port numbers.

No.	Time	Source	Destination	Protocol	Length/Info
1	0.000888	192.168.1.182	192.168.1.1	DNS	86 Standard query 0x3615 HTTPS contacts.fe2.apple-dns.net
2	0.008193	192.168.1.182	192.168.1.1	DNS	86 Standard query 0x8efc A contacts.fe2.apple-dns.net
3	0.012342	192.168.1.1	192.168.1.182	DNS	171 Standard query response 0x3616 HTTPS contacts.fe2.apple-dns.net SOA ns-1396.awsdns-46.org
4	0.021981	192.168.1.1	192.168.1.182	DNS	158 Standard query response 0x8efc A contacts.fe2.apple-dns.net A 17.248.219.67 A 17.248.219.66 A 17.248.219.64 A 17.248.219.65
58	0.742678	192.168.1.182	192.168.1.1	DNS	83 Standard query 0x4076 HTTPS p49-contacts.icloud.com
61	0.742952	192.168.1.182	192.168.1.1	DNS	83 Standard query 0x0d83 A p49-contacts.icloud.com
63	0.752913	192.168.1.1	192.168.1.182	DNS	208 Standard query response 0x4b76 HTTPS p49-contacts.icloud.com CNAME contacts.fe2.apple-dns.net SOA ns-1396.awsdns-46.org
68	0.754855	192.168.1.1	192.168.1.182	DNS	187 Standard query response 0x0d83 A p49-contacts.icloud.com CNAME contacts.fe2.apple-dns.net A 17.248.219.67 A 17.248.219.66 A 17.248.219.64 A 17.248.219.65
196	2.161137	17.253.66.253	192.168.1.182	UDP	108 443 - 58481 Len=38
197	2.161289	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
198	2.162859	17.253.66.253	192.168.1.182	UDP	74 443 - 58481 Len=32
199	2.442462	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
200	2.442516	17.253.66.253	192.168.1.182	UDP	108 443 - 58481 Len=35
241	1.750377	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
282	2.442466	17.253.66.253	192.168.1.182	UDP	108 443 - 58481 Len=66
283	2.442478	17.253.66.253	192.168.1.182	UDP	77 443 - 58481 Len=35
284	2.442703	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
285	2.442798	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
218	2.815941	17.253.66.253	192.168.1.182	UDP	108 443 - 58481 Len=66
211	2.815943	17.253.66.253	192.168.1.182	UDP	77 443 - 58481 Len=35
212	2.816098	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
213	2.816151	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
251	3.489194	192.168.1.182	192.168.1.1	DNS	91 Standard query 0x1310 HTTPS guzzoni-apple-com.v.aapling.com
252	3.489208	192.168.1.182	192.168.1.1	DNS	91 Standard query 0x8e28 A guzzoni-apple-com.v.aapling.com
258	3.411819	192.168.1.1	192.168.1.182	DNS	165 Standard query response 0x1310 HTTPS guzzoni-apple-com.v.aapling.com SOA ns4.g.slib.aapling.com
259	3.411811	192.168.1.1	192.168.1.182	DNS	187 Standard query response 0x8e28 A guzzoni-apple-com.v.aapling.com A 3.185.196.9
288	3.568783	17.253.66.253	192.168.1.182	UDP	108 443 - 58481 Len=66
289	3.568784	17.253.66.253	192.168.1.182	UDP	77 443 - 58481 Len=35
298	3.568862	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
201	3.568925	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
482	5.104681	17.253.66.253	192.168.1.182	UDP	108 443 - 58481 Len=66
483	5.104682	17.253.66.253	192.168.1.182	UDP	77 443 - 58481 Len=35
484	5.104682	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
485	5.104897	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
478	6.453919	192.168.1.182	172.224.38.196	UDP	83 59545 - 443 Len=41
479	6.463238	172.224.38.196	192.168.1.182	UDP	75 443 - 39545 Len=33
544	7.169188	192.168.1.182	192.168.1.1	DNS	89 Standard query 0x9fd5 HTTPS ocp2-lb.apple.com.akadns.net
545	7.152069	192.168.1.182	192.168.1.1	DNS	89 Standard query 0xe57e A ocp2-lb.apple.com.akadns.net
546	7.257681	192.168.1.182	192.168.1.1	DNS	86 Standard query 0x2f1a HTTPS config.teams.microsoft.com
547	7.157688	192.168.1.182	192.168.1.1	DNS	86 Standard query 0x3d29 A config.teams.microsoft.com
548	7.166865	192.168.1.1	192.168.1.182	DNS	386 Standard query response 0x2f61 HTTPS config.teams.microsoft.com CNAME config.teams.trafficmanger.net CNAME s-0005-teams.config.skype.com
549	7.169164	192.168.1.182	192.168.1.1	DNS	79 Standard query 0x8c54 HTTPS s-0005.s-msedge.net
550	7.169988	192.168.1.1	192.168.1.182	DNS	250 Standard query response 0x4200 A config.teams.trafficmanger.net CNAME s-0005-teams.config.skype.com
551	7.174775	192.168.1.1	192.168.1.182	DNS	196 Standard query response 0x7d76 HTTPS ocp2-lb.apple.com.akadns.net SOA ns4.g.aapling.com
552	7.175177	192.168.1.182	192.168.1.1	DNS	79 Standard query 0xcf55 HTTPS ocp2-g.aapling.com
553	7.177696	192.168.1.1	192.168.1.182	DNS	139 Standard query response 0x8c54 HTTPS s-0005.s-msedge.net SOA ns1.s-msedge.net
555	7.185289	192.168.1.1	192.168.1.182	DNS	153 Standard query response 0x8cf6 HTTPS ocp2-g.aapling.com SOA ns4.g.aapling.com
559	7.189823	192.168.1.1	192.168.1.182	DNS	154 Standard query response 0x857e A ocp2-lb.apple.com.akadns.net CNAME ocp2-g.aapling.com A 17.253.121.282 A 17.253.121.281
600	8.027232	17.253.66.253	192.168.1.182	UDP	108 443 - 58481 Len=66
601	8.027232	17.253.66.253	192.168.1.182	UDP	77 443 - 58481 Len=35
602	8.027232	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
603	8.027357	192.168.1.182	17.253.66.253	ICMP	78 Destination unreachable (Port unreachable)
604	8.027357	192.168.1.182	192.168.1.182	DNS	91 Standard query 0x9748 HTTPS teams-events-data.microsoft.com

- 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



The screenshot shows a code editor interface with two tabs: "Server_Side.py" and "client_side.py". The "Server_Side.py" tab is active, displaying the following Python code:

```
 1 import socket
 2
 3 # Create a UDP server socket
 4 server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
 5 server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
 6 server_socket.bind(('localhost', 12345))
 7
 8 print("Server is running...")
 9
10 while True:
11     # Receive message from client
12     message, client_address = server_socket.recvfrom(1024)
13
14     # Process the message
15     char_count = len(message)
16     uppercase_message = message.decode('utf-8').upper()
17     response = f'{char_count} characters: {uppercase_message}'
18
19     # Send response back to client
20     server_socket.sendto(response.encode("utf-8"), client_address)
21     print(f"Processed message: {message.decode('utf-8')}")
22
```

Below the code editor is a terminal window showing the execution of the script and its output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
/usr/local/bin/python3 "/Users/architchandna/Desktop/SIT 202 Computer networks and Communication/5.1P/Server_Side.py"
(base) architchandna@ARCHIT-Pro:~$ /usr/local/bin/python3 "/Users/architchandna/Desktop/SIT 202 Computer networks and Communication/5.1P/Server_Side.py"
Server is running...
Processed message: Hello SIT202
```

The terminal also shows the file path and the Python version used: "/usr/local/bin/python3" and "Python 3.12.5 64-bit".

The screenshot shows a code editor window with the following details:

- File Explorer:** Shows two files: `Server_Side.py` and `client_side.py`.
- Code Editor:** The `client_side.py` file contains Python code for a UDP client. It imports `socket`, creates a socket, sends a message to the server, receives a response, and prints it.
- Terminal:** The terminal shows the command `/usr/local/bin/python3 "/Users/architchandna/Desktop/SIT 282 Computer networks and Communication/5.1P/client_side.py"` being run, followed by the server's response "Hello SIT282".
- Status Bar:** Shows the current line (Ln 16, Col 1), spaces used (Spaces: 4), encoding (UTF-8), language (Python 3.12.5 64-bit), and Go Live status.

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.

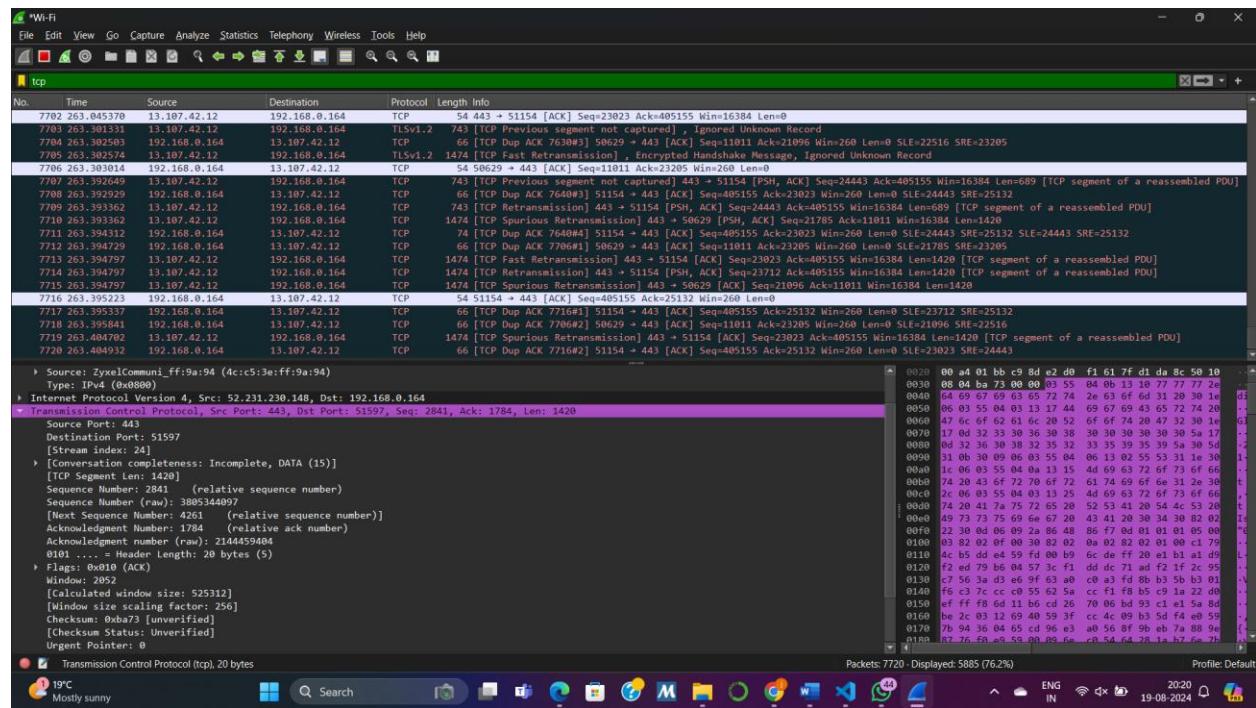
6. Direct Data: They use port numbers to route data to the correct application.

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

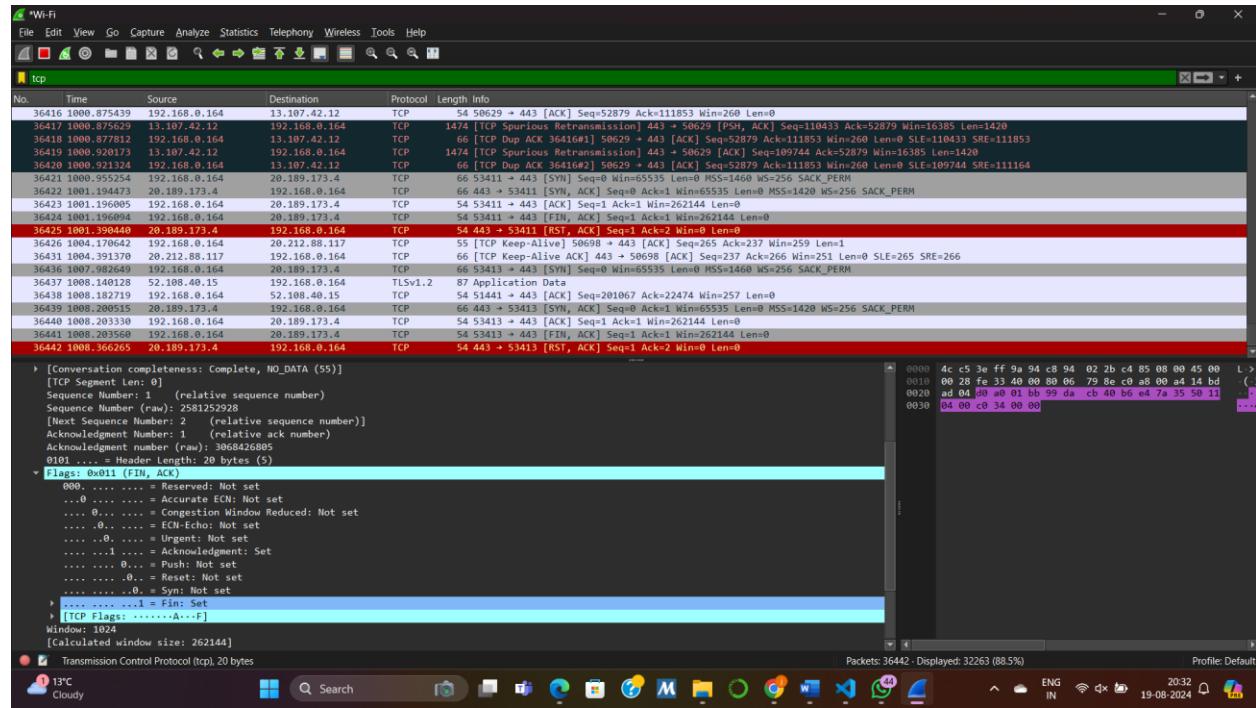
Window: 2052

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 :-

=> Facilitated 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

Receiver

- 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

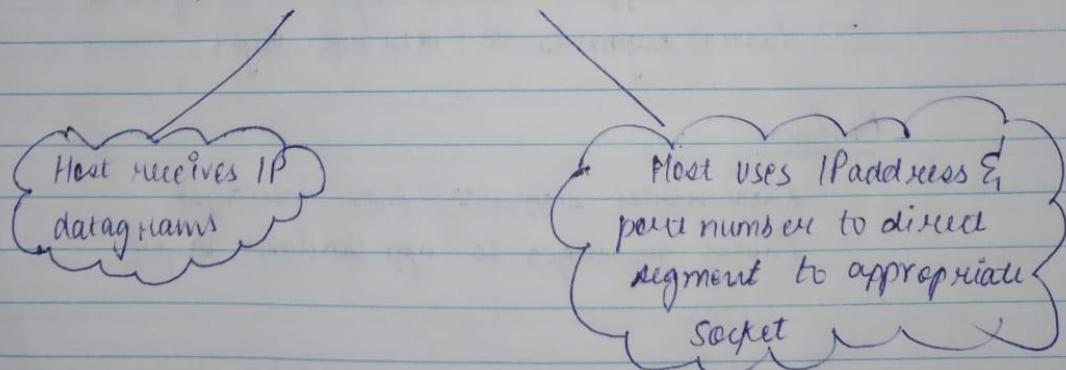
DEMULTIPLEXING:

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

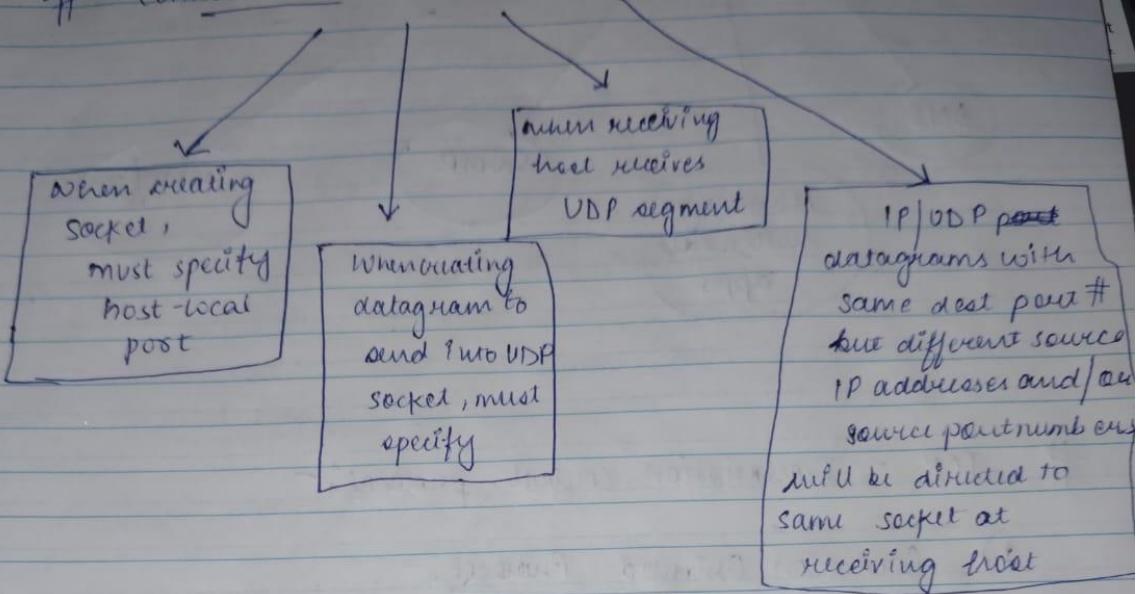
Sockets :-

- Process sends/receives messages to/from its socket
- Socket analogous to door
 - * sending process sends message outside
 - * 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



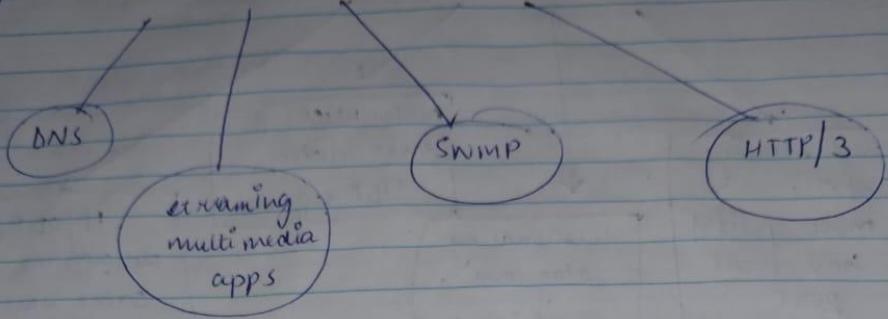
CONNECTIONLESS DEMULTIPLEXING :-



UDP - USER DATAGRAM PROTOCOL :-

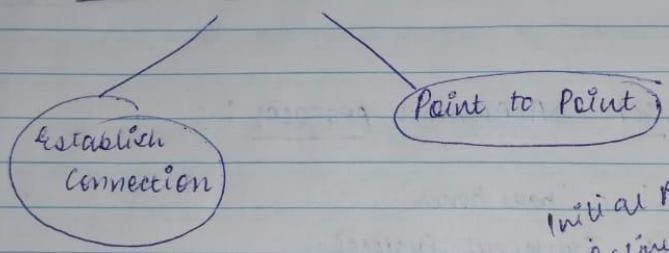
- "no frills", "bare bones" Internet Transport Protocol.
- "best effort" service, UDP segments maybe:
 - * lost
 - * delivered out-of-order to app.
- Connectionless:-
 - * no handshaking between UDP sender, receiver.
 - * each UDP segment handled independently of others.

UDP - User Datagram Protocol :- uses



TCP - Transmission control protocol :-

1) Connection Oriented Protocol :-



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 :
• 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 acknowledgement of the receipt of data .

PUSH : 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.

GRO-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.