

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

The image shows a Wireshark packet capture window. The top pane displays a list of packets. The selected packet is a UDP packet (No. 76) with the following details:

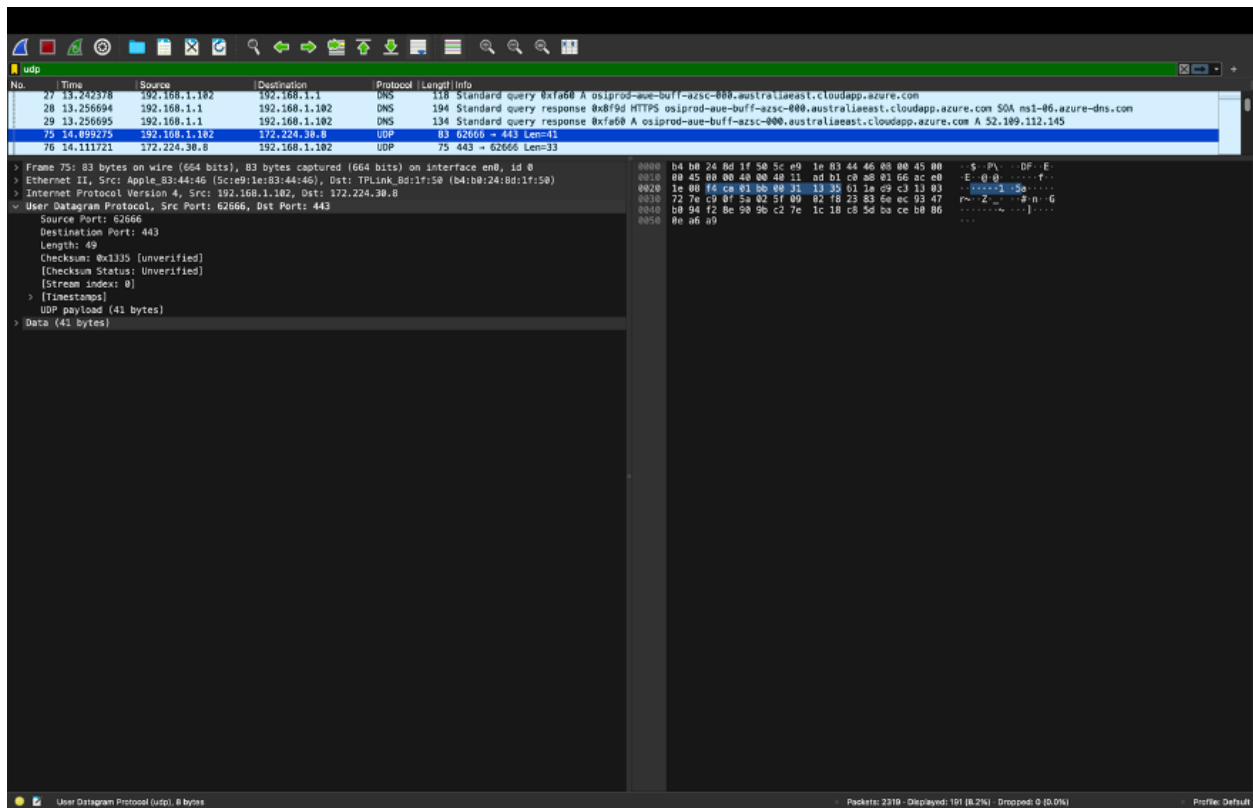
No.	Time	Source	Destination	Protocol	Length	Info
76	14.111721	172.224.30.8	192.168.1.182	UDP	75	443 → 62666 [Len=33]

The middle pane shows the packet details for the selected UDP packet:

- Frame 76: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface en0, id 0
- Ethernet II, Src: TP-Link_Bd1f:50:8d (b8:10:24:8d:1f:50), Dst: Apple_83:44:46 (8c:91:1e:83:44:46)
- Internet Protocol Version 4, Src: 172.224.30.8, Dst: 192.168.1.182
- User Datagram Protocol, Src Port: 443, Dst Port: 62666
 - Source Port: 443
 - Destination Port: 62666
 - Length: 41
 - Checksum: 0eba4a [unverified]
 - [Checksum Status: Unverified]
 - [Stream index: 0]
 - [Timestamps]
 - UDP payload (33 bytes)
 - Data (33 bytes)

The bottom pane shows the raw packet data in hexadecimal and ASCII:

```
0000 5c e9 1e 83 44 46 b4 b0 24 8d 1f 50 8d 00 00 00  \...DP: 5...P:E:
0010 00 3e 61 07 40 00 38 11 53 02 ac e0 1e 00 c0 a8  ->0.0.0.5:5...
0020 01 66 01 bb f4 c0 00 29 ba 4a 56 41 bb 42 5a 4b  -Yxxxxx) 3VA: BZX
0030 0d bf e6 96 13 65 72 86 20 5f 99 b6 90 9e 7e 00  -...kr) (-...-
0040 ba c0 7d b9 cc 33 1d 07 29 ab 3f  -} 3 } 7
```



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:

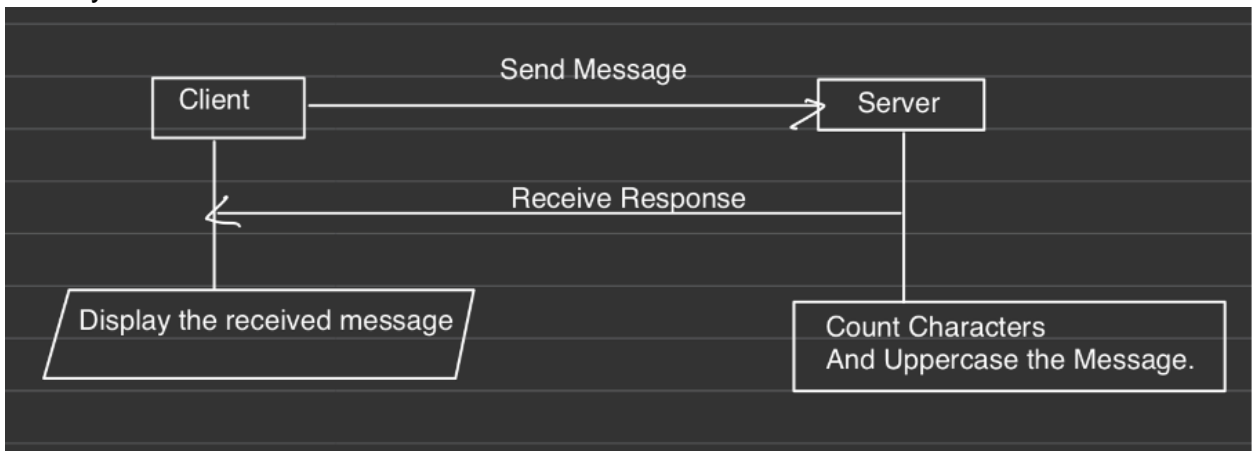
- Varies: Different applications/sessions use different port numbers.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.102	192.168.1.1	DNS	86	Standard query 0x3616 HTTPS contacts.fe2.apple-dns.net
2	0.000193	192.168.1.102	192.168.1.1	DNS	86	Standard query 0x3616 HTTPS contacts.fe2.apple-dns.net
3	0.012342	192.168.1.1	192.168.1.102	DNS	171	Standard query response 0x3616 HTTPS contacts.fe2.apple-dns.net SOA ns-1396.awdns-46.org
4	0.021981	192.168.1.1	192.168.1.102	DNS	158	Standard query response 0x3616 HTTPS 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
60	0.742678	192.168.1.102	192.168.1.1	DNS	83	Standard query 0x4b76 HTTPS p49-contacts.icloud.com
61	0.742952	192.168.1.102	192.168.1.1	DNS	83	Standard query 0x4b76 HTTPS p49-contacts.icloud.com
63	0.752913	192.168.1.1	192.168.1.102	DNS	208	Standard query response 0x4b76 HTTPS p49-contacts.icloud.com CNAME contacts.fe2.apple-dns.net SOA ns-1396.awdns-46.org
68	0.754856	192.168.1.1	192.168.1.102	DNS	187	Standard query response 0x4b76 HTTPS p49-contacts.icloud.com CNAME contacts.fe2.apple-dns.net A 17.248.219.67 A 17.248.219.65 A 17.248.219.64 A 17.248.219.66
196	2.161137	17.253.66.253	192.168.1.102	UDP	188	443 → 58481 Len=56
197	2.161180	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
198	2.162854	17.253.66.253	192.168.1.102	UDP	74	443 → 58481 Len=32
199	2.162932	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
200	2.256186	17.253.66.253	192.168.1.102	UDP	188	443 → 58481 Len=66
201	2.256377	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
202	2.442466	17.253.66.253	192.168.1.102	UDP	188	443 → 58481 Len=66
203	2.442478	17.253.66.253	192.168.1.102	UDP	77	443 → 58481 Len=35
204	2.442783	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
205	2.442798	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
218	2.815941	17.253.66.253	192.168.1.102	UDP	188	443 → 58481 Len=66
219	2.815943	17.253.66.253	192.168.1.102	UDP	77	443 → 58481 Len=35
212	2.816080	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
213	2.816151	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
251	3.482729	192.168.1.102	192.168.1.1	DNS	91	Standard query 0x1310 HTTPS guzzoni-apple-com.v.aaplimg.com
252	3.482894	192.168.1.102	192.168.1.1	DNS	91	Standard query 0x1310 HTTPS guzzoni-apple-com.v.aaplimg.com
258	3.411818	192.168.1.1	192.168.1.102	DNS	185	Standard query response 0x1310 HTTPS guzzoni-apple-com.v.aaplimg.com SOA a.gslb.aaplimg.com
259	3.411811	192.168.1.1	192.168.1.102	DNS	187	Standard query response 0x1310 HTTPS guzzoni-apple-com.v.aaplimg.com A 3.185.196.9
288	3.568783	17.253.66.253	192.168.1.102	UDP	188	443 → 58481 Len=66
289	3.568784	17.253.66.253	192.168.1.102	UDP	77	443 → 58481 Len=35
298	3.568862	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
299	3.568925	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
482	5.104681	17.253.66.253	192.168.1.102	UDP	188	443 → 58481 Len=66
483	5.104682	17.253.66.253	192.168.1.102	UDP	77	443 → 58481 Len=35
484	5.104862	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
485	5.104897	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
478	6.455915	192.168.1.102	172.224.30.196	UDP	83	59545 → 443 Len=41
479	6.463259	172.224.30.196	192.168.1.102	UDP	75	443 → 59545 Len=33
544	7.158188	192.168.1.102	192.168.1.1	DNS	89	Standard query 0xd7dc HTTPS ocs2-lb.apple.com.akadns.net
545	7.158269	192.168.1.102	192.168.1.1	DNS	89	Standard query 0xe57e A ocs2-lb.apple.com.akadns.net
546	7.157681	192.168.1.102	192.168.1.1	DNS	86	Standard query 0x2fa1 HTTPS config.teams.microsoft.com
547	7.157698	192.168.1.102	192.168.1.1	DNS	86	Standard query 0x1420 A config.teams.microsoft.com
548	7.168085	192.168.1.1	192.168.1.102	DNS	386	Standard query response 0x2fa1 HTTPS config.teams.microsoft.com CNAME config.teams.trafficmanager.net CNAME s-8005-teams.config.skype.com
549	7.169164	192.168.1.102	192.168.1.1	DNS	79	Standard query 0x8c34 HTTPS s-8005-s-esedge.net
550	7.169980	192.168.1.1	192.168.1.102	DNS	250	Standard query response 0x1420 A config.teams.microsoft.com CNAME config.teams.trafficmanager.net CNAME s-8005-teams.config.skype.com
551	7.174775	192.168.1.1	192.168.1.102	DNS	196	Standard query response 0xd7dc HTTPS ocs2-lb.apple.com.akadns.net CNAME ocs2-g.aaplimg.com SOA a.gslb.aaplimg.com
552	7.175177	192.168.1.102	192.168.1.1	DNS	79	Standard query 0xcfd5 HTTPS ocs2-g.aaplimg.com
553	7.177696	192.168.1.1	192.168.1.102	DNS	139	Standard query response 0xcfd5 HTTPS s-8005-s-esedge.net SOA ns1.s-esedge.net
555	7.185289	192.168.1.1	192.168.1.102	DNS	153	Standard query response 0xcfd5 HTTPS ocs2-g.aaplimg.com SOA a.gslb.aaplimg.com
559	7.189828	192.168.1.1	192.168.1.102	DNS	154	Standard query response 0xe57e A ocs2-lb.apple.com.akadns.net CNAME ocs2-g.aaplimg.com A 17.253.121.202 A 17.253.121.201
668	8.027232	17.253.66.253	192.168.1.102	UDP	188	443 → 58481 Len=66
661	8.027232	17.253.66.253	192.168.1.102	UDP	77	443 → 58481 Len=35
682	8.027315	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
683	8.027357	192.168.1.102	17.253.66.253	ICMP	78	Destination unreachable (Port unreachable)
684	8.029321	192.168.1.102	192.168.1.1	DNS	91	Standard query 0x5938 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



Server_Side.py X client_side.py

Users > architchandna > Desktop > SIT 202 Computer networks and Communicatio > 5.1P > Server_Side.py > ...

```
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
```

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
```

Ln 22, Col 1 Spaces: 4 UTF-8 LF Python 3.12.5 64-bit Go Live

```
Server_Side.py client_side.py x
Users > architchandna > Desktop > SIT 202 Computer networks and Communicatio > 5.1P > client_side.py > ...
1 import socket
2
3 # Create a UDP client socket
4 client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
5
6 # Message to send
7 message = "Hello SIT202"
8 client_socket.sendto(message.encode('utf-8'), ('localhost', 12345))
9
10 # Receive and print response from server
11 response, _ = client_socket.recvfrom(1024)
12 print(f"Server response: {response.decode('utf-8')}")
13
14 client_socket.close()
15
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(base) architchandna@ARCHITs-MacBook-Pro fly_catch_s_e % /usr/local/bin/python3 "/Users/architchandna/Desktop/SIT 202 Computer networks and Communicatio/5.1P/client_side.py"
Server response: 12 characters: HELLO SIT202
(base) architchandna@ARCHITs-MacBook-Pro fly_catch_s_e %
```

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.

Wireshark packet capture showing a TCP segment. The packet list shows a packet from 192.168.0.164 to 192.168.0.164, port 443 to 51597. The packet details show the TCP header fields: Source Port: 443, Destination Port: 51597, Sequence Number: 2841, Acknowledgment Number: 1784, Window: 2052, Flags: 0x010, Checksum: 0xba73, Urgent Pointer: 0. The packet bytes show the raw data of the TCP segment.

Source port: 443
Destination port: 51597
Sequence number: 2841
Acknowledgement number: 1784
Header lengths: 20 bytes
Flags: 0x010
Window: 2052
Checksum : 0xba73
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?

Wireshark packet capture showing a TCP three-way handshake. The packet list on the left shows packets 36416 through 36442. Packet 36416 is a SYN segment from 192.168.0.164 to 13.107.42.12. Packet 36417 is a SYN/ACK segment from 13.107.42.12 to 192.168.0.164. Packet 36418 is a duplicate ACK segment from 192.168.0.164 to 13.107.42.12. Packet 36419 is a duplicate SYN segment from 192.168.0.164 to 13.107.42.12. Packet 36420 is a duplicate ACK segment from 13.107.42.12 to 192.168.0.164. Packet 36421 is a duplicate SYN segment from 192.168.0.164 to 13.107.42.12. Packet 36422 is a duplicate SYN/ACK segment from 13.107.42.12 to 192.168.0.164. Packet 36423 is a duplicate ACK segment from 192.168.0.164 to 13.107.42.12. Packet 36424 is a duplicate SYN segment from 192.168.0.164 to 13.107.42.12. Packet 36425 is a duplicate ACK segment from 13.107.42.12 to 192.168.0.164. Packet 36426 is a duplicate SYN segment from 192.168.0.164 to 13.107.42.12. Packet 36427 is a duplicate SYN/ACK segment from 13.107.42.12 to 192.168.0.164. Packet 36428 is a duplicate ACK segment from 192.168.0.164 to 13.107.42.12. Packet 36429 is a duplicate SYN segment from 192.168.0.164 to 13.107.42.12. Packet 36430 is a duplicate SYN/ACK segment from 13.107.42.12 to 192.168.0.164. Packet 36431 is a duplicate ACK segment from 192.168.0.164 to 13.107.42.12. Packet 36432 is a duplicate SYN segment from 192.168.0.164 to 13.107.42.12. Packet 36433 is a duplicate SYN/ACK segment from 13.107.42.12 to 192.168.0.164. Packet 36434 is a duplicate ACK segment from 192.168.0.164 to 13.107.42.12. Packet 36435 is a duplicate SYN segment from 192.168.0.164 to 13.107.42.12. Packet 36436 is a duplicate SYN/ACK segment from 13.107.42.12 to 192.168.0.164. Packet 36437 is a duplicate ACK segment from 192.168.0.164 to 13.107.42.12. Packet 36438 is a duplicate SYN segment from 192.168.0.164 to 13.107.42.12. Packet 36439 is a duplicate SYN/ACK segment from 13.107.42.12 to 192.168.0.164. Packet 36440 is a duplicate ACK segment from 192.168.0.164 to 13.107.42.12. Packet 36441 is a duplicate SYN segment from 192.168.0.164 to 13.107.42.12. Packet 36442 is a duplicate SYN/ACK segment from 13.107.42.12 to 192.168.0.164. The packet details pane on the right shows the selected packet 36416, which is a SYN segment from 192.168.0.164 to 13.107.42.12. The packet length is 54 bytes. The details pane shows the TCP segment structure, including the source and destination ports, sequence number, acknowledgment number, and flags. The flags field shows the SYN flag set to 1. The packet bytes pane on the right shows the raw data of the packet, including the Ethernet II header, Internet Protocol Version 4 header, and Transmission Control Protocol header.

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 :-

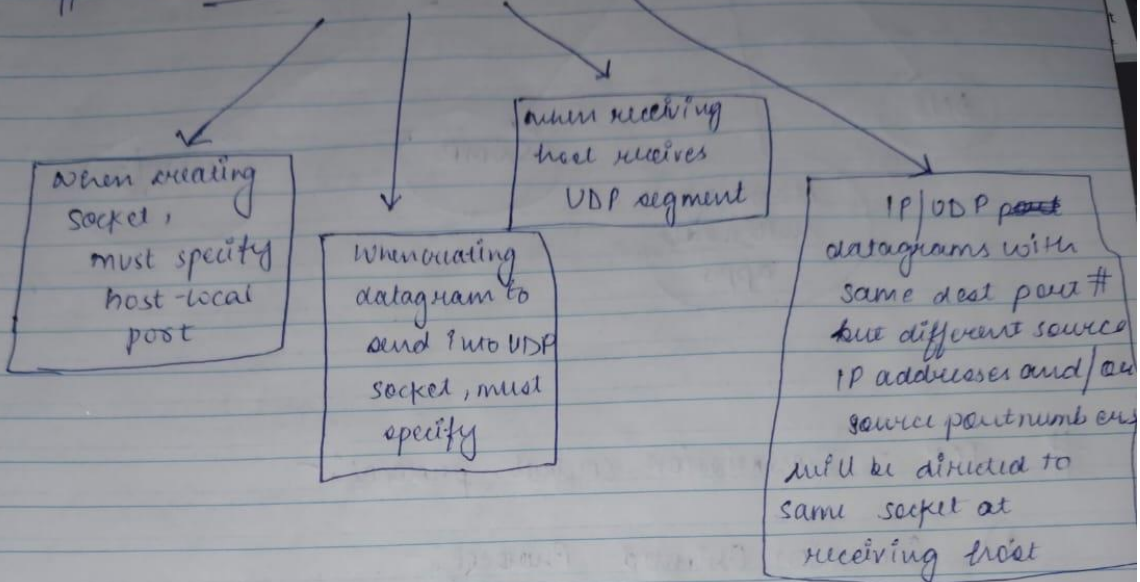
- Process sends/receives messages to/from its socket
- Socket analogous to door
 - * sending process sends message outdoor
 - * sending process relies on transport infrastructure on other side of door to deliver messages to socket at receiving process
 - * two sockets involved: one on each side

HOW DEMULTIPLEXING WORKS IN HOST

Host receives IP datagrams

Host uses IP address & port number to direct segment to appropriate socket

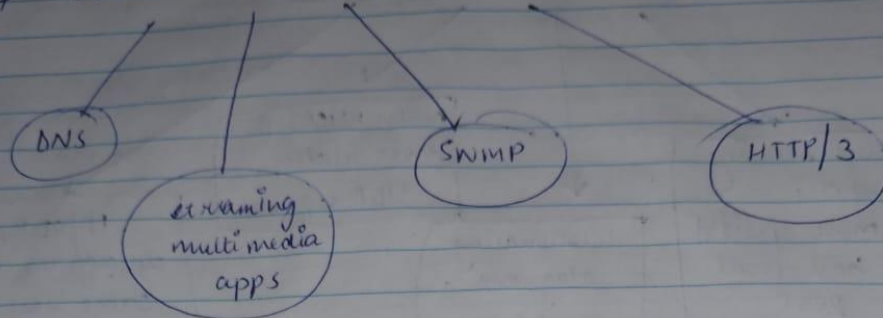
CONNECTIONLESS DEMULTIPLEXING :-



UDP - USER DATAGRAM PROTOCOL :-

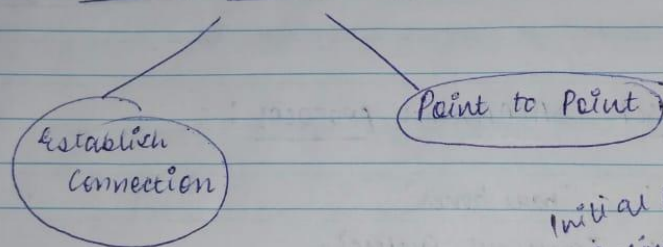
- "no frills", "bare bones" Internet Transport Protocol.
- "best effort" service, UDP segments may be:
 - * lost
 - * delivered out of order to app.
- Connectionless:-
 - * no handshaking between UDP sender, receiver.
 - * each UDP segment handled independently of others.

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 : ~~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.

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

ARQ 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 acknowledgement, 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.
- ★ ~~Solution~~

SELECTIVE REPEAT

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