Table of Contents

[Experiment 1 1](#_bookmark0)

[Experiment 2 4](#_bookmark1)

[Experiment 3 9](#_bookmark2)

[Experiment 4 10](#_bookmark3)

[Experiment 5 12](#_bookmark4)

[Experiment 6 14](#_bookmark5)

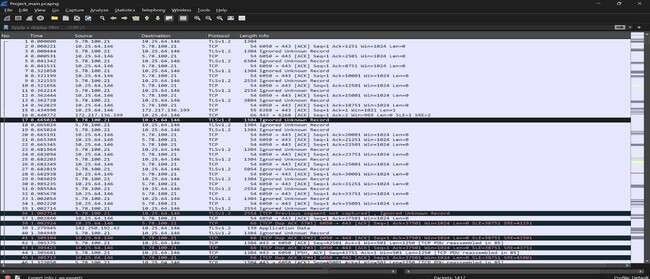
[Experiment 7 16](#_bookmark6)

[Experiment 8 18](#_bookmark7)

[Experiment 9 21](#_bookmark8)

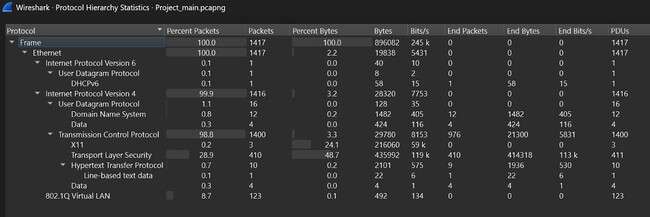
[Experiment 10 23](#_bookmark9)

# Experiment 1



## Aim – List the different protocols that appear in the protocol column in the unfiltered packet-listing window in Wireshark GUI.

Step 1 – Double click to open Wireshark, the following window will open.



Steps to view the Protocol Hierarchy Tab in Wireshark.

#### Access Protocol Hierarchy:

* 1. Navigate to the "Statistics" menu in the top toolbar.
  2. Select "Protocol Hierarchy" from the dropdown list.

1. **Analyze the Tab:** The "Protocol Hierarchy Statistics" window will appear, displaying a tree- like structure of the protocols detected in your capture, along with their packet counts, percentages, and byte counts.

#### Protocols Discovered During Network Analysis Using Wireshark

During network analysis with Wireshark, a wide array of protocols can be observed in the unfiltered packet-listing window and further detailed in the Protocol Hierarchy tab. The specific protocols you encounter will depend on the network traffic being captured, but common examples include:

* **TCP (Transmission Control Protocol):** A core protocol of the Internet Protocol Suite, providing reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating via an IP network.
* **UDP (User Datagram Protocol):** A connectionless protocol that, unlike TCP, offers no guarantees of delivery, ordering, or duplicate protection. It is often used for applications where speed is more critical than reliability, such as streaming video or online gaming.
* **HTTP (Hypertext Transfer Protocol):** The foundation of data communication for the World Wide Web, where hypertext documents are exchanged.
* **HTTPS (Hypertext Transfer Protocol Secure):** An extension of HTTP that provides secure communication over a computer network, widely used on the internet. HTTPS communication is encrypted and authenticated.
* **DNS (Domain Name System):** Translates human-readable domain names (like google.com) into machine-readable IP addresses (like 172.217.160.142), allowing devices to locate each other on the internet.
* **DHCP (Dynamic Host Configuration Protocol):** Enables a server to automatically assign an IP address and other communication parameters to a client connected to a network.
* **TLS (Transport Layer Security) / SSL (Secure Sockets Layer):** Cryptographic protocols designed to provide communications security over a computer network. They are widely used for securing web browsing, email, instant messaging, and other data transfers. The current TLS version is 1.2, replacing SSLv3, which is now obsolete.

By examining the Protocol column and the Protocol Hierarchy, users can gain valuable insights into the types of traffic flowing across their network, identify potential issues, and analyze application behavior.

Conclusion – This experiment demonstrates the functionality of wireshark UI, how it lists down the different protocols that are captured using the capture packet option.

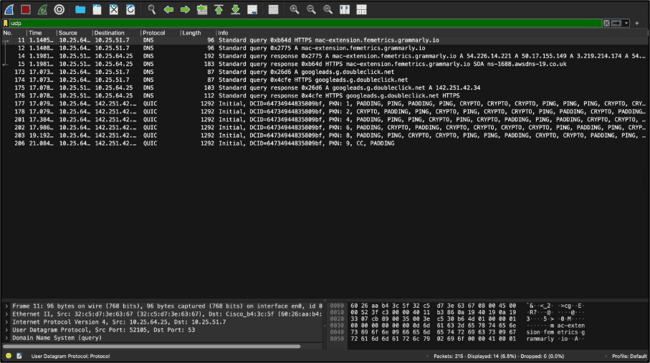
# Experiment 2

Aim – To Apply different types of filters and explain your findings.

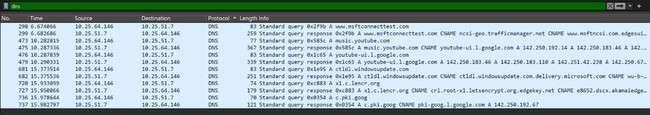
#### Protocol based filtering

* + Protocol-based filtering in Wireshark focuses on packets of a specific network protocol.
  + Achieved by entering protocol names (e.g., http, dns, tcp) in the display filter bar.

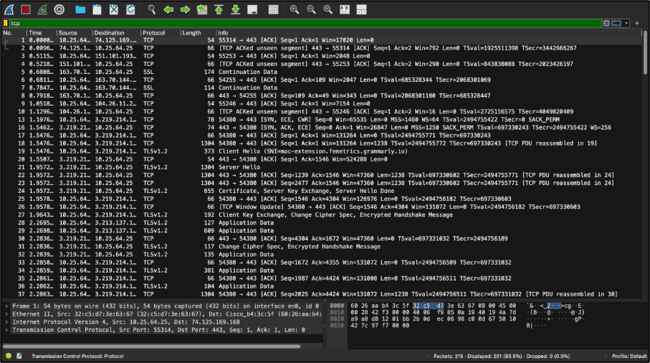
#### Filtering for UDP packets



#### Filtering for DNS packets



#### Filtering for TCP packets



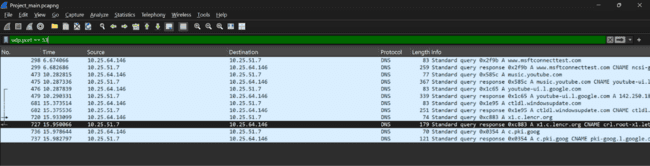
#### Port based filtering

* + Port-based filtering isolates packets based on source or destination port numbers.
  + This is effective for focusing on traffic from specific services or applications, which often use well-known port numbers (e.g., port 80 for HTTP, port 443 for HTTPS).
  + Filters can be applied using syntax like tcp.port == [port\_number], udp.port == [port\_number].

#### Filtering for TCP port 443 (for HTTPS)



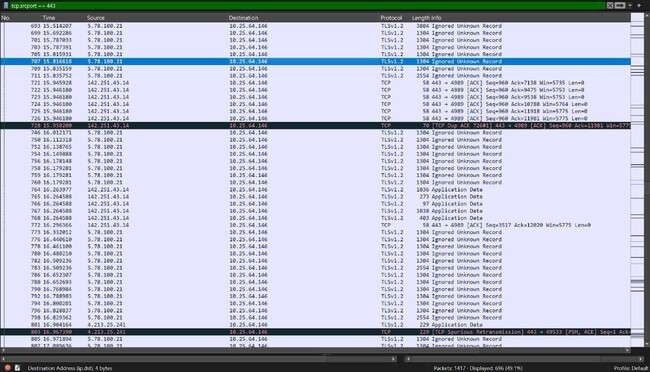
#### Filtering for UDP port 53 (for DNS)



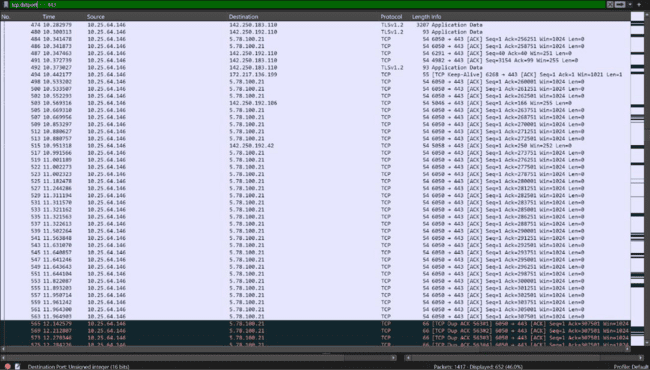
#### Source and Destination based filtering

* + Source and destination-based filtering allows users to isolate network traffic originating from or destined for specific IP addresses or MAC addresses.
  + This is particularly useful for analyzing communication patterns between particular hosts or for troubleshooting connectivity issues related to a specific device. Filters can be applied using syntax such as ip.src == [IP\_address], ip.dst == [IP\_address], tcp.srcport == [Port].

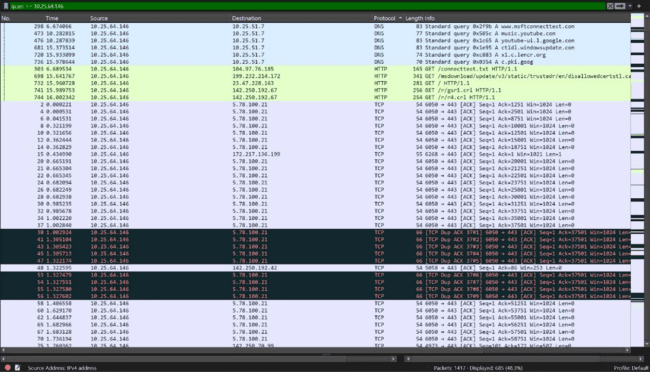
#### Filtering for TCP source port

****

#### Filtering for TCP destination port



#### Filtering for IP source address



#### Operators for filtering

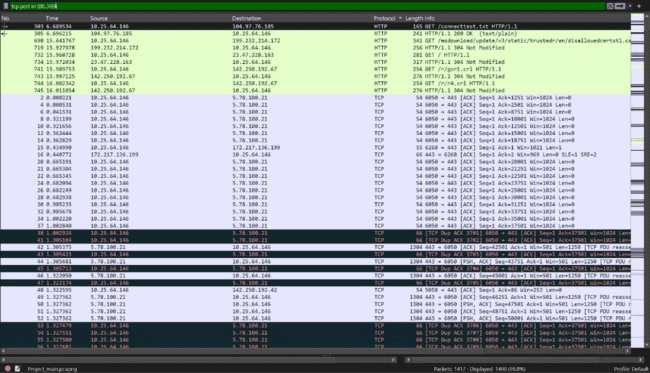
&& operator for checking both conditions, || operator for checking either conditions, ! operator for excluding. Range based filtering like tcp.port in {8000..8005}Use && to check if both conditions are met.

* + Use || to check if either condition is met.
  + Use ! for exclusion.
  + Apply range-based filtering, e.g., tcp.port in {8000..8005}.

## Filtering for TCP ACK packet using HTTPS

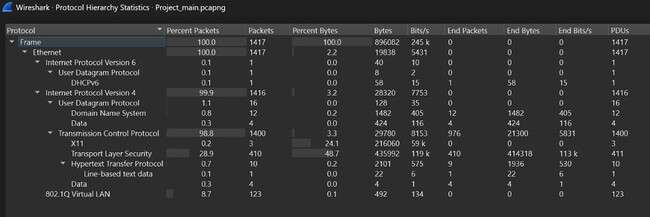


## Filtering for TCP packets in the range 80 to 500



# Experiment 3

Aim - Calculate total number of captured packets for each protocol.



* + **Ethernet** (1417 packets)
    - **Internet Protocol Version 4 (IPv4)** (1416 packets, 99.9%)
      * **User Datagram Protocol (UDP)** (16 packets, 1.1%)
        + **Domain Name System (DNS)** (12 packets, 0.8%)
        + **Data** (4 packets, 0.3%)
      * **Transmission Control Protocol (TCP)** (1400 packets, 98.8%)
        + **Transport Layer Security (TLS)** (410 packets, 28.9%) → **HTTPS**
        + **Hypertext Transfer Protocol (HTTP)** (10 packets, 0.7%)

**Line-based text data** (1 packet, 0.1%)

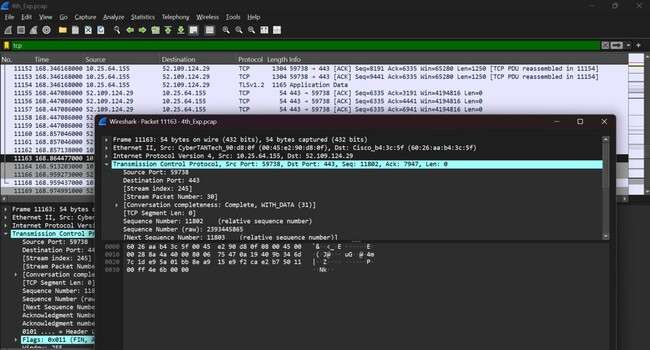
**Data** (4 packets, 0.3%)

* + - * + **X11** (3 packets, 0.2%)
        + **Data** (4 packets, 0.3%)
    - **Internet Protocol Version 6 (IPv6)** (1 packet, 0.1%)
      * **User Datagram Protocol (UDP)** (1 packet, 0.1%)
        + **DHCPv6** (1 packet, 0.1%)
  1. **Q Virtual LAN** (123 packets, 8.7%)

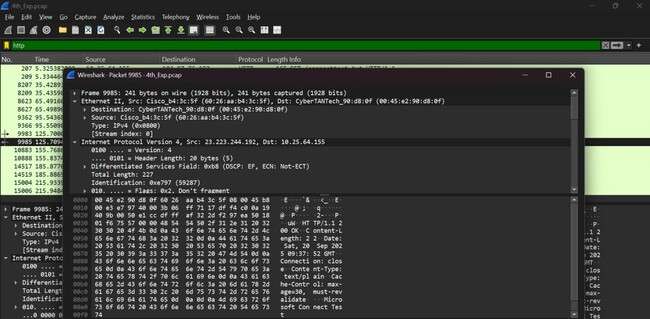
# Experiment 4

Aim – Select and highlight one packet each for the different protocols like ARP, TCP, UDP, HTTP, DNS etc.

#### TCP Packet

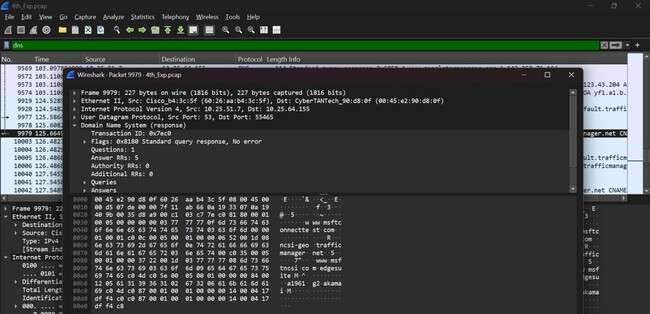


#### HTTPS Packet

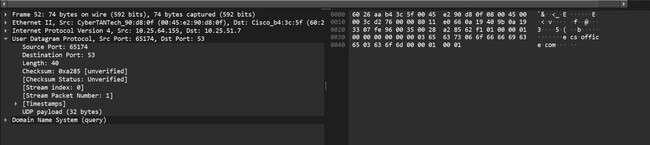


\

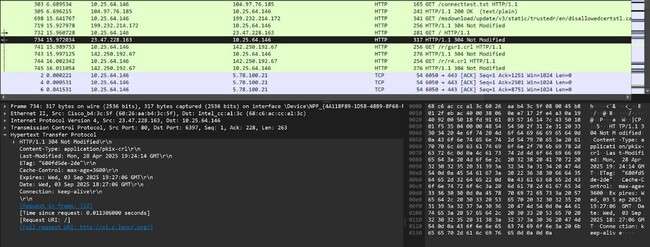
#### DNS Packet



#### UDP Packet



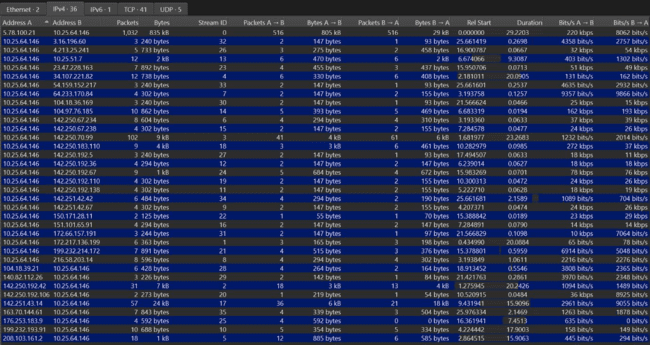
HTTP Packet



# Experiment 5

Aim – To Find out the IP addresses of the client and server using the statistics tool of Wireshark.

#### Conversations in Wireshark



Address A - Source IP Address Address B - Destination IP Address

Steps:

* + 1. **Open Wireshark** → Load your capture file.
    2. Go to **Statistics → Conversations**.
    3. Select the **IPv4** tab (or IPv6 if applicable).
    4. You’ll see a table with **Source and Destination IP addresses**.
    5. Identify:

**Client IP** → usually your machine’s local/private IP.

**Server IP** → the remote/public IP (website or server you connected to).

#### Additional info:

* + - * From the top menu, I went to **Statistics → Conversations**.
      * In the window that appeared, I clicked on the **IPv4 tab** to see all the IP conversations.
      * The table showed two columns, **Address A** and **Address B**, which represent the devices communicating.
      * By checking the number of packets and data size, I identified which IP was sending requests (the client) and which IP was replying with more data (the server).

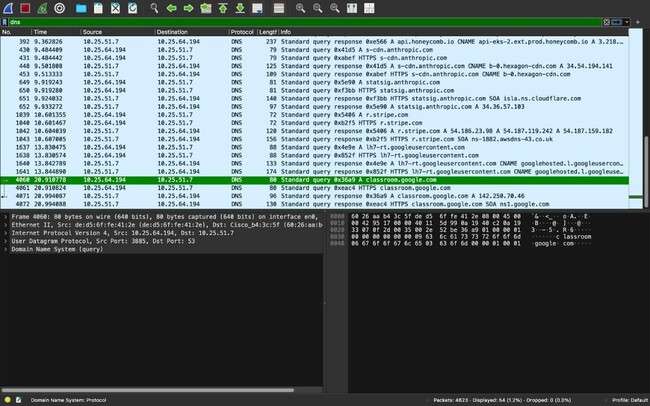
Finally, we were able to identify the IP address of the **client** (requesting device) and the **server**

(responding device).

This confirms how Wireshark’s statistics tool helps us clearly distinguish communication roles in a network.

# Experiment 6

### Aim – Identify the Internet (IP) address of the URLs you visited during the capture and list the IP address with the site URL.



## Open Wireshark capture file (.pcap) → Load the capture where your browsing traffic was recorded.

## Apply DNS filter → Type dns in the display filter bar to see domain name queries and responses.

## Look at “Info” column → Find the domain names (like googleads.g.doubleclick.net, learning.sap.com, etc.).

## Check response packets → Select the DNS response packets (not just queries) to see the IP addresses resolved for each domain.

## Note down results → For each URL, match the domain name → IP address from the DNS response.

## Create table/report → List all visited site URLs alongside their resolved IP addresses.

## More explanation:

## DNS Request & Response Timing: I analyzed the time difference between the query and response to understand my network's performance.

## Packet Count per URL: I counted multiple queries for a single site to see how complex its infrastructure is.

## Record Types: I identified different record to reveal more about how the websites were hosted.

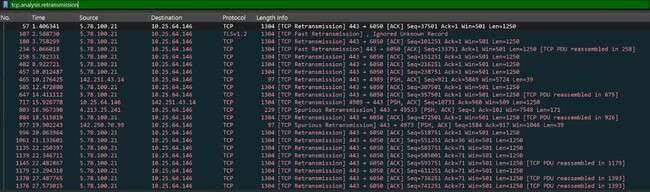
## Source and Destination IP: I noted my computer's IP as the source and the DNS server's IP as the destination.

## We successfully identified the specific IP addresses corresponding to various online services and websites by analyzing DNS queries. This shows how our computer resolves domain names into numerical addresses to access content on the internet, providing a clear demonstration of a fundamental networking process.

# Experiment 7

### Aim – To Evaluate the total Number of lost packets using Wireshark.

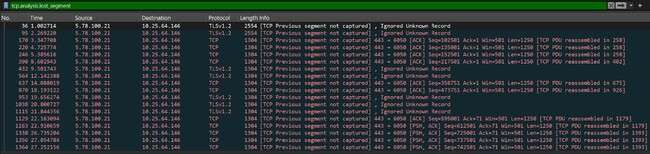
## Retransmitted TCP Packets in Wireshark



### **Filter Applied:** The tcp.analysis.retransmission filter highlights packets that have been resent by the source.

### **Underlying Causes:** TCP segments are retransmitted due to either the sender's retransmission timer expiring or the receiver sending duplicate ACKs. This mechanism ensures reliable data delivery by resending lost original data.

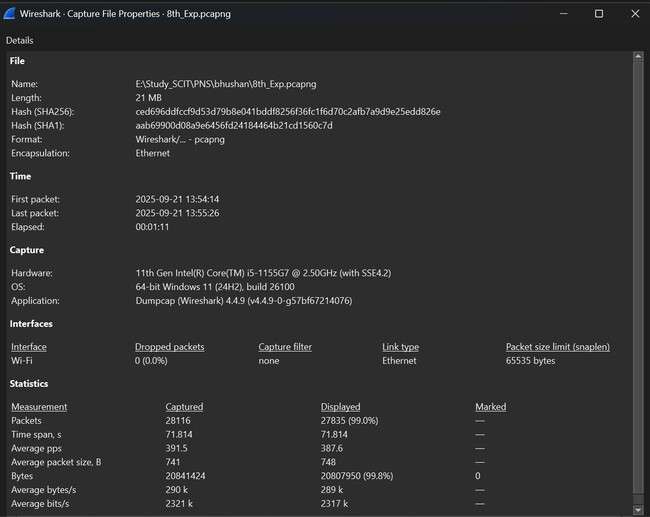
## Lost TCP Segments in Wireshark



### **Filter Used:** The tcp.analysis.lost\_segment filter is applied to identify packets where a TCP segment was anticipated but not received.

### **Cause:** The **Info** column for several packets shows [TCP Previous segment not captured]. This means that a packet that was part of a TCP conversation did not arrive at your machine. As a result, Wireshark and the receiving application (your browser, for example) detected a gap in the TCP sequence numbers.

## Captured Packet Info

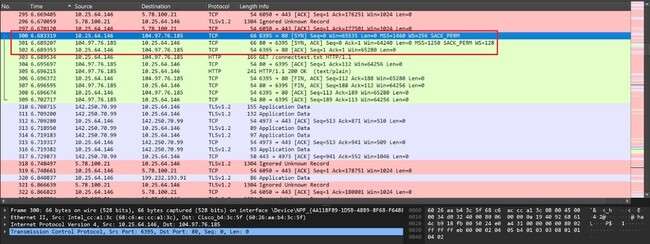


### **Dropped Packets**: Regarding dropped packets, the most important detail is that 0 packets (0.0%) were dropped by the capture interface. This indicates that the computer's network card and the Wireshark application were able to handle all incoming traffic without dropping any packets before Wireshark could process them. This is a crucial distinction. The dropped packets shown in other screenshots are due to network issues, not due to the capture setup.

# Experiment 8

Aim – To Capture one TCP 3-way handshake and explain the process.

#### Identified TCP three-way handshake



The **TCP 3-way handshake** is a fundamental synchronization protocol for establishing a reliable connection in a TCP/IP network. This process, also known as a three-step handshake, is crucial for ensuring that both the sender and receiver are ready to transfer data and that their initial sequence numbers are synchronized.

The process unfolds in three sequential steps:

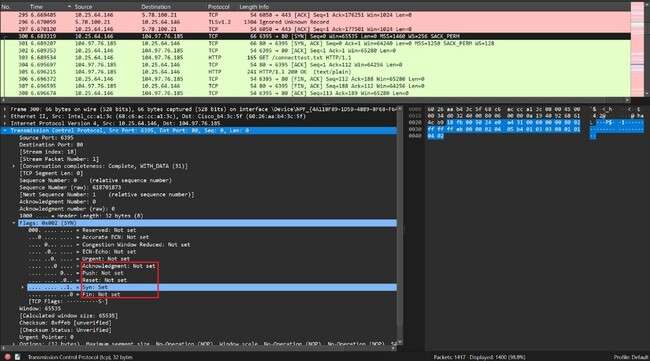
1. **SYN (Synchronize):** The client initiates the connection by sending a TCP segment with the

**SYN** flag set to the server. This packet contains the client's initial sequence number.

1. **SYN-ACK (Synchronize-Acknowledge):** Upon receiving the SYN segment, the server responds with a segment where both the **SYN** and **ACK** flags are set. The SYN flag is set to synchronize the server's sequence number with the client, and the ACK flag is set to acknowledge the client's initial SYN packet. This segment contains the server's own initial sequence number and an acknowledgment number derived from the client's sequence number.

**ACK (Acknowledge):** The client completes the handshake by sending a final segment with the **ACK** flag set. This acknowledges the server's SYN-ACK segment, and the connection is then considered fully established and ready for data transfer.

#### SYN flag set

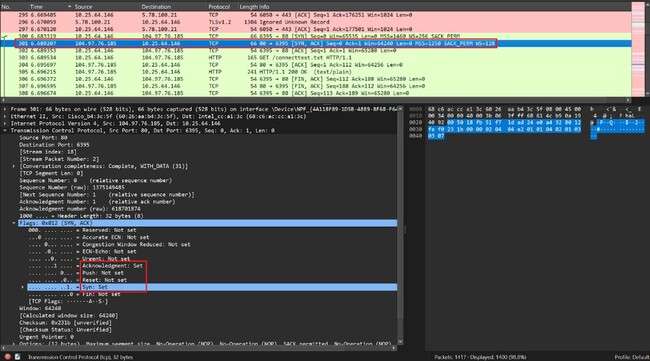


#### Packet 1: SYN (Synchronize)

Packet **300** initiates the handshake.

* + **Source and Destination:** The client (10.25.64.146) sends this packet to the server (104.97.76.185).
  + **TCP Flags:** The **SYN (Synchronize)** flag is set, which signals the intention to establish a new connection.

#### SYN and ACK flag set



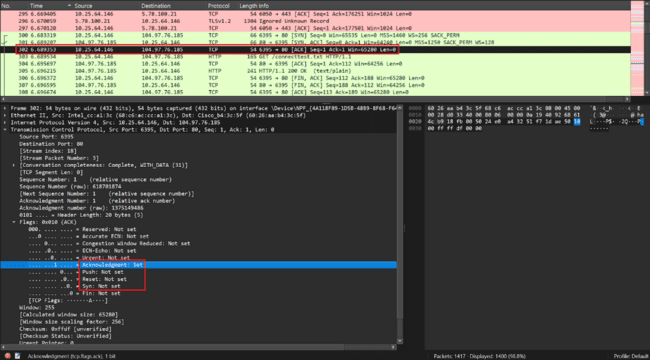
#### Packet 2: SYN-ACK (Synchronize-Acknowledge)

Packet **301** is the server's response to the client's request.

* + **Source and Destination:** The server (104.97.76.185) responds to the client (10.25.64.146).

**TCP Flags:** Both the **SYN** and **ACK (Acknowledgment)** flags are set. The **SYN** flag is set to synchronize the server's sequence number with the client. The **ACK** flag is set to acknowledge the client's initial SYN packet.

#### ACK flag set



#### Packet 3: ACK (Acknowledge)

Packet **302** is the final step, completing the handshake.

* + **Source and Destination:** The client (10.25.64.146) sends this final packet to the server (104.97.76.185).
  + **TCP Flags:** Only the **ACK** flag is set. This acknowledges the server's SYN-ACK packet and confirms the client's readiness to begin data transfer.

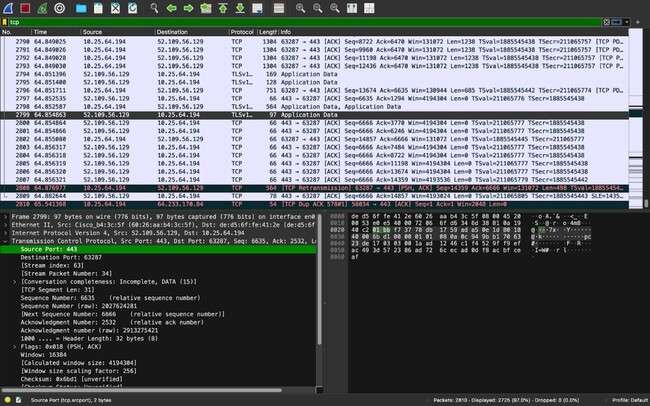
# Experiment 9

Aim – Select one TCP packet and explore the features in the packet header window. Identify the TCP header and IP header details for the selected packet.

#### Same packet from the three-way handshake

#### TCP Header Details TCP header

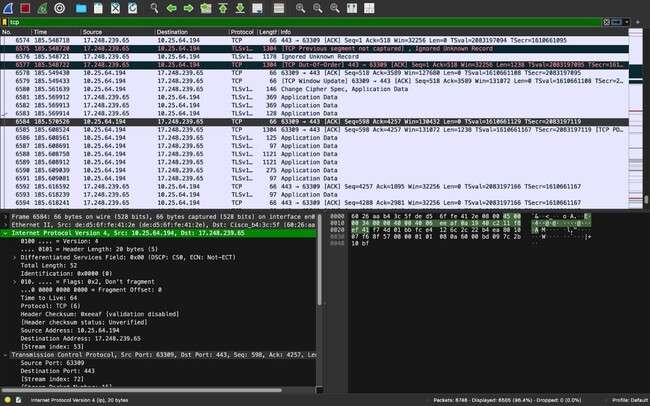
* **Source Port:** The port number of the sender.
* **Destination Port:** The port number of the receiver.
* **Sequence Number:** The sequence number of the first byte in the segment.
* **Acknowledgment Number:** The next sequence number that the sender is expecting.
* **Data Offset:** Specifies the size of the TCP header.
* **Flags:** Various control flags (e.g., SYN, ACK, FIN).
* **Window Size:** The size of the sender's receive window (flow control).
* **Checksum:** Used for error-checking the header and data.



#### IP Header Details.

* **Version**: The IP version (usually 4 or 6).
* **Header Length:** The length of the IP header.
* **Source IP Address:** The IP address of the sender.
* **Destination IP Address:** The IP address of the receiver.
* **Total Length:** The total length of the IP packet.
* **Protocol:** Identifies the protocol (e.g., TCP or UDP).
* **Time to Live (TTL):** The maximum time the packet is allowed to traverse the network.

#### IP Header of the captured packet

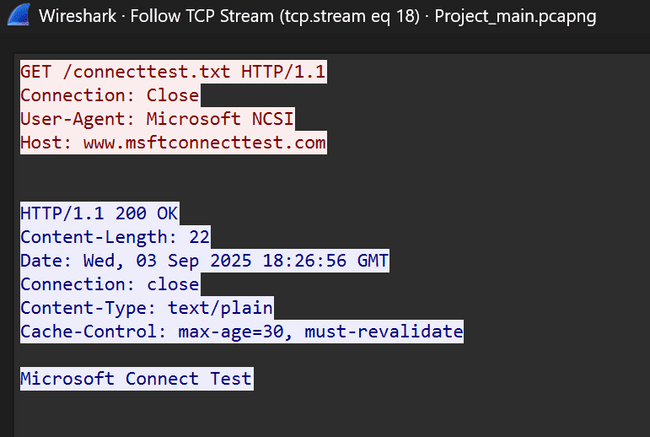
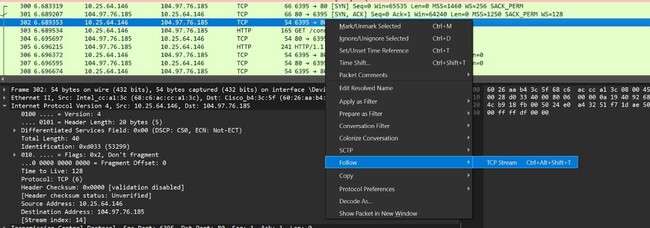


#### Steps:

1. Open the capture file in Wireshark.
2. Apply TCP filter → Type tcp in the filter bar to show only TCP packets.
3. Select a TCP packet (as shown in your screenshot, e.g., packet with source 172.253.118.188 and destination 10.25.64.194).
4. Expand “Transmission Control Protocol” section in the middle pane.
   * Note down: Source Port, Destination Port, Sequence Number, Acknowledgment Number, Flags (SYN, ACK, etc.), Window Size, Checksum
5. Expand “Internet Protocol” section for the same packet. Note down: Source IP address, Destination IP address

# Experiment 10

Aim – To Try exploring the Follow TCP stream feature in Wireshark, and explain your findings.



The second screenshot presents the reconstructed TCP stream, revealing a standard **HTTP/1.1 (Hypertext Transfer Protocol)** transaction. The content of this stream can be systematically categorized into two distinct parts: the client's request and the server's response.

1. **Client Request:** The communication is initiated by a client, identified by the User-Agent string as Microsoft NCSI (Network Connectivity Status Indicator). The client's request is a

GET method directed at the resource /connecttest.txt on the host [www.msftconnecttest.com.](http://www.msftconnecttest.com/) This specific URL is a known endpoint used by Microsoft Windows to verify internet connectivity. The inclusion of the Connection: Close header signals the client's intent to terminate the TCP session immediately after receiving the response.

**Server Response:** The server's response begins with the status line HTTP/1.1 200 OK, confirming the successful processing of the client's request. The response headers provide essential metadata, including a Content-Length of 22 bytes and a Content-Type of text/plain. The Cache-Control header further dictates caching policies for the retrieved resource. The body of the response, which is the requested resource itself, contains the text "Microsoft Connect Test".