# Wireshark Traffic Analysis Project Report

~ Olina Kundu

## Objective:

The objective of the project is to analyse live network traffic using Wireshark to understand the behaviour and structure of common network protocols such as TCP, DNS, HTTP, and ICMP. The goal is to build foundational skills in traffic inspection and packet-level analysis.

## Environmental setup:

* Operating system: Windows 11
* Tool used: Wireshark
* Network Interface Used: Wi-Fi
* Traffic generated during capture: Visited websites : google.com, example.com.

Pinged 8.8.8.8 and ran DNS queries using nslookup.

Used YouTube/ other applications to generate varied traffic.

* Capture Duration: ~ 3 minutes
* Capture File: wireshark\_capture.pcapng

## Packet Analysis

### TCP Handshake Analysis:

The TCP 3-way handshake was observed within **TCP stream 40**, beginning at **Packet No. 470**, as highlighted in the screenshot below.

* **Packet 470 (SYN):** The client (192.168.1.4) sends a SYN packet to initiate a TCP connection with the server (23.192.228.80) on port 80.
* **Packet 504 (SYN, ACK):** The server replies with a SYN-ACK, acknowledging the request and offering its own sequence number.
* **Packet 505 (ACK):** The client responds with a final ACK, completing the 3-way handshake.

**Handshake Details:**

* Source Port: 51603
* Destination Port: 80 (HTTP)
* Sequence Number: 0 → 1
* Acknowledgment Number: 1
* Window Size: 65535 (initial), 64240 (from server)
* MSS(Maximum Segment Size): 1460
* SACK Permitted (Selective Acknowledgment): Yes
* TCP Timestamps: Present

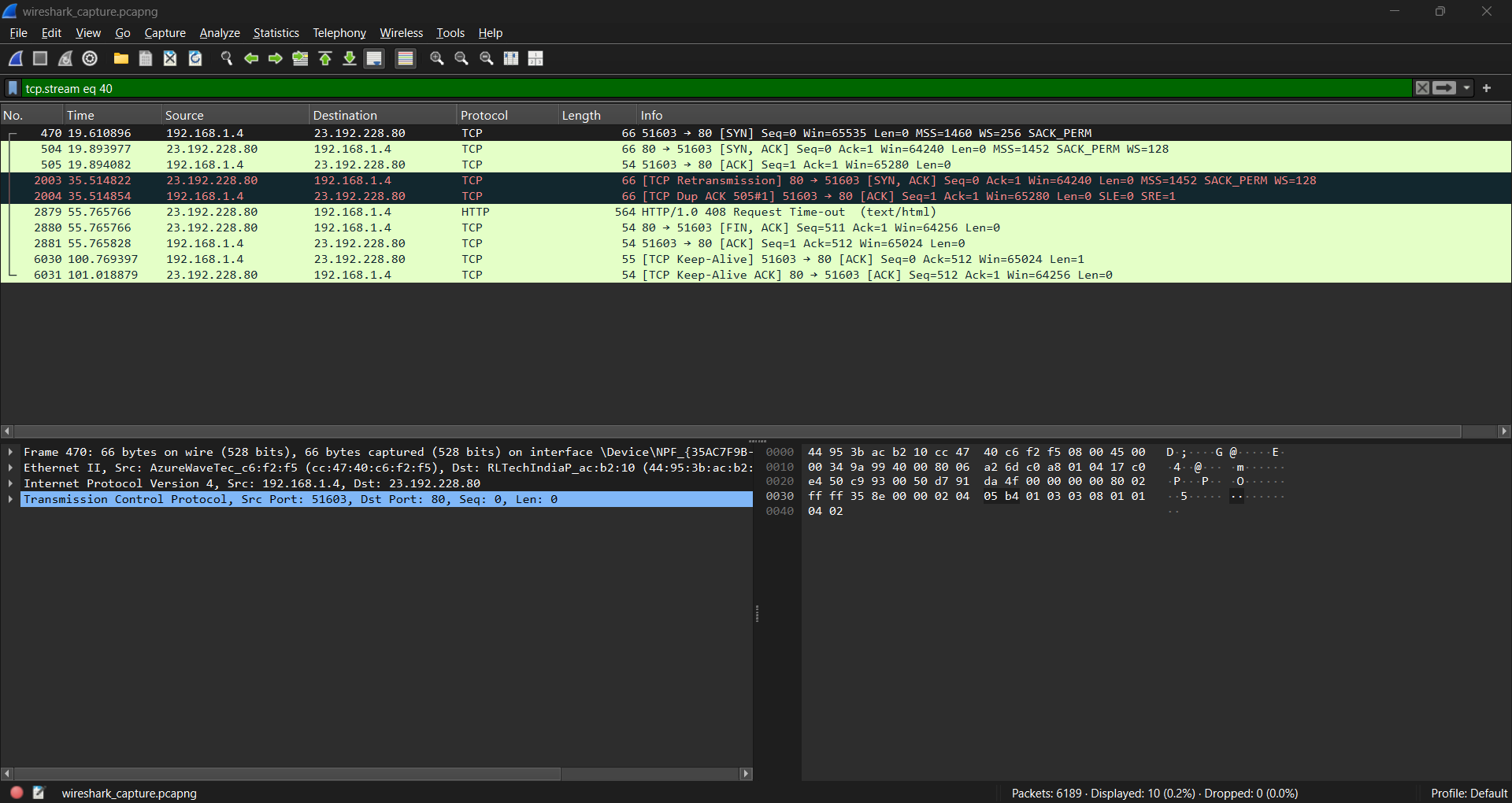
This handshake establishes a reliable TCP connection between the host machine and the web server, following the standard TCP protocol.

The SYN packet from the client (Packet 470) initiates the handshake with a sequence number of 0. It has only the SYN flag set and includes options like MSS (1460), SACK permitted, window scaling, and timestamps. The advertised window size is 65535 bytes.

The SYN-ACK from the server (Packet 504) responds with a sequence number of 0 and an acknowledgment number of 1. Both SYN and ACK flags are set. It mirrors similar TCP options and advertises a window size of 64240 bytes.

The final ACK from the client (Packet 505) completes the handshake with a sequence number of 1 and an acknowledgment number of 1. Only the ACK flag is set, and the window size is now adjusted to 65280 bytes.

*Refer to the screenshot below showing Packet 470, 504, and 505.*

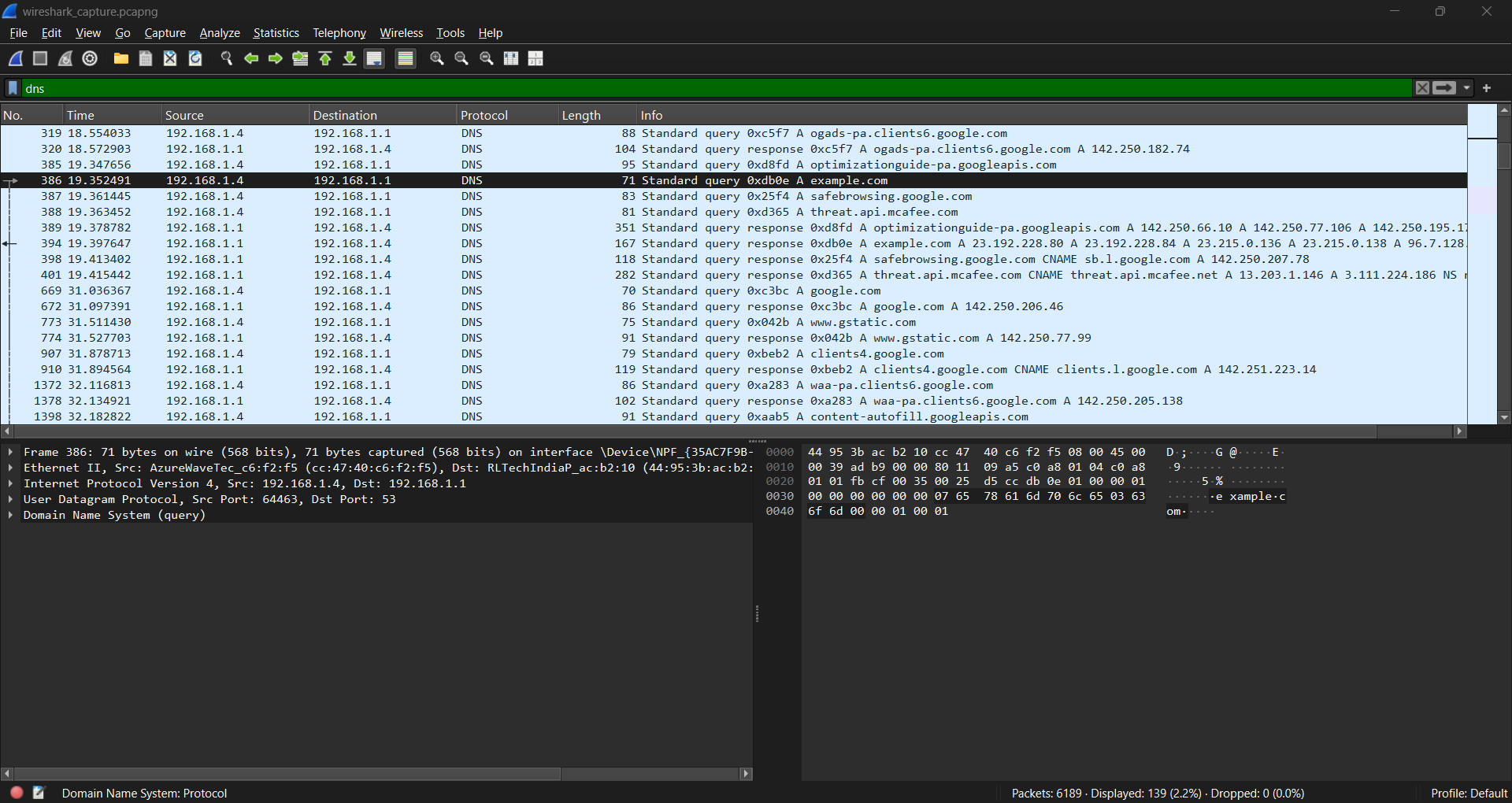


### DNS Query Analysis

From the screenshot below, a DNS query was observed in **Packet 386**, where the host 192.168.1.4 sent a standard query to the DNS server 192.168.1.1 requesting the IPv4 address (A record) for the domain example.com. This was transmitted over UDP, using **source port 64463** and **destination port 53**. The query had a unique identifier 0xd8be and followed standard DNS formatting.

The corresponding response appears in **Packet 394**, in which the DNS server replies with an A record for example.com, resolving it to IP addresses including 23.192.228.80, 23.215.0.136, and others. The response matches the transaction ID and confirms a successful resolution, as indicated by the “No error” code.

This DNS exchange demonstrates how a client converts a domain name into one or more usable IP addresses using UDP-based DNS, a fundamental step before establishing any actual network connection with the server.

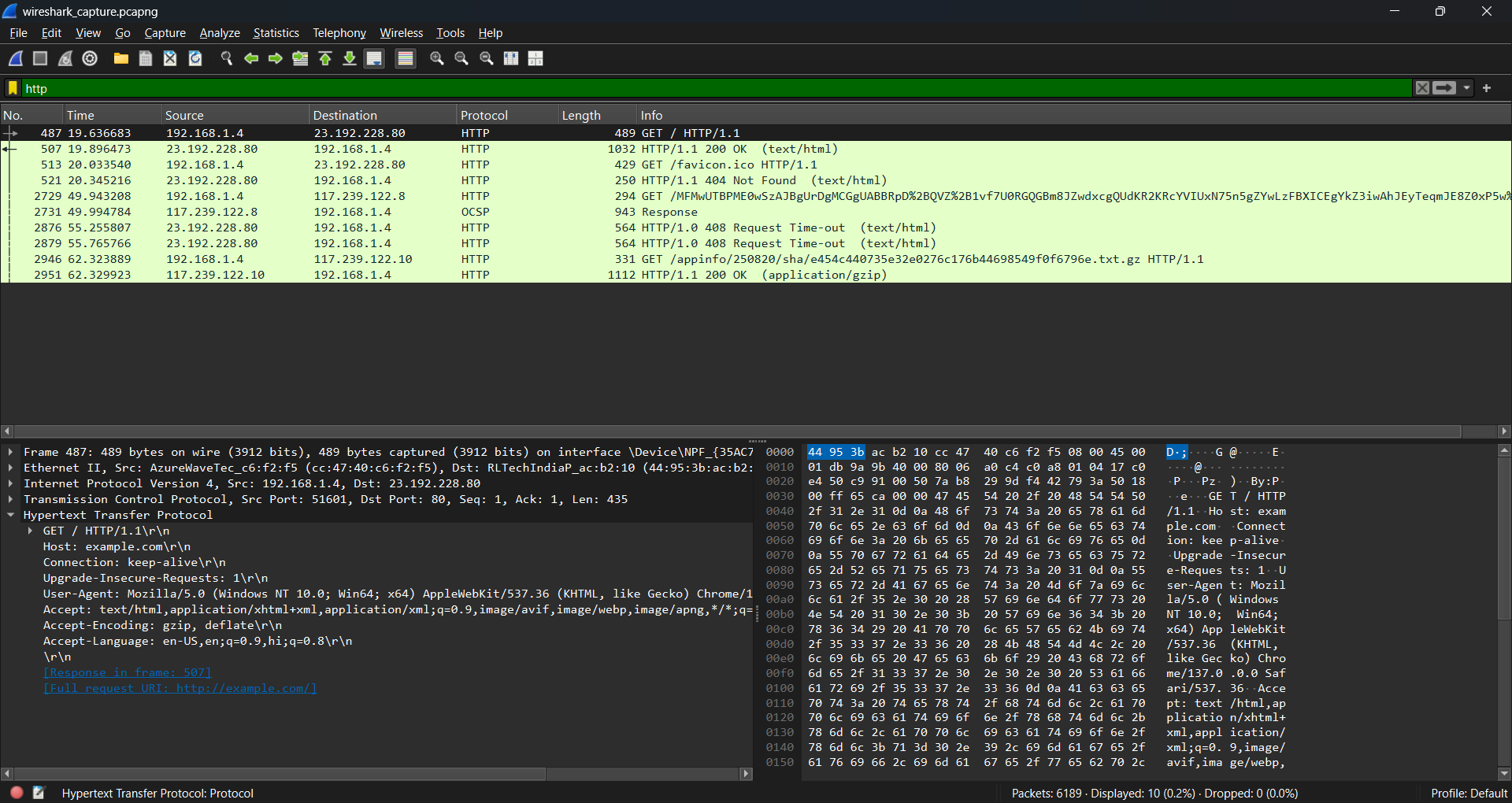


### HTTP Request/Response Analysis

In Packet 487, the client 192.168.1.4 initiates an HTTP session by sending a **GET** request to the server at 23.192.228.80 over port 80. The request targets the path / using the HTTP/1.1 protocol. Several headers are included, such as Host: example.com, Connection: keep-alive, and a detailed User-Agent string identifying the browser and system. This request is carried over TCP, with the client’s source port being 51601.

The server replies in Packet 507 with a standard **HTTP/1.1 200 OK** response, indicating the request was successfully processed. The response includes headers like Content-Type: text/html and a content length, signaling that the body contains an HTML page. This completes a typical HTTP transaction — a stateless request-response model layered on top of TCP.

This exchange highlights how browsers fetch web content by using plain-text HTTP messages structured with headers and status codes, forming the foundation of web communication.

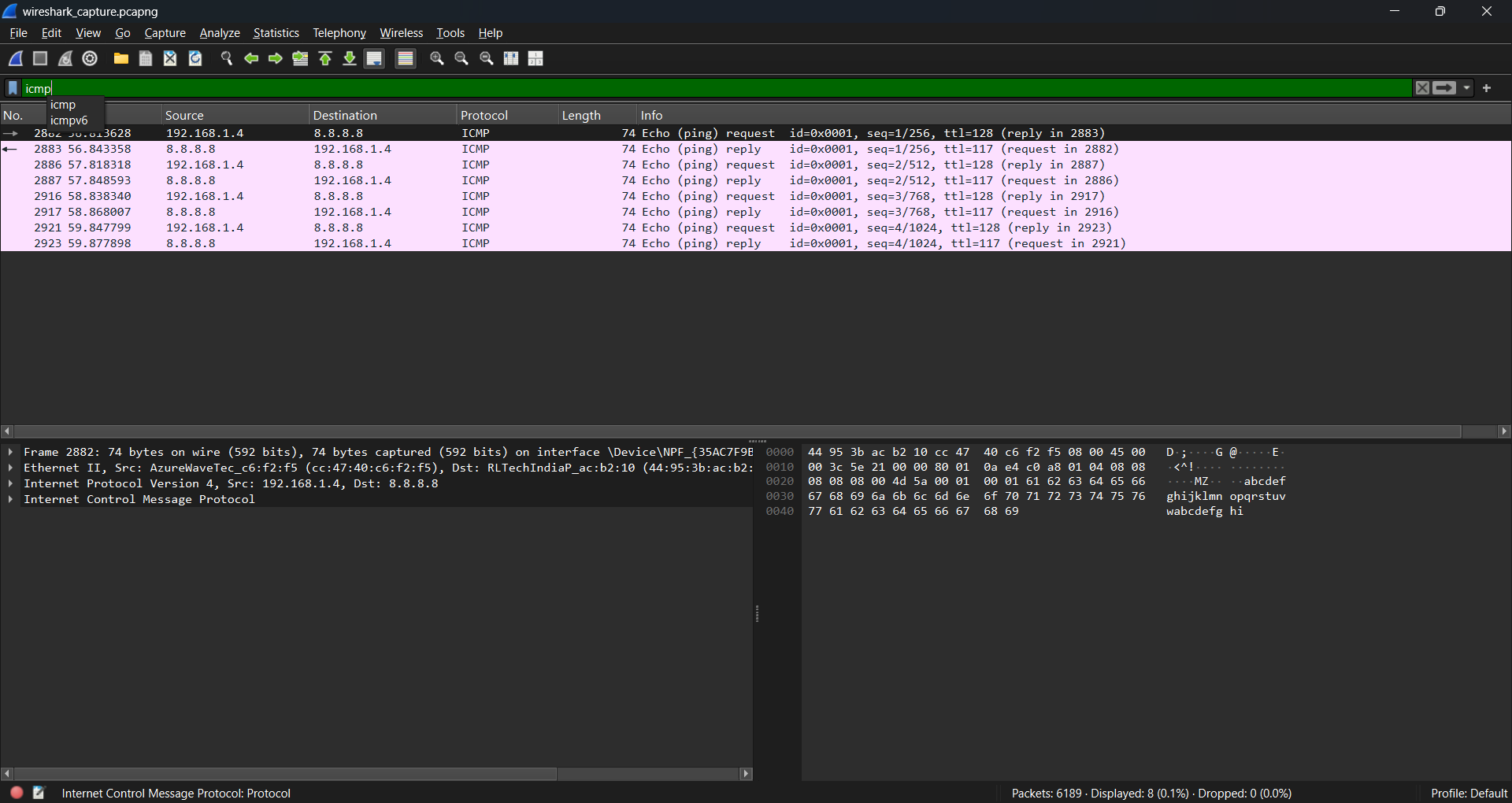


### ICMP Echo Request/Reply Analysis

In Packet 2882, the host 192.168.1.4 sends an **ICMP Echo Request** to Google's public DNS server at 8.8.8.8. This is part of a standard "ping" operation, used to test connectivity and measure round-trip time. The packet includes an identifier (id=0x0001) and sequence number (seq=1) for tracking. The TTL (Time-To-Live) is set to 128, and the payload contains ASCII characters (a-z) for testing integrity.

Packet 2883 shows the corresponding **ICMP Echo Reply** from 8.8.8.8, confirming successful delivery and return of the request. The reply matches the identifier and sequence number, and the TTL is now 117, indicating a hop count of 11 between the client and destination.

This exchange demonstrates the functionality of the **Internet Control Message Protocol (ICMP)** in diagnosing network reachability and latency between nodes.



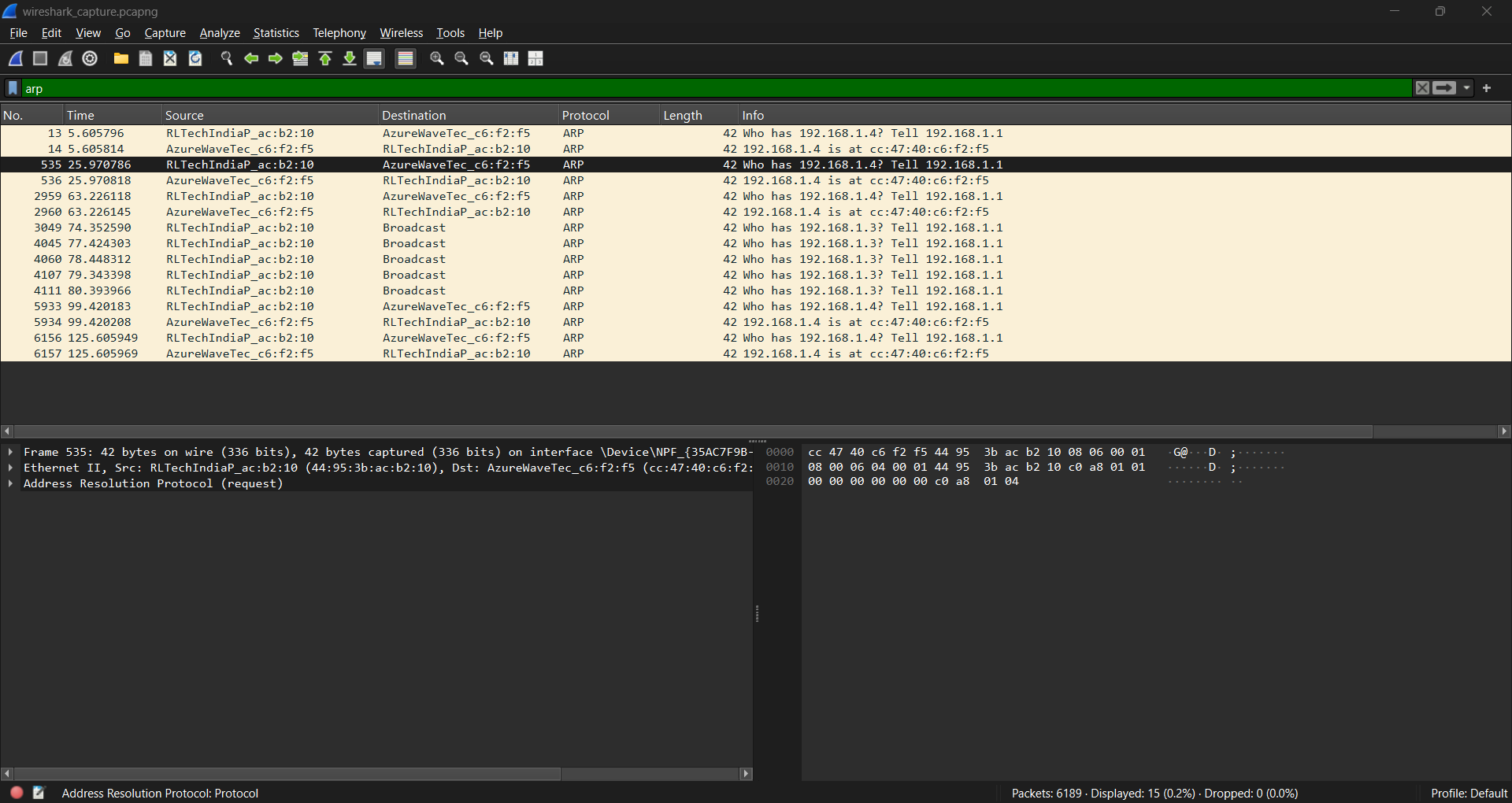
### Address Resolution Protocol (ARP)

In this capture, we observe a typical ARP exchange between two devices on the local network. ARP is used to map IP addresses to MAC addresses within the same subnet.

In **packet 535**, the device with IP address 192.168.1.1 sends an ARP **request** asking,  
**"Who has 192.168.1.4? Tell 192.168.1.1"**.  
This means the sender (likely the router) wants to know the MAC address of the device with IP 192.168.1.4.

In **packet 536**, we see the **ARP reply** from 192.168.1.4, stating:  
**"192.168.1.4 is at cc:47:40:c6:f2:f5"**, which provides the MAC address of the host.

This ARP exchange helps devices communicate on a local network by resolving IPs into physical hardware addresses.



## Conclusion

This Wireshark traffic analysis successfully captured and examined multiple key protocols used in day-to-day network communication. Through the filtered packet inspection and breakdown of different interactions, the following observations were made:

* **TCP Handshake:** A complete three-way handshake was observed between the host (192.168.1.4) and external servers, confirming the establishment of reliable connections.
* **DNS Resolution:** Multiple domain name lookups (e.g., example.com, googleapis.com) occurred, with responses successfully returned by the DNS server (192.168.1.1).
* **HTTP Traffic:** Application-layer communication over HTTP was visible, with GET requests and proper server responses including 200 OK and 404 Not Found.
* **ICMP Packets:** Echo request-reply pairs (ping) between the host and public IP (8.8.8.8) verified basic connectivity and round-trip time.
* **ARP Exchanges:** ARP requests and replies were captured, resolving IP-to-MAC address mappings necessary for LAN communication.
* **DHCP Absence:** No DHCP handshake was found in the capture, likely because the host had already obtained a valid IP before capturing started.

Overall, the exercise highlighted the layered nature of network communication and reinforced how protocols operate and interact in real-world traffic. Tools like Wireshark provide crucial insights into these processes, making them invaluable for anyone studying or working in networking and cybersecurity.