TRAFFIC CAPTURE ANALYSIS WITH WIRESHARK AND TCPDUMP

OBJECTIVE: To learn how to utilize various packet capture and protocol analyzer tools, such as Wireshark and Tcpdump, and help identify risk, vulnerabilities, and threats associated with traffic.

Project Description: This project contains two traffic captures, one captured with Wireshark and the other with tcpdump. These captures were saved as packet capture (pcap) files and later used for protocol analysis using Wireshark filters and tcpdump capture filters, as further documented in this report.

Tools Used:

- Wireshark
- Tcpdump
- Snipping Tool
- Kali VM

Tools Used	**Description**
Wireshark	This is a cross-platform network protocol analyzer tool that provides a user-friendly Graphical User Interface(GUI) for users to analyze and filter traffic running on their networks, and it is perfect for anomaly detection, troubleshooting network issues, protocol analysis, and security analysis, such as spotting misconfigurations, identifying malware
Tcpdump	This is a Command Line Interface (CLI) Network packet capture and protocol analyzer that is capable of capturing live traffic and widely used for network troubleshooting, security monitoring, and is particularly suited to Unix-like operating Systems

Snipping Tool	This was used to capture various vital screenshots shown in this report.
Kali VM	This is the Virtual Machine OS used to practice tcpdump commands and capture its traffic.

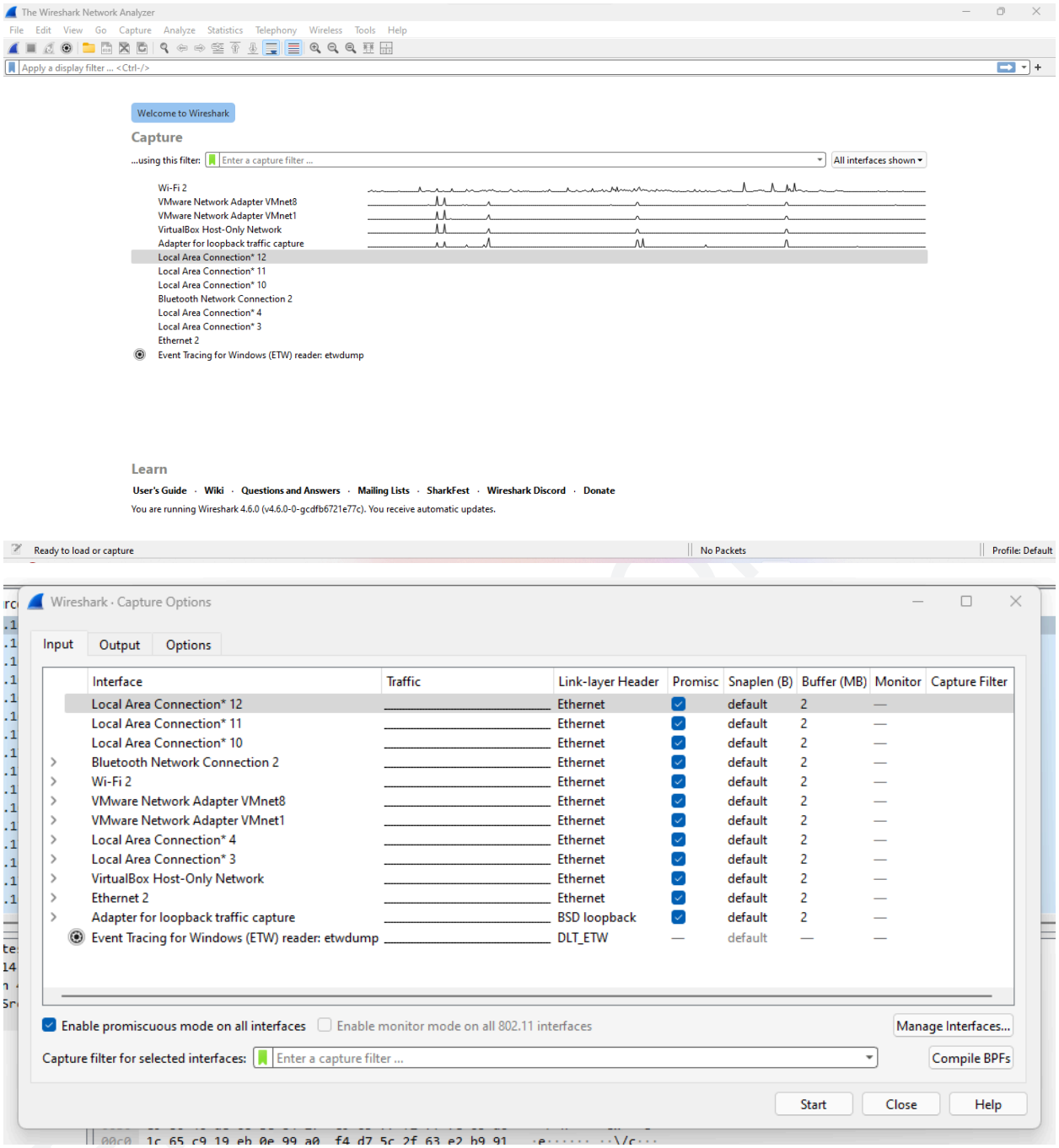
REPORT

WIRESHARK

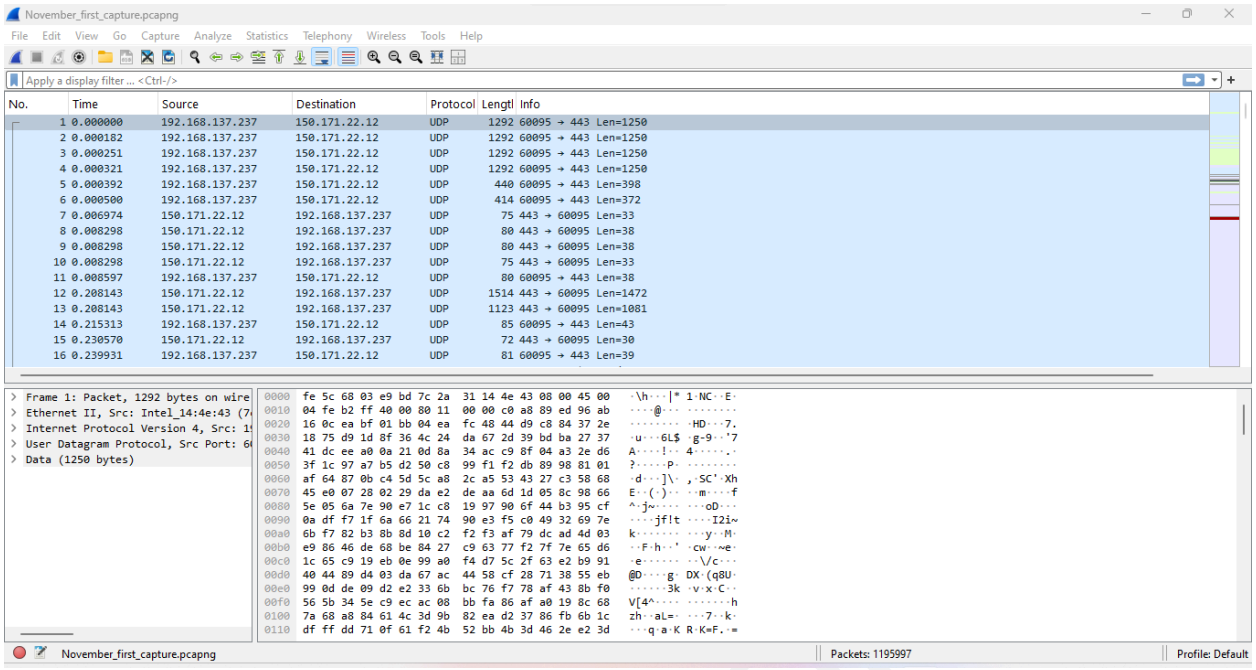
Wireshark is a cross-platform GUI protocol analyzer that provides various filters for analyzing protocols and traffic, and also supports colorization to identify packet types. And by default, they are:

Color in Wireshark	Packet Types
Light Purple	TCP
Light Blue	UDP
Black	Packet with errors
Light Green	HTTP traffic
Light Yellow	Window-Specific traffic, including server message block (SMB) and NetBios
Dark Yellow	Routing
Dark Gray	TCP, SYN, FIN, and ACk traffic

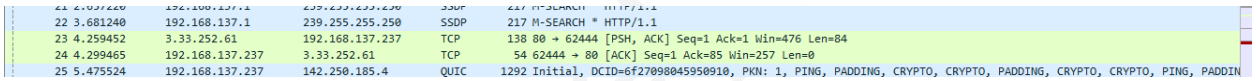
The Wireshark dashboard, displayed in the snippet, presents the various interfaces available for traffic capture. Interfaces currently experiencing traffic flow are indicated by zigzag lines. To begin the capture process, click the icon resembling the main Wireshark icon. For optimal results, it is crucial to set the network interface to **promiscuous mode**. To do this, click **"Capture"** on the taskbar, select **"Options,"** and then ensure that **"Enable Promiscuous mode on all interfaces"** is checked, as if not selected the network card will only capture traffic sent to its own network addresses.



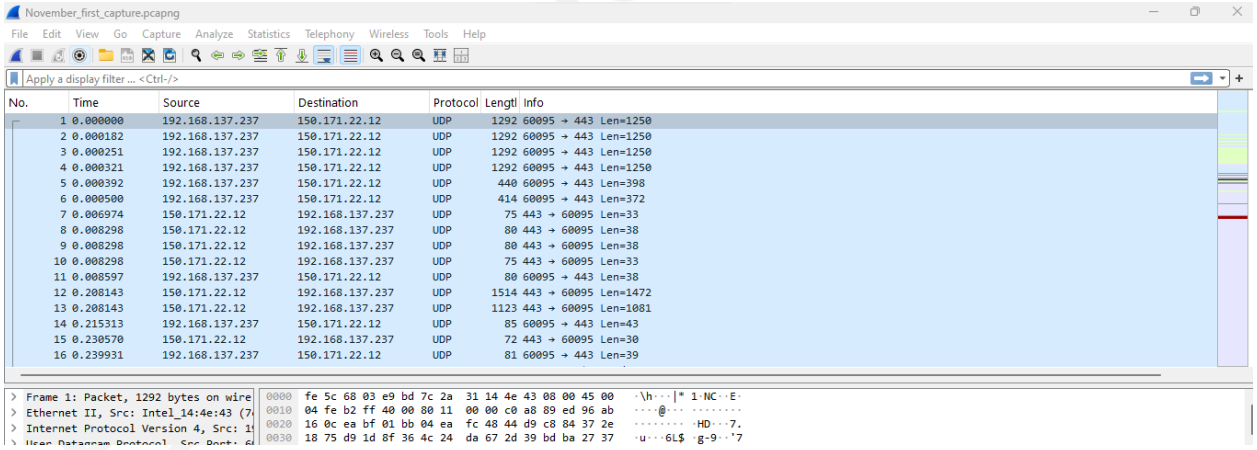
The project utilizes a pre-existing PCAP file for analysis. This file, which contains **1,195,997** captured packets of various traffic types, was collected over a three-hour duration during a live YouTube program.



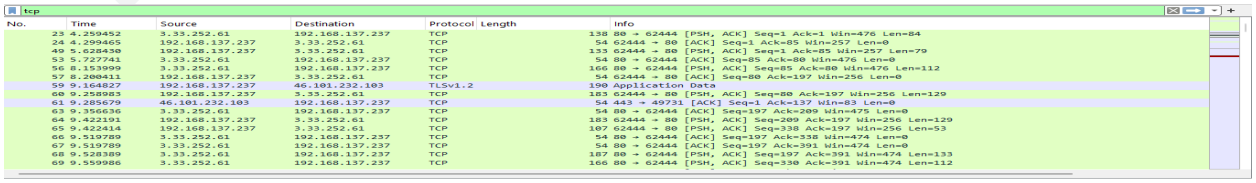
Various Color codes are displayed through the following snippet, ranging from ***Light Yellow***, showing the packet type of Window-Specific traffic.



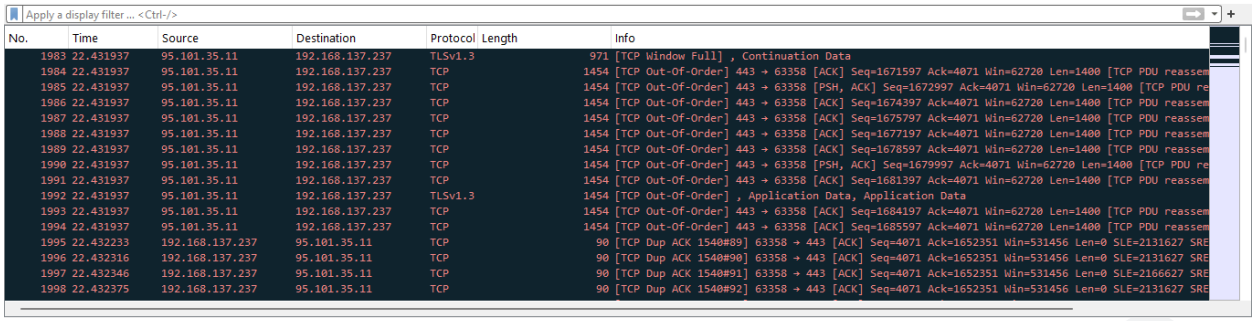
To ***Light Blue***, which shows the packet type of UDP:



To ***Light Green***, which shows the packet type of HTTP traffic



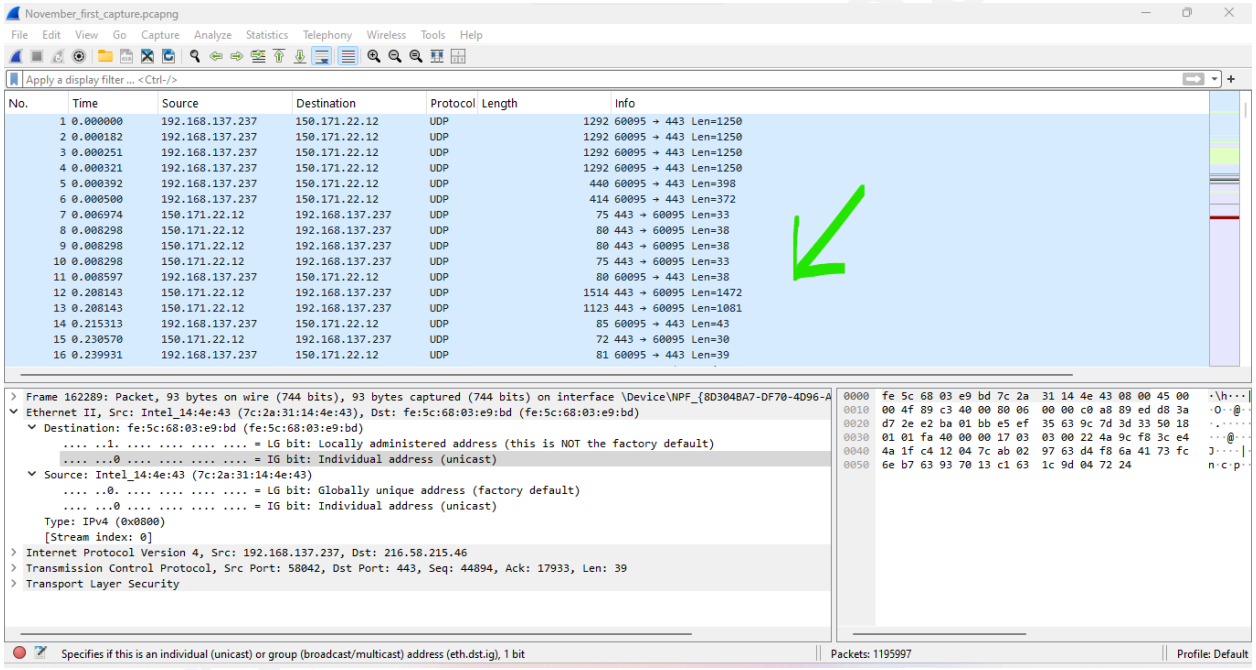
To ***Black***, which identifies the packet type that contains errors



The Wireshark Packet Capture Dashboard

The Wireshark packet capture dashboard consists of three main sessions, which include: the packet list pane, the packet details pane, packet bytes pane

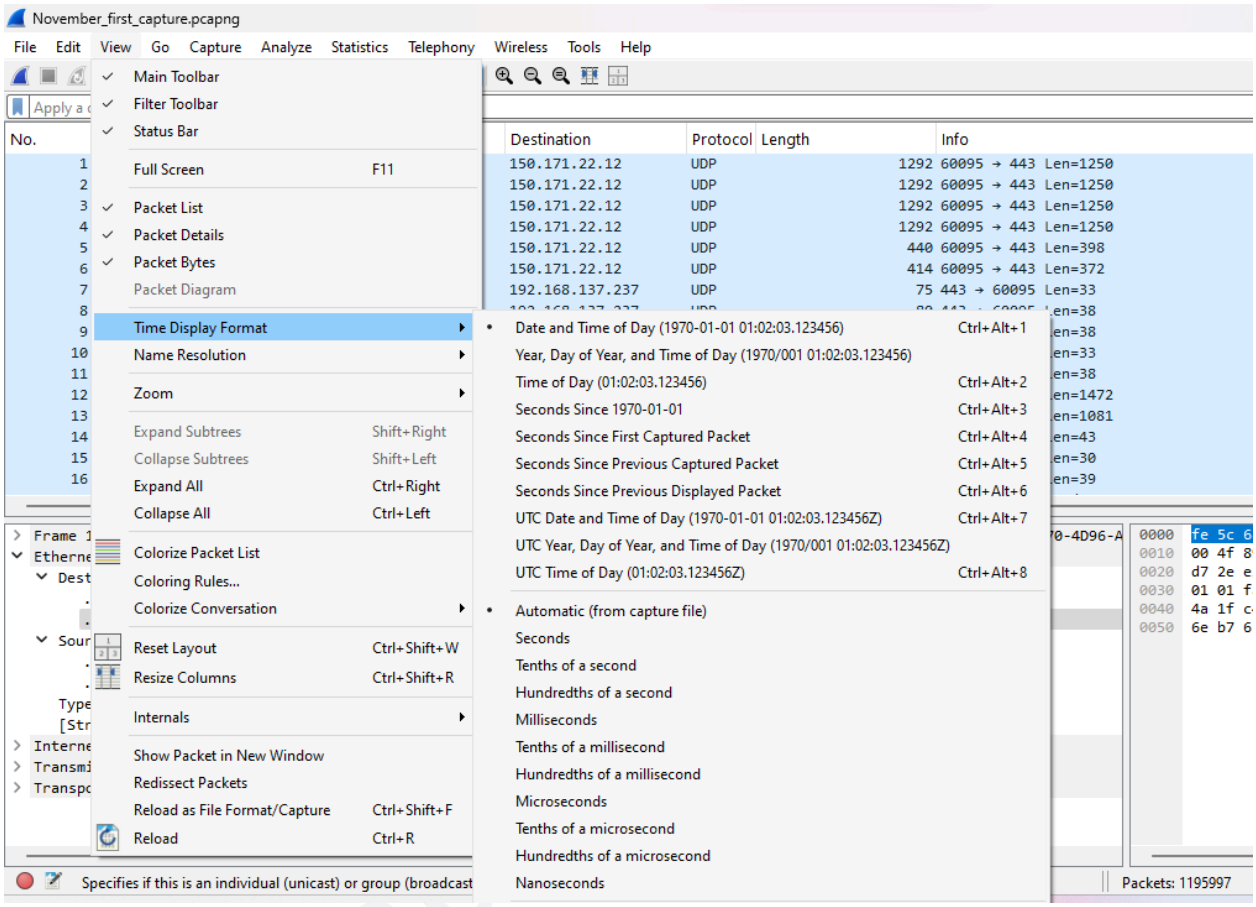
The Packet List Pane



The packet list pane is the portion of the snippet mapped out by the green marker, and it gives the summary view of the packet capture, as it would display a list of all packets captured in the file and contain the following details

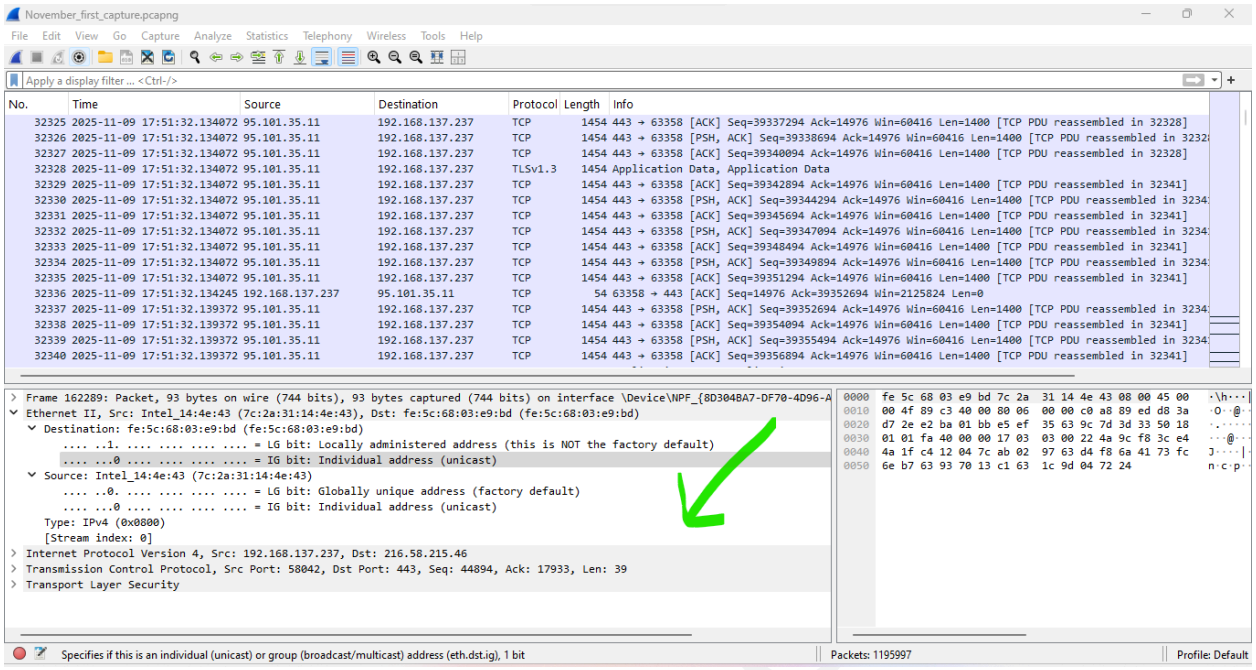
- **No. (Number):** This is the index number assigned to a packet as it is received and serves the purpose of a serial number.
- **Time:** This is the timestamp of when the packet was captured, and it is measured in ***microseconds*** by default. To adjust the time

display format, go to **View**, then select **Time Display format** and choose the option that best suits your needs.



- **Source:** This is the IP address of the endpoint sending the packet
- **Destination:** This is the IP address of the endpoint receiving the packet
- **Protocol:** This is a protocol (established language for communication) used to communicate between the sending and receiving endpoints, such as ARP, DNS, TCP, UDP, and many more
- **Length:** This represents the size of the packet in bytes
- **Info:** This would provide a brief and short summary of the content of the packet that can help us understand the purpose or what is happening with the packet

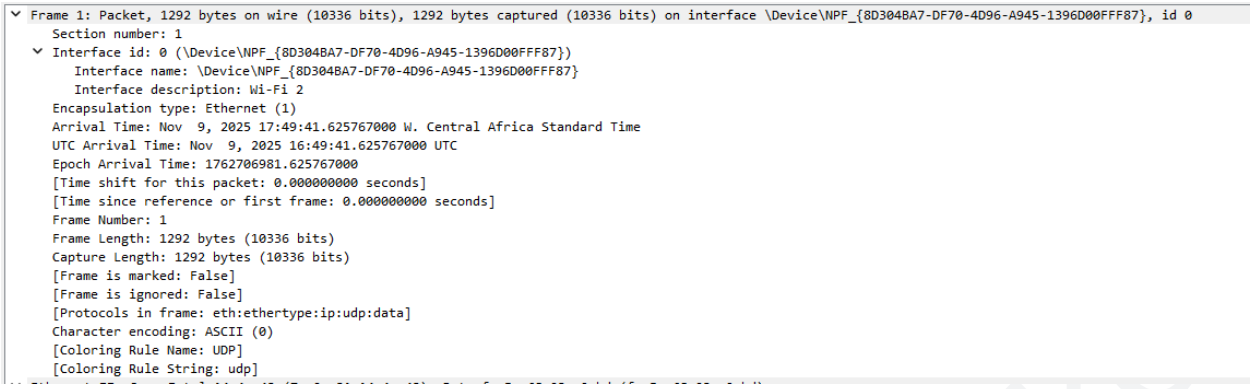
The Packets Details pane



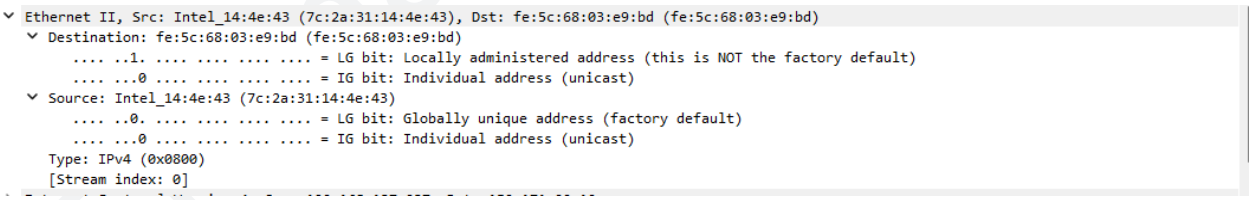
The packet's details pane gives a more granular detail of the packet by **dissecting it layer by layer**, following the OSI model.

- With the **Frame** representing the **physical layer,** giving details about the capture interface, total bits/bytes captured on the wire, and the Frame length

In the snippet below, we can see that the capture interface is **Wi-Fi 2**, and also the arrival time in a different format, and the Frame number is **1**, indicating that it is the first frame captured by Wireshark, and the frame length and the capture length as **1292 bytes** representing the total size of the packet as transmitted on the ***physical medium which is Wi-Fi*** in this case and the actual size of the packet as saved to file on wireshark respectively and then it further provided information about whether the **frame is marked or ignored** which in this case is **false for both**, and the **Protocols in frame section** gives a view of the protocol stack used in the packet indicating the layers of encapsulation showing its movement from ***ethernet to IP to UDP and finally to data***



The **Ethernet II** section presents the **Data Link layer** and deals with MAC addresses, and the major components in this section are the **Source**, which represents the MAC address of the sending endpoint, and the **Destination**, which means the MAC address of the receiving endpoint, the **IG bit**, means the **individual/Group bit** and determines if the address is for a single device (Unicast) when *IG bit* = 0 or to a group (mulitcast) where *IG bit* = 1. The **LG bit** means **Local/Global bit**, which determines how the address was assigned, whether it was manually or dynamically changed by the operating system or by virtualization software when *LG bit* = 1, or it is the factory-burned-in MAC address when *LG bit* = 0. The **Type field with parameter IPV4(0x0800)** tells the receiving device the protocol encapsulated inside the Ethernet frame and the next protocol layer to handle the packets. The **Stream index** is used for tracking related packets beginning with a single connection. The snippet below is a capture of Ethernet II



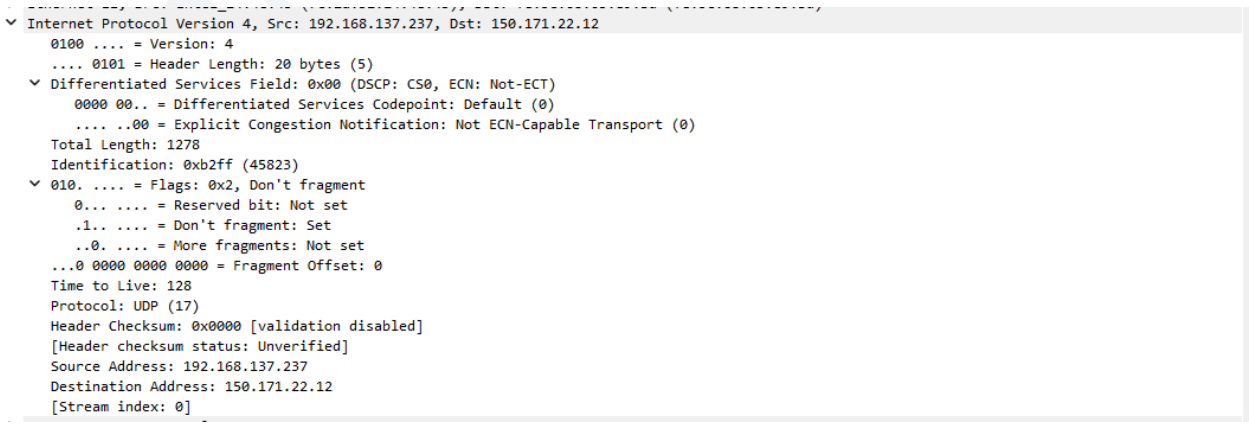
The **Internet Protocol Version 4** section represents the **Network layer** that handles logical addressing. In the snippet below, the **version: 4** confirms that it is an *IPv4* protocol, then the **Header length** tells us about the length of the IPv4 header, which is **20 bytes**.

Differentiated Services Field is used in classifying and managing the network traffic,allowing certain types of data to be priortized over others, **The Differentiated services Codepoint: Default (0)** indicates

that this current packet has default priority which is best effort, T**he Explicit Congestion Notification: Not ECN-Capable Transport(0)** indicates a mechanism for devices used to signal congestion and reduce retransmmission, in this case the packet is not participating in ECN, **Total Length: 1278** identify the size of the entire IP Datagram as **1278 bytes, Identification: 0xb2ff (45823)** help to inform destination endpoint about fragment belonging to the same original packet and the **number (45823)** is unique to fragments of the same transmission, The **Flags: 0x2**, is used to control how packet being fragmented is handled depending on how the *Reserved bit, Don't Fragment, More fragment* is set, in this case the "1.. = Don't fragment: Set." meaning that the packet is not to be fragmented, **The Fragment Offset** normally represent the fragment number, in this case it is 0 meaning that it is not fragmented and no other fragment is expected as this is the entire packet.

The Time to Live: 128 represents a counter that prevents packets from circulating endlessly. As the value decreases by 1 and it reaches 0, the packet is dropped. **Protocol: UDP (17)**, the section tells the receiving machine which protocol is to be handled in layer 4, which corresponds to UDP. In this case, the **Header Checksum** is used to verify if the IP header has not been corrupted during transit, and in this case, the validation was disabled, and the checksum is not Unverified.

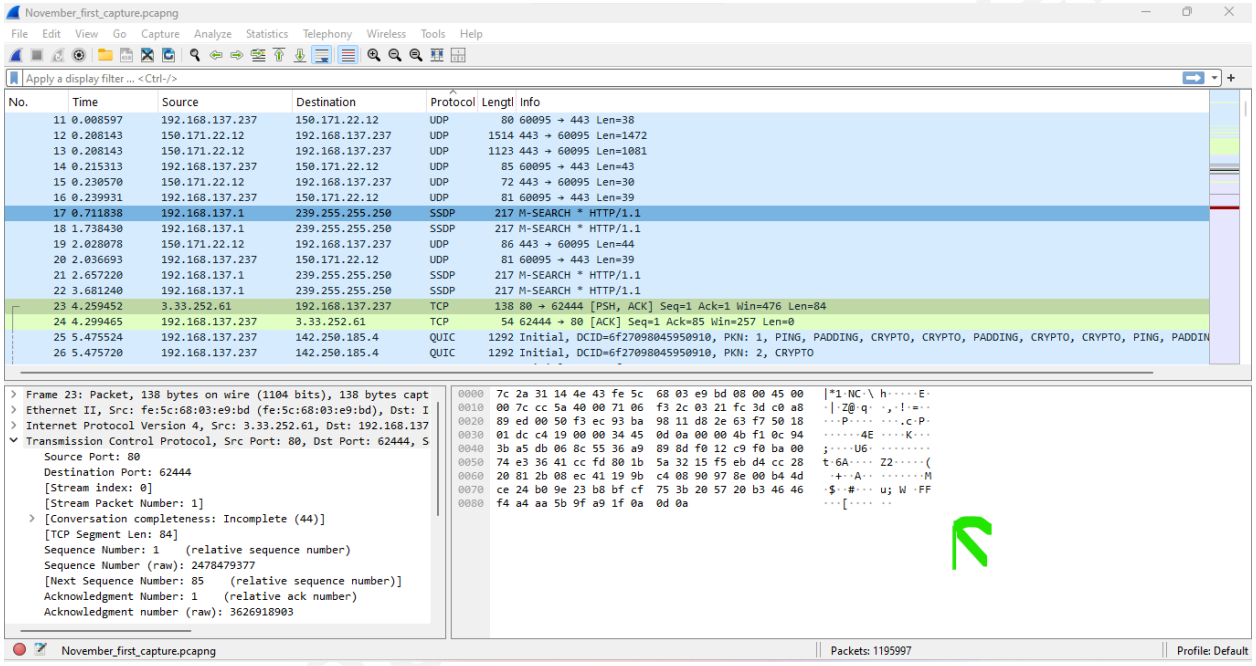
Ensuring that validation is enabled is important for identifying malformed packets. **Source Address and Destination Addresses** represent the IP address for the sending and receiving endpoints, respectively, and the **Stream index** is used for tracking related packets beginning with a single connection, as previously stated



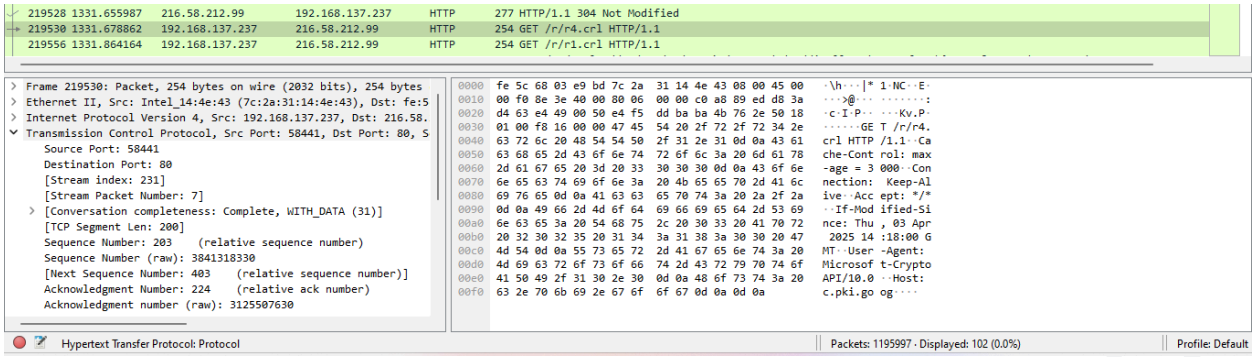
The **Transmission Control Protocol,** which in other cases can be the User Datagram Protocol section, represents the **Transport layer**, and it determines the connection reliability. In a further section of this documentation, there will be a detailed explanation of TCP, UDP, and other protocols analyzed in this project.

The next section, which mostly represents the **Application layer**, is treated differently and will be discussed further in another part of the report

The Packets Bytes pane



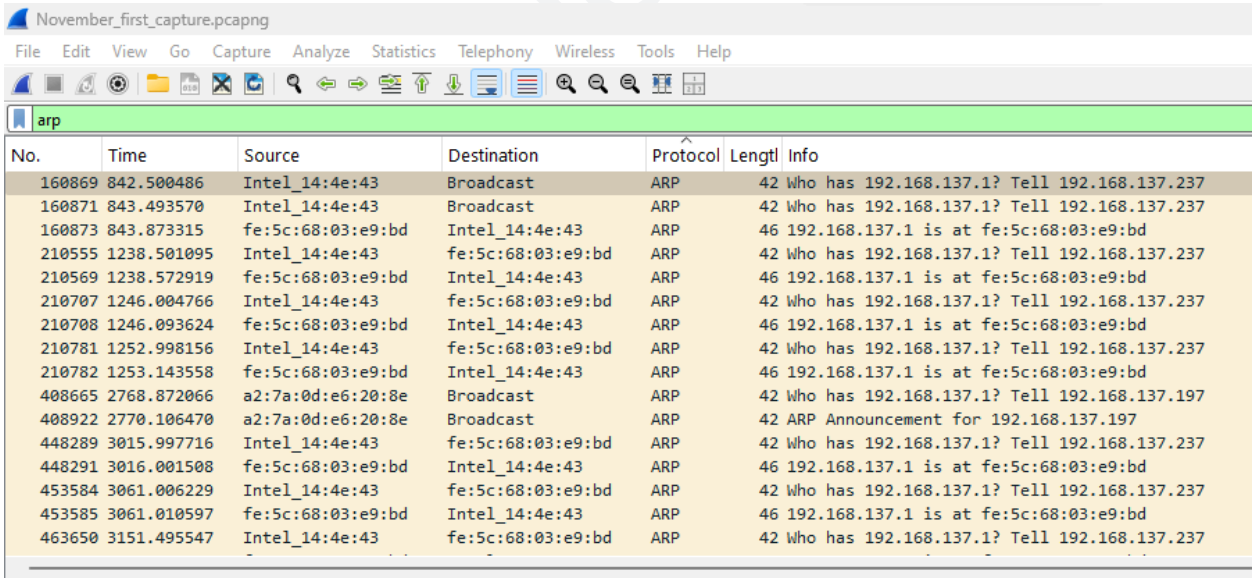
The **Packet Byte pane** shows the exact data as the computer sees it, so the packet byte pane gives the raw data view, On the left side of the byte pane is data formatted in **base 16 (Hexadecimal)** which is what the machine reads and On the right side of the byte pane is data formatted in **ASCII** and it translates the base-16 data on the left to human readable text, the ASCII text may produce human readable text or not depending on the protocol being evaluated, the snippet above is not so readable unlike the one below



Filtering with Wireshark

Most times, capturing network traffic would provide an enormous amount of data, and as a network administrator or security professional scanning through this data is largely a waste of time, so we can utilize various filters in Wireshark to make our search more seamless and effective when carrying out protocol analysis and packet capture, such as:

Filtering with the protocol: One of the easiest ways to filter through the packet capture is to directly filter through the protocol name, such as DNS, ARP, TCP, HTTP, and so on



Filtering with Addresses: We can also filter traffic via addresses, such as ***source and destination MAC addresses, source and destination IP addresses***. The common syntax to filter with addresses would be

eth.addr == 00:00:5e:00:53:00, eth.src == fe:5c:68:03:e9:bd, eth.dst == fe:5c:68:03:e9:bd, ip.addr == 192.168.137.1, ip.src == 192.168.137.1, ip.dst == 192.168.137.1

Common comparison operators to use in Wireshark

Abbreviation	Operator	Meaning
eq	==	Equal
ne	!=	Not Equal
gt	>	Greater Than
lt	<	Less Than
ge	>=	Greater than or Equal to
le	<=	Less than or Equal to

Search and Match operators

Operator	Description
`contains`	Tests whether a protocol, field, or slice contains a specific value
`matches`, `~`	Checks if the string matches the specified Perl-compatible regular expression, ignoring case.

Filtering based on various protocol fields: we can filter based on various properties in the protocol fields, such as ports, responses, and so much more. Common syntax is: *tcp.port == 80, udp.port == 53, http. request.method == "GET", dns. flags.response == 1, arp.opcode == 1 (for ARP request), arp.opcode == 2 (for ARP reply), ip.proto == 6 (for TCP), ip.proto == 17 (for UDP), icmp.type == 8 (for Echo request), icmp.type == 0 (for Echo reply).*

Filtering based on frame properties: We can filter packets based on their inherent characteristics, irrespective of the encapsulated protocols:

- **Filter by size/length:** *frame.len > 1500* (shows packets larger than the standard Ethernet MTU), *frame.len <= 100* (shows very small packets).
- **Filter by frame number:** *frame.number == 1000* (shows the 1000th captured packet), *frame.number > 5000 and frame.number < 5100* (shows a range of packets).
- **Filter by errors:** *expert.severity == error* (identifies packets flagged by Wireshark's expert system as having errors, often useful for troubleshooting malformed or corrupted packets).

Logical Operators

Combining filters using logical operators allows for complex and precise traffic analysis:

- **AND (&&):** Requires both conditions to be true. Example: **ip.addr == 192.168.1.100 && tcp.port == 443** (Traffic to/from 192.168.1.100 on HTTPS port).
- **OR (||):** Requires at least one condition to be true. Example: **dns || arp** (Shows all DNS or ARP packets).

Protocol Analysis with Wireshark

This section of the report takes certain protocols and dives into a deeper explanation of the components of these protocols as defined in Wireshark.

Protocol analysis of UDP

User Datagram Protocol(UDP) is a connectionless-oriented protocol that operates at layer 4 of the OSI model (Transport Layer), and it is responsible for carrying data or packets that require less reliability and are ideal for applications where occasional lost packets are tolerated, such as in DNS queries and video streaming. It has the protocol number of 17

```
User Datagram Protocol, Src Port: 60095, Dst Port: 443
    Source Port: 60095
    Destination Port: 443
    Length: 1258
    Checksum: 0xfc48 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
    [Stream Packet Number: 1]
  [Timestamps]
      [Time since first frame: 0.000000000 seconds]
      [Time since previous frame: 0.000000000 seconds]
    UDP payload (1250 bytes)
Data (1250 bytes)
    Data [...]: 44d9c884372e1875d91d8f364c24da672d39bdba273741dceea00a210d8a34acc98f04a32ed63f1c97a7b5d250c899f1f2db89988101af64870bc45d5ca82ca5534327c3
    [Length: 1250]
```

From the snippet above, the UDP contains the Source port of **60095**, which represents the source port used by the sending endpoint to communicate with the service of the receiving endpoint, which has the destination port of **443**, which is widely recognized as the HTTPS protocol, and the **Length** represents the length of the entire UDP payload. The **Checksum** represents a mathematical value used in detecting errors in the packet's header and data. The checksum is unverified, which means the capture device didn't calculate it; the stream index is used to keep track of traffic within the same stream or thread, and the Timestamps show the relative timing measurements of the packets sent.

The UDP payload contains the actual application data being carried inside the UDP packet,

UDP Packet with SSDP as Data payload

```
forward, hold to see history
> Frame 17: Packet, 217 bytes on wire (1736 bits), 217 bytes captured (1736 bits) on interface \Device\NPF_{8D304BA7-DF70-4D96-A945-1396D00FFF87}, id
> Ethernet II, Src: fe:5c:68:03:e9:bd (fe:5c:68:03:e9:bd), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
> Internet Protocol Version 4, Src: 192.168.137.1, Dst: 239.255.255.250
> User Datagram Protocol, Src Port: 59967, Dst Port: 1900
v Simple Service Discovery Protocol
  > M-SEARCH * HTTP/1.1\r\n
    HOST: 239.255.255.250:1900\r\n
    MAN: "ssdp:discover"\r\n
    MX: 1\r\n
    ST: urn:dial-multiscreen-org:service:dial:1\r\n
    USER-AGENT: Google Chrome/118.0.5993.120 Windows\r\n
    \r\n
    [Full request URI: http://239.255.255.250:1900*]
```

This capture shows a UDP packet with the payload as **SSDP**, which is short for ***Simple Service Discovery Protocol***, which is commonly used by devices on a local network to discover services offered by other devices, it is important to notice that the destination ip is for a

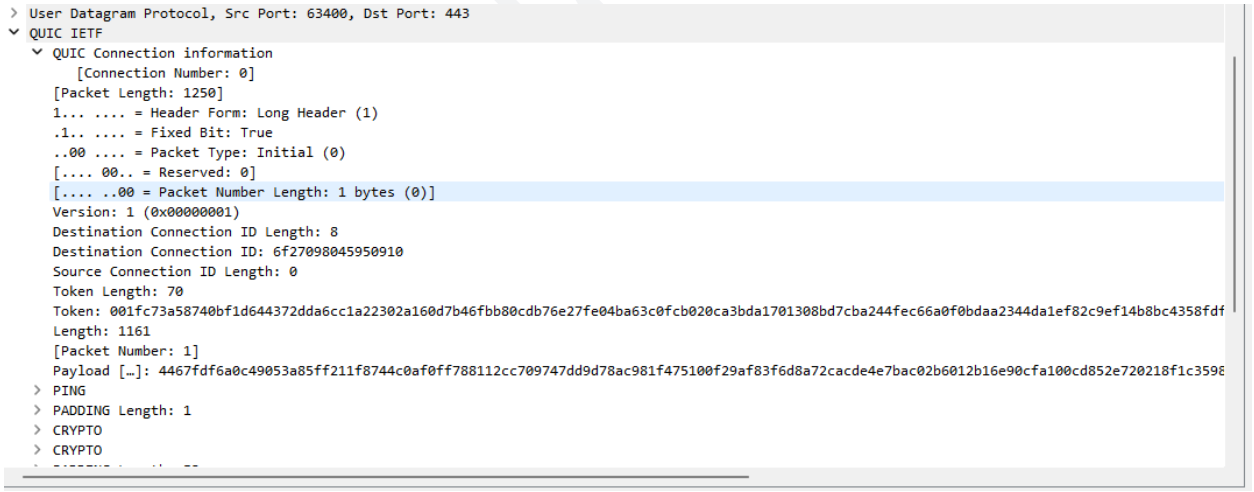
multicast(239.255.255.250) and not designated to a single device, The SSDP data is structured using HTTP-like headers, encapsulated within the UDP packet. **M-SEARCH * HTTP/1.1:** This is the method used. M-SEARCH stands for Method Search and is the request sent by a client to discover available devices or services. **HOST: 239.255.255.250:1900:** Confirms the request is targeted at the standard SSDP multicast address and port.

MAN: "ssdp: discover": The MAN (Mandatory) header specifies the search method, confirming that the client wants to discover services. **MX: 1:** The ***Maximum Wait Time*** in seconds. Devices should wait a random amount of time up to 1 second before responding to prevent all devices from responding simultaneously and flooding the network.

ST: urn:dial-multiscreen-org:service:dial:1: The Service Type (ST) header is the core of the request, asking specifically for a device or service that supports the DIAL (Discovery and Launch) protocol, version 1.

USER-AGENT: Google Chrome/118.0.5993.120 Windows: This identifies the client initiating the search.

UDP packets with QUIC as payload



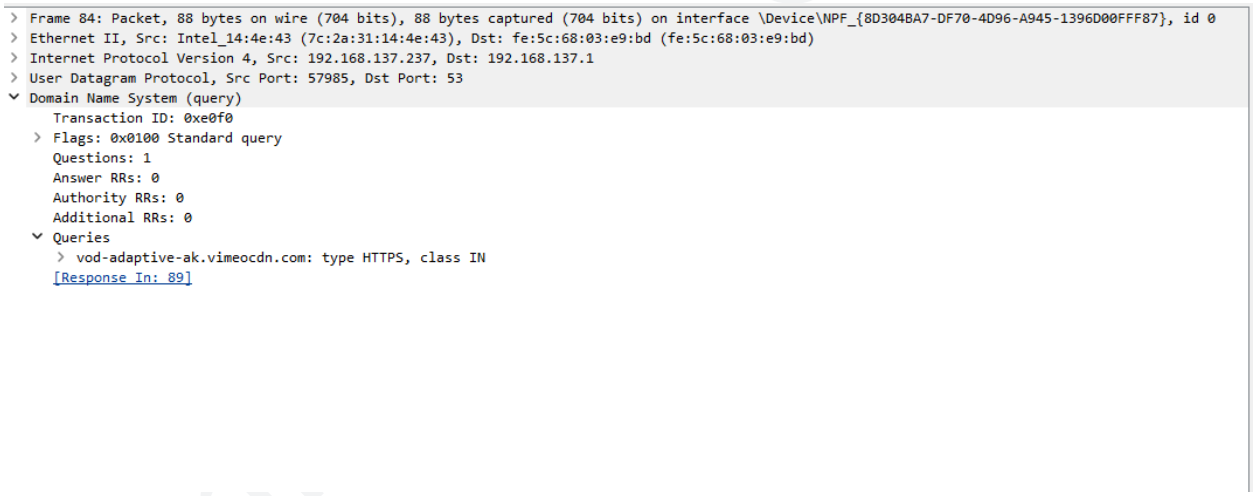
This snippet shows a packet using **QUIC,** which means ***Quick UDP Internet Connections Protocol,*** which is a modern transport layer network protocol developed by Google and it is to replace TCP/TLS for HTTP traffic to make internet connections faster and more secure. The following components forms the QUIC protocol: ***Destination Connection***

ID which is the unique identifier for the connection chosen by the server and this is used to route packets to the connection handler, the ***Token with length 70*** is a cryptographic value used by the server to validate the client's address which forms a defense against DOS attacks, the payload in the QUIC is encrypted and contains components such as:

- **Packet Number:** the sequence number for the QUIC packet
- **PING:** A frame used to confirm the connection is still active and can receive data
- **PADDING LENGTH:** This is the length of the padding frames, which is sometimes used to make a packet meet the minimum size
- **CRYPTO:** This frame carries encrypted handshake data used to negotiate the security keys for the connection

UDP packet with DNS as payload

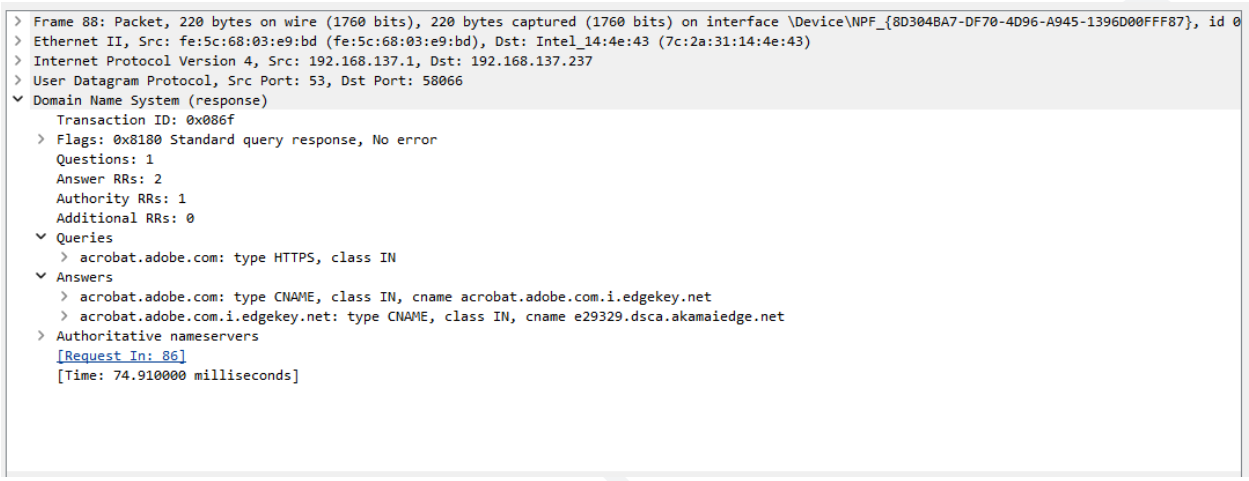
DNS Query



The snippet above describes a ***DNS Query*** to resolve a hostname by asking a local DNS server to find the IP address for the hostname vod-adaptive-ak.vimeocdn.com, The structure of DNS Query request is defined with the **Flags: 0x010** Standard query indicates that the packet is a standard query, **Transaction ID: 0xe0f0** represents a unique identifier assigned to this request, **Questions: 1** shows that the client is asking for the resolution of one single hostname, **Answer RRs: 0 / Authority RRs: 0 / Additional RRs: 0**

These fields are all zero because this is a query, not a response. In a response packet, these fields would indicate the number of resource records (RRs) being sent back. **Queries > vod-adaptive-ak.vimeocdn.com**: This is the hostname (domain name) the client wants resolved to an IP address

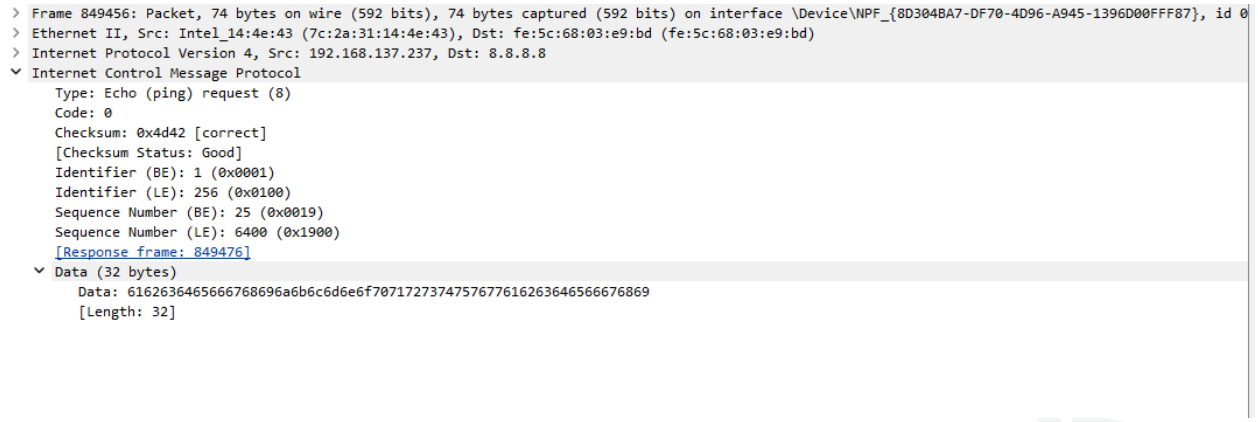
DNS response



The snippet above shows a ***DNS response*** that has a simple structure with the DNS Query.

Protocol Analysis of ICMP

Internet Control Message Protocol (ICMP) is a network layer (Layer 3) protocol used by network devices, such as routers and hosts, to send error messages and operational information, such as whether a requested service is available or if a host or router can be reached. ***ICMP*** is crucial for network troubleshooting and diagnostics, most famously utilized by the **ping** and **traceroute** utilities. ICMP messages are encapsulated within an IP packet. Key components of an ICMP packet include the **Type** and **Code** fields, which together specify the nature of the message (e.g., Echo Request, Echo Reply, Destination Unreachable, Time Exceeded).

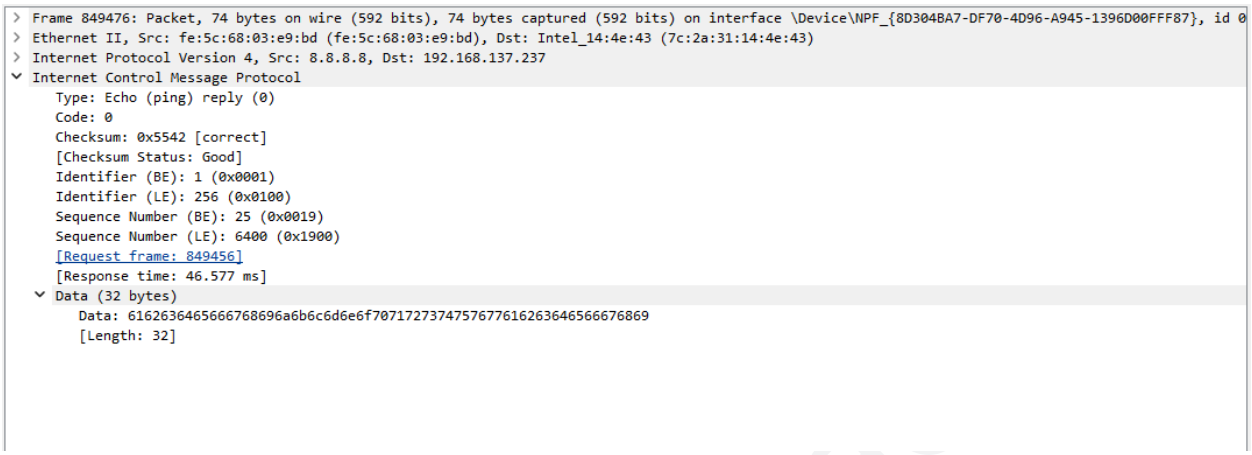


ICMP Echo Request

This snippet displays an **ICMP Echo Request** message, commonly known as a **ping request**, sent from a source host to a destination host to check connectivity.

- **Type: 8 (Echo (ping) request):** This field explicitly identifies the packet as an ICMP Echo Request.
- **Code: 0:** This indicates that the request is a standard Echo Request.
- **Checksum: 0x98f8 [correct]:** This is the value calculated to ensure the integrity of the ICMP header and data during transit. The **[correct]** annotation confirms that the calculated checksum matches the value in the packet.
- **Identifier (BE): / Identifier (LE): :** The Identifier is used by the sending host to match Echo Requests with their corresponding Echo Replies. The (BE) and (LE) values represent the identifier in Big-Endian and Little-Endian byte order, respectively, which is useful for different system architectures.
- **Sequence Number (BE): / Sequence Number (LE)::** The Sequence Number is a counter that increases with each subsequent ICMP message sent. It helps the sending host determine if packets were lost or received out of order.

- **Data (32 bytes):** This field contains the arbitrary data payload sent within the ICMP request



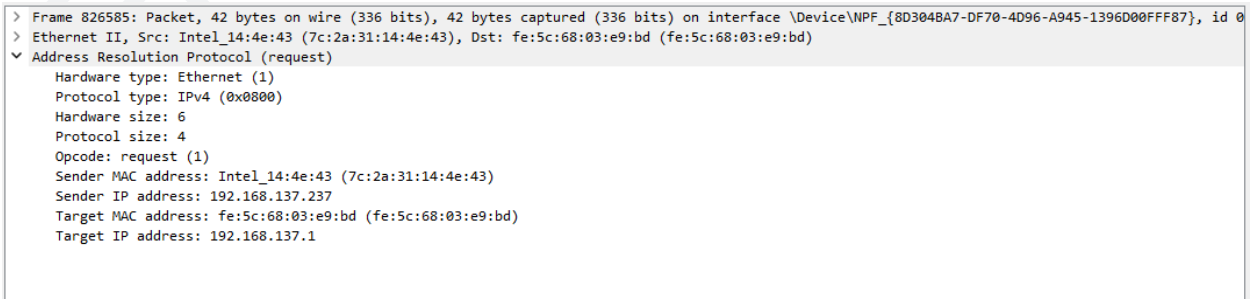
ICMP Echo Reply

This snippet displays an **ICMP Echo Reply** message, and it has a similar structure to ICMP Echo Request, which is the response to an Echo Request, confirming that the destination host is reachable.

Protocol Analysis of ARP

Address Resolution Protocol (ARP) is a critical Layer 2 (Data Link) protocol used to map an Internet Protocol (IP) address to its corresponding physical hardware address, known as the ***Media Access Control (MAC) address***. This translation is essential for devices to communicate with each other within a local network segment (LAN). Since IP addresses are used for routing at Layer 3, and MAC addresses are required for actual frame delivery at Layer 2, ARP acts as the bridge between these two addressing schemes.

ARP Request

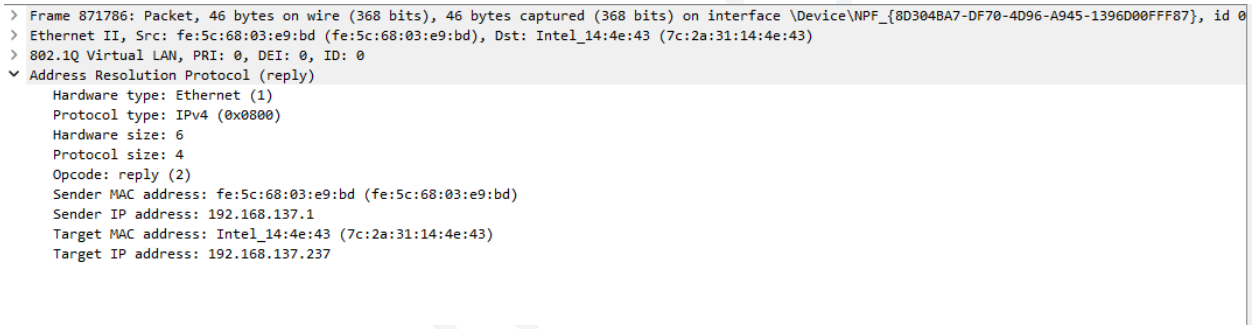


An ARP transaction begins with an **ARP Request**. This request is broadcast across the entire local network segment to all connected devices.

- **Opcode: 1 (Request):** This field identifies the packet as an ARP Request.
- **Sender MAC Address** and **Sender IP Address:** The addresses of the host initiating the request.
- **Target IP Address:** The IP address for which the MAC address is being sought.
- **Target MAC Address:** This field is set to **00:00:00:00:00:00** (or similar null value) because the MAC address is unknown.

When a device receives an ARP Request, it compares the Target IP Address in the packet with its own IP address. Only the device whose IP address matches the target IP will respond.

ARP Reply



The device that recognizes the Target IP Address as its own will send an **ARP Reply** directly back to the requesting host (Unicast).

- **Opcode: 2 (Reply):** This field identifies the packet as an ARP Reply.
- **Sender MAC Address** and **Sender IP Address:** The MAC and IP addresses of the responding host (which were the Target addresses in the Request).
- **Target MAC Address** and **Target IP Address:** The MAC and IP addresses of the original requesting host.

Protocol Analysis of TCP

Transmission Control Protocol (TCP) is a core, connection-oriented protocol operating at Layer 4 (Transport Layer) of the OSI model. Unlike UDP, TCP guarantees reliable, ordered, and error-checked delivery of data between applications. This reliability is achieved through mechanisms like the three-way handshake for connection establishment, sequencing, acknowledgments (ACKs), and flow control, making it ideal for applications that require high integrity, such as web browsing (HTTP/HTTPS), email transfer (SMTP), and file transfer (FTP).

TCP Components

The TCP header contains several vital components necessary for maintaining the connection and ensuring reliable delivery:

- **Source Port** and **Destination Port:** These fields identify the application or service sending and receiving the data, respectively.
- **Sequence Number:** A field representing the sequence number of the first data byte in the segment. It is used to reassemble the data in the correct order.
- **Acknowledgment Number (ACK):** A field containing the next sequence number the sender of the segment is expecting to receive from the other side, confirming receipt of all prior data.
- **Header Length:** Specifies the size of the TCP header.
- **Flags/Control Bits:** These 9 individual bits are crucial for connection management and flow control:
    - **URG (Urgent):** Indicates the Urgent Pointer field is significant.

    - **ACK (Acknowledgment):** Indicates the Acknowledgment Number field is significant (always set after the initial handshake SYN).
    - **PSH (Push):** Forces the sending application to deliver data to the receiving application immediately.
    - **RST (Reset):** Terminates a connection abruptly due to an error.
    - **SYN (Synchronize):** Used to initiate a connection (first step of the three-way handshake).
    - **FIN (Finish):** Used to gracefully terminate a connection.
- **Window Size:** A field used for flow control, indicating the amount of receive buffer space (in bytes) available, informing the sender how much data it can transmit before receiving an acknowledgment.
- **Checksum:** A field used for error-checking the TCP header and data payload.
- **Urgent Pointer:** Points to the byte offset in the sequence number where urgent data ends.
- **Options:** Allows for optional features, such as Maximum Segment Size (MSS) negotiation.

PACKET CAPTURE WITH TCPDUMP

This is a Command Line Interface (CLI) Network packet capture and protocol analyzer that is capable of capturing live traffic and widely used for network troubleshooting, security monitoring, and is particularly suited to Unix-like operating Systems, it is similar to wireshark in its functionality in capturing packets

Common Commands used in Tcpdump and their uses

Sudo tcpdump -D : this command would list out all interface on which tcpdump can capture traffic on

```
zsh: corrupt history file /home/kali/.zsh_history
┌──(kali㉿kali)-[~]
└─$ sudo tcpdump -D
[sudo] password for kali:
1.eth0 [Up, Running, Connected]
2.any (Pseudo-device that captures on all interfaces) [Up, Running]
3.lo [Up, Running, Loopback]
4.bluetooth-monitor (Bluetooth Linux Monitor) [Wireless]
5.nflog (Linux netfilter log (NFLOG) interface) [none]
6.nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
7.dbus-system (D-Bus system bus) [none]
8.dbus-session (D-Bus session bus) [none]

┌──(kali㉿kali)-[~]
└─$
```

Sudo tcpdump -i eth0: This command tells tcpdump to capture traffic on the eth0 interface

```
┌──(kali㉿kali)-[~]
└─$ sudo tcpdump -i eth0
[sudo] password for kali:
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
```

Sudo tcpdump -n -i eth0: this command tells tcpdump to capture traffic on the eth0 interface and ensures that name resolution is disabled

```
┌──(kali㉿kali)-[~]
└─$ sudo tcpdump -n -i eth0
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
04:51:37.509622 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 53, length 64
04:51:37.515851 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 53, length 64
04:51:38.511471 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 54, length 64
04:51:38.521995 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 54, length 64
04:51:39.514864 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 55, length 64
04:51:39.521535 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 55, length 64
04:51:40.517851 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 56, length 64
04:51:40.525573 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 56, length 64
04:51:41.674043 ARP, Request who-has 10.0.2.2 tell 10.0.2.15, length 28
04:51:41.677240 ARP, Reply 10.0.2.2 is-at 52:55:0a:00:02:02, length 50
04:51:41.678482 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 57, length 64
04:51:41.845596 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 57, length 64
04:51:43.499540 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 58, length 64
04:51:43.506009 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 58, length 64
^X04:51:44.501927 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 59, length 64
04:51:44.518827 IP 8.8.8.8 > 10.0.2.15: ICMP echo reply, id 1, seq 59, length 64
04:51:46.313232 IP 10.0.2.15 > 8.8.8.8: ICMP echo request, id 1, seq 60, length 64
^C
17 packets captured
18 packets received by filter
0 packets dropped by kernel
```

Sudo tcpdump -c10 -i eth0: this command tells tcpdump to capture

traffic on the eth0 interface and ensures to only capture 10 packets.

```
┌──(kali㉿kali)-[~]
└─$ sudo tcpdump -c10 -i eth0
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
04:55:00.772812 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 223, length 64
04:55:00.780063 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 223, length 64
04:55:01.774892 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 224, length 64
04:55:01.781622 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 224, length 64
04:55:02.775966 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 225, length 64
04:55:02.782597 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 225, length 64
04:55:03.777733 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 226, length 64
04:55:03.783680 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 226, length 64
04:55:04.784069 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 227, length 64
04:55:04.791720 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 227, length 64
10 packets captured
42 packets received by filter
26 packets dropped by kernel
```

Sudo tcpdump -w file.pcap: This command tells tcpdump to store the captured packets in file.pcap

```
┌──(kali㉿kali)-[~]
└─$ sudo tcpdump -w file.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C49 packets captured
51 packets received by filter
0 packets dropped by kernel

┌──(kali㉿kali)-[~]
└─$ ls
Desktop    Downloads  Karatu            Karatu-PKI  packages.microsoft.gpg  Project_Nov_capture2.pcap  Public     testFolder   Videos      wget-log    wget-log.2
Documents  file.pcap  Karatu-Assymetric  Music       Pictures                Project_Nov_capture.pcap   Templates  testFolder2  vscode.deb  wget-log.1  wget-log.3
```

Sudo tcpdump -r file.pcap: This command tells tcpdump to read into the file.pcap file and displays it on the screen

```
┌──(kali㉿kali)-[~]
└─$ sudo tcpdump -r file.pcap
reading from file file.pcap, link-type EN10MB (Ethernet), snapshot length 262144
04:57:35.428613 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 377, length 64
04:57:35.435417 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 377, length 64
04:57:36.430272 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 378, length 64
04:57:36.438776 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 378, length 64
04:57:37.431787 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 379, length 64
04:57:37.446639 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 379, length 64
04:57:38.433612 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 380, length 64
04:57:38.440321 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 380, length 64
04:57:39.434684 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 381, length 64
04:57:39.446933 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 381, length 64
04:57:40.436501 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 382, length 64
04:57:40.445422 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 382, length 64
04:57:41.437912 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 383, length 64
04:57:41.444296 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 383, length 64
04:57:42.438577 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 384, length 64
04:57:42.449598 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 384, length 64
04:57:43.440001 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 385, length 64
04:57:43.447076 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 385, length 64
04:57:44.441972 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 386, length 64
04:57:44.448907 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 386, length 64
04:57:45.444009 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 387, length 64
04:57:45.450635 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 387, length 64
04:57:46.445662 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 388, length 64
04:57:46.480326 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 388, length 64
04:57:47.447802 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 389, length 64
04:57:47.454477 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 389, length 64
04:57:48.449936 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 390, length 64
04:57:48.457854 IP dns.google > 10.0.2.15: ICMP echo reply, id 1, seq 390, length 64
04:57:49.452012 IP 10.0.2.15 > dns.google: ICMP echo request, id 1, seq 391, length 64
```

There are so many tcpdump command options to explore when dealing with tcpdump, and you get a cheatsheet at https://www.comparitech.com/net-admin/tcpdump-cheat-sheet/

Summary: This project report provided a detailed analysis of network traffic capture and protocol analysis using two essential tools: Wireshark and Tcpdump. The primary objective was to demonstrate the practical application of these tools in identifying risks, vulnerabilities, and threats within network traffic.

The report first established the context, detailing the cross-platform GUI-based capabilities of **Wireshark** for in-depth protocol analysis and filtering, alongside the CLI-based efficiency of **Tcpdump** for live traffic capture, particularly suited for Unix-like operating systems.

The core of the analysis focused on dissecting a captured PCAP file using Wireshark. The report meticulously detailed the Wireshark interface, including the Packet List Pane, Packet Details Pane (explaining the OSI layer dissection), and Packet Bytes Pane (showing

Hexadecimal and ASCII data). Crucial filtering techniques were explored, utilizing protocols, addresses, comparison operators, and logical operators (AND/OR) to streamline the analysis of large datasets.

In the **Protocol Analysis** section, key transport and network layer protocols were examined:

- **UDP:** Its connectionless nature was highlighted, alongside analysis of payloads such as **SSDP** (multicast service discovery) and **QUIC** (modern, fast, secure internet connections).
- **DNS:** The structure and distinction between DNS Queries and Responses were outlined.
- **ICMP:** Its role in network diagnostics (ping/traceroute) was covered, with a breakdown of Echo Request and Echo Reply messages.
- **ARP:** The Layer 2 process of mapping IP to MAC addresses via ARP Request (broadcast) and ARP Reply (unicast) was explained.
- **TCP:** As a reliable, connection-oriented protocol, its essential components, like Sequence Numbers, Acknowledgment Numbers, Flags (SYN, ACK, FIN, RST), and Window Size, were documented.

Finally, the report provided a cheatsheet of common **Tcpdump** commands for interface selection, packet count limits, name resolution disabling, and file input/output operations, confirming its utility for command-line-based network security monitoring and troubleshooting.

Conclusion: The successful execution of this project has demonstrated a solid foundational understanding of network traffic analysis using industry-standard tools. By performing deep dives into the packet structures of various protocols (UDP, ICMP, ARP, TCP, DNS, QUIC, SSDP), the report successfully illustrated how specific protocol fields and headers inform network behavior, connectivity, and potential security issues.

The ability to effectively filter traffic in Wireshark and manage captures using Tcpdump commands is paramount for network administrators and security professionals. These skills are directly applicable to:

1. **Troubleshooting:** Quickly isolating connectivity issues (e.g., failed TCP handshakes, ICMP errors) or application latency (e.g., analyzing DNS response times).
2. **Security Analysis:** Identifying anomalous traffic patterns, detecting unauthorized service discovery (SSDP), spotting malformed packets (indicated by Wireshark errors), or recognizing suspicious connection attempts (analyzing TCP flags).
3. **Vulnerability Identification:** Pinpointing unencrypted communication (e.g., clear-text HTTP, older protocols) or observing behaviors that could indicate a Man-in-the-Middle or reconnaissance attack (excessive ARP or DNS queries).

In conclusion, Wireshark and Tcpdump are indispensable assets in the cybersecurity and network engineering toolkit. Mastery of these tools is crucial for ensuring network integrity, optimizing performance, and proactively defending against a constantly evolving threat landscape.