

**LAB RECORD**

**COMPUTER NETWORKS LAB**

**20CYS382**

**Name: CH S N Kapil Anirudh**

**Roll No: CH.EN.U4CYS22009**

**Amrita Vishwa Vidyapeetham Chennai – 601 103, Tamil Nadu, India.**

**October 2024**

1. **Basic Networking Commands:**

# ***TRACERT:-***

This command is used to diagnose path-related problems. On an IP network, routers exchange IP packets between the source and the destination. They take IP packets from the source host and forward them in a sequence until they reach the destination host. The sequence of routers between the source and destination is known as the path. A path consists of all routers in a sequence that IP packets sent from the source host traverse to reach the destination host.

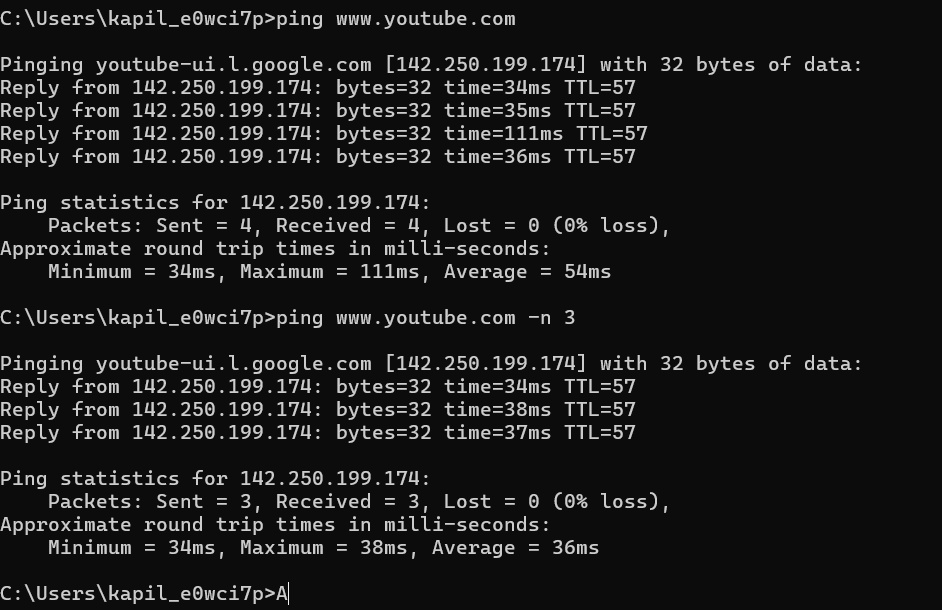
The **tracert** command prints the path. If all routers on the path are functional, this command prints the full path. If a router is down on the path, this command prints the path up to the last operational router.





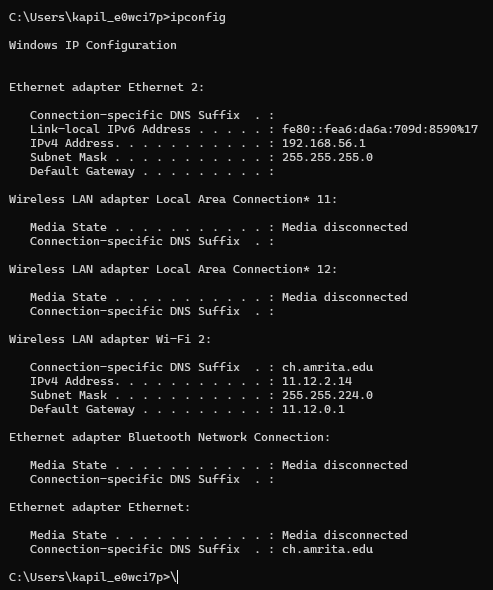
# ***PING:-***

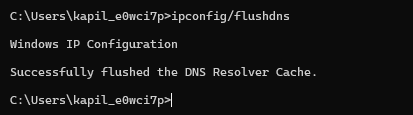
The **ping** command is used to test connectivity between two hosts. It sends ICMP echo request messages to the destination. The destination host replies with ICMP reply messages. If the ping command gets a reply from the destination host, it displays the reply along with round-trip times.



# ***IP CONFIGURATION: -***

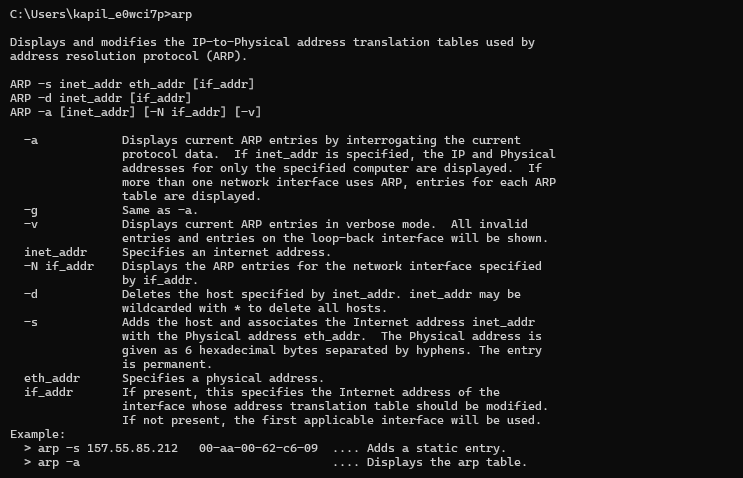
This command displays all current TCP/IP network configuration values and refreshes Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) settings. This command is mainly used to view the IP addresses on the computers that are configured to obtain their IP address automatically.



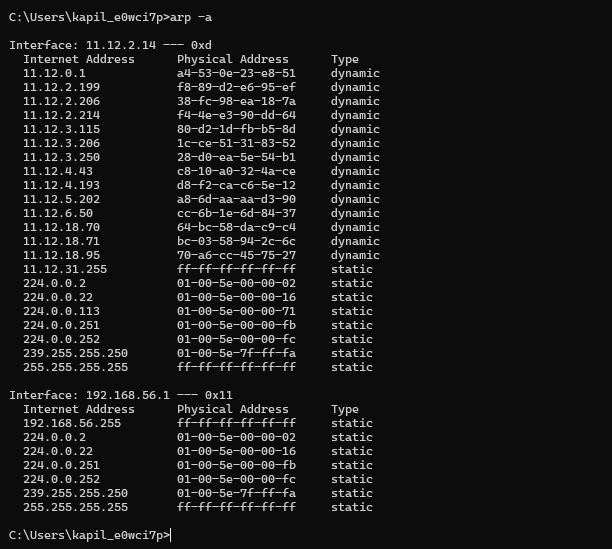


# ***ARP:-***

The ARP protocol broadcasts a given IP address over a local network. The corresponding host responds to the broadcast with its MAC address. To avoid repetition, ARP stores the answer in a table known as **ARP table**. ARP maintains a separate ARP table for each NIC.



Arp -a:

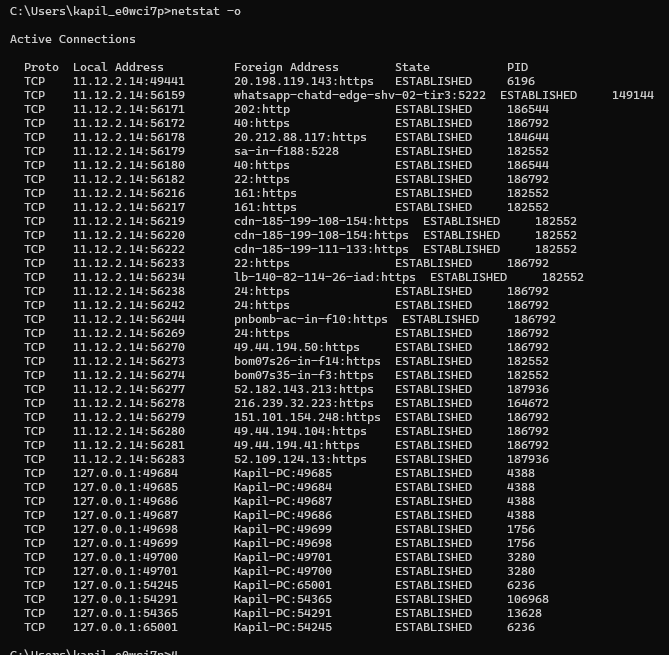


# ***NETSTAT:-***

The netstat command is like a special tool in Linux that helps you understand and check things about how your computer connects to the internet. It can tell you about the connections your computer is making, the paths it uses to send information, and even some technical details like how many packets of data are being sent or received. In simple terms, it’s like a window that shows you what’s happening with your computer and the internet. This article will help you learn how to use netstat, exploring different ways to get specific information.



**Netstat -o:**



**2) Client Server Programming using TCP Protocol:**

Server code:

import java.io.\*;

import java.net.\*;

public class TCPServer {

    public static void main(String[] args) {

        ServerSocket serverSocket = null;

        Socket clientSocket = null;

        try {

            serverSocket = new ServerSocket(6789);

            System.out.println("Server is listening on port 6789");

            clientSocket = serverSocket.accept();

            System.out.println("Client connected");

            BufferedReader inFromClient = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

            DataOutputStream outToClient = new DataOutputStream(clientSocket.getOutputStream());

            String clientMessage = inFromClient.readLine();

            System.out.println("Received from client: " + clientMessage);

            String response = "Hello from the server";

            outToClient.writeBytes(response + "\n");

        } catch (IOException e) {

            e.printStackTrace();

        } finally {

            try {

                if (clientSocket != null) clientSocket.close();

                if (serverSocket != null) serverSocket.close();

            } catch (IOException e) {

                e.printStackTrace();

            }

        }

    }

}

**Client code:**

import java.io.\*;

import java.net.\*;

public class TCPClient {

    public static void main(String[] args) {

        String serverName = "localhost";

        int port = 6789;

        try {

            Socket clientSocket = new Socket(serverName, port);

            DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());

            BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

            String message = "Hello from the client";

            outToServer.writeBytes(message + "\n");

            String response = inFromServer.readLine();

            System.out.println("Received from server: " + response);

            clientSocket.close();

        } catch (IOException e) {

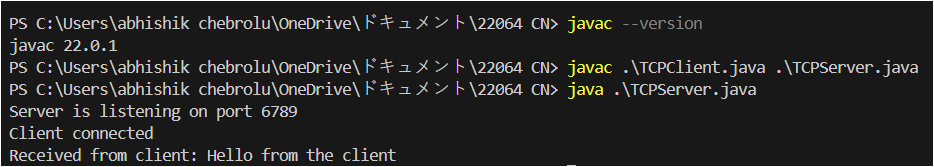
            e.printStackTrace();

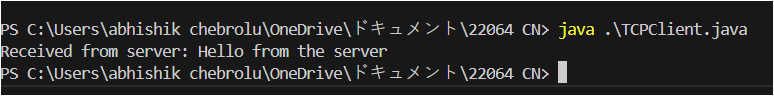
        }

    }

}

**Output:**





1. **Client Server Programming using UDP Protocol:**

**Server code:**

import java.net.\*;

public class UDPServer {

    public static void main(String[] args) {

        DatagramSocket socket = null;

        try {

            socket = new DatagramSocket(9876);

            byte[] receiveData = new byte[1024];

            while (true) {

                DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

                socket.receive(receivePacket);

                String message = new String(receivePacket.getData(), 0, receivePacket.getLength());

                InetAddress clientAddress = receivePacket.getAddress();

                int clientPort = receivePacket.getPort();

                System.out.println("Received message from " + clientAddress + ":" + clientPort + " - " + message);

                String responseMessage = "Message received!";

                byte[] sendData = responseMessage.getBytes();

                DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, clientAddress, clientPort);

                socket.send(sendPacket);

            }

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

            if (socket != null) {

                socket.close();

            }

        }

    }

}

**Client code:**

import java.net.\*;

public class UDPClient {

    public static void main(String[] args) {

        DatagramSocket socket = null;

        try {

            socket = new DatagramSocket();

            InetAddress serverAddress = InetAddress.getByName("localhost");

            String message = "Hello from UDP client";

            byte[] sendData = message.getBytes();

            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, 9876);

            socket.send(sendPacket);

            byte[] receiveData = new byte[1024];

            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

            socket.receive(receivePacket);

            String responseMessage = new String(receivePacket.getData(), 0, receivePacket.getLength());

            System.out.println("Response from server: " + responseMessage);

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

            if (socket != null) {

                socket.close();

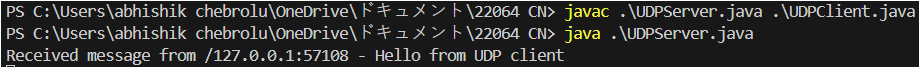
            }

        }

    }

}

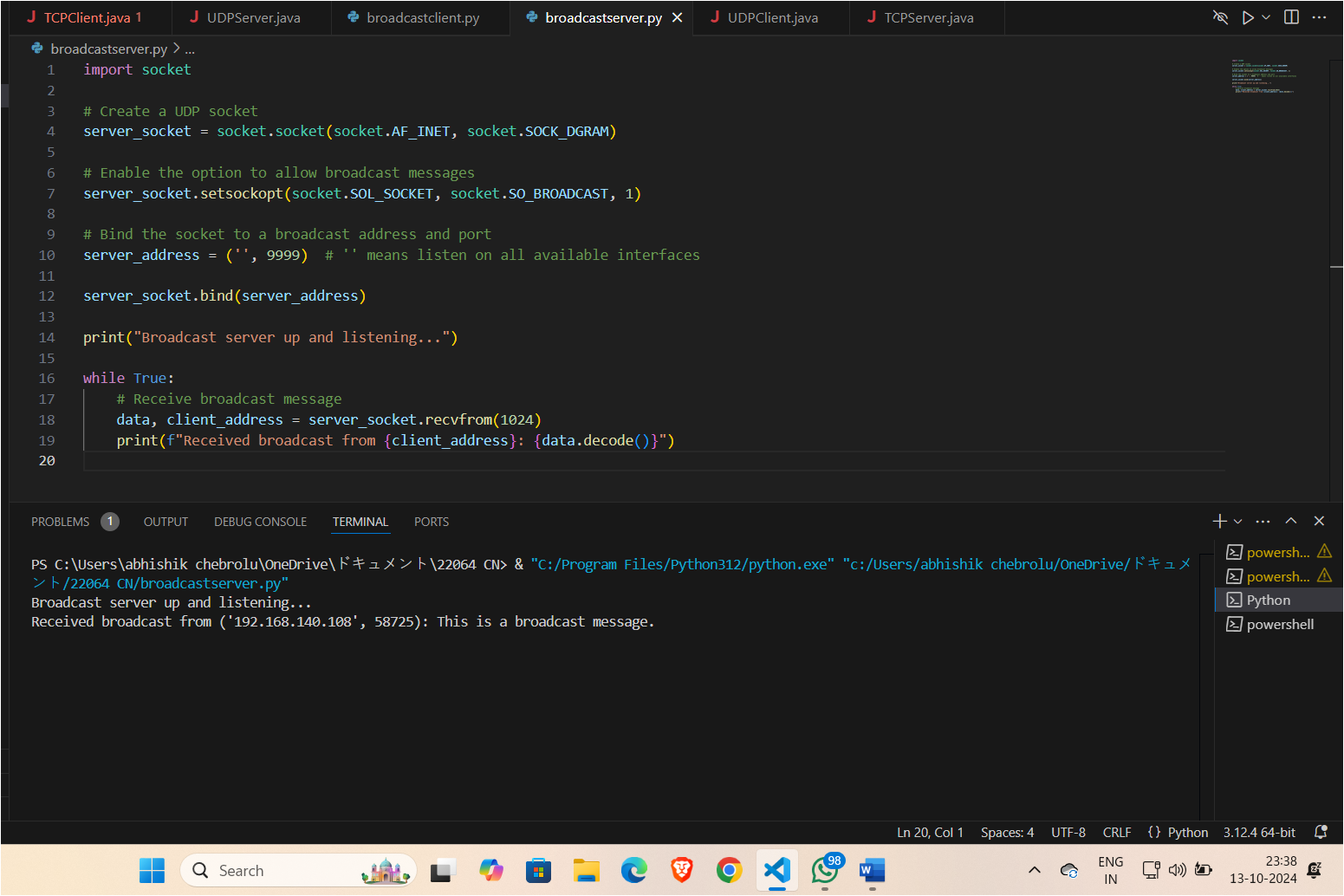
**Output:**

****

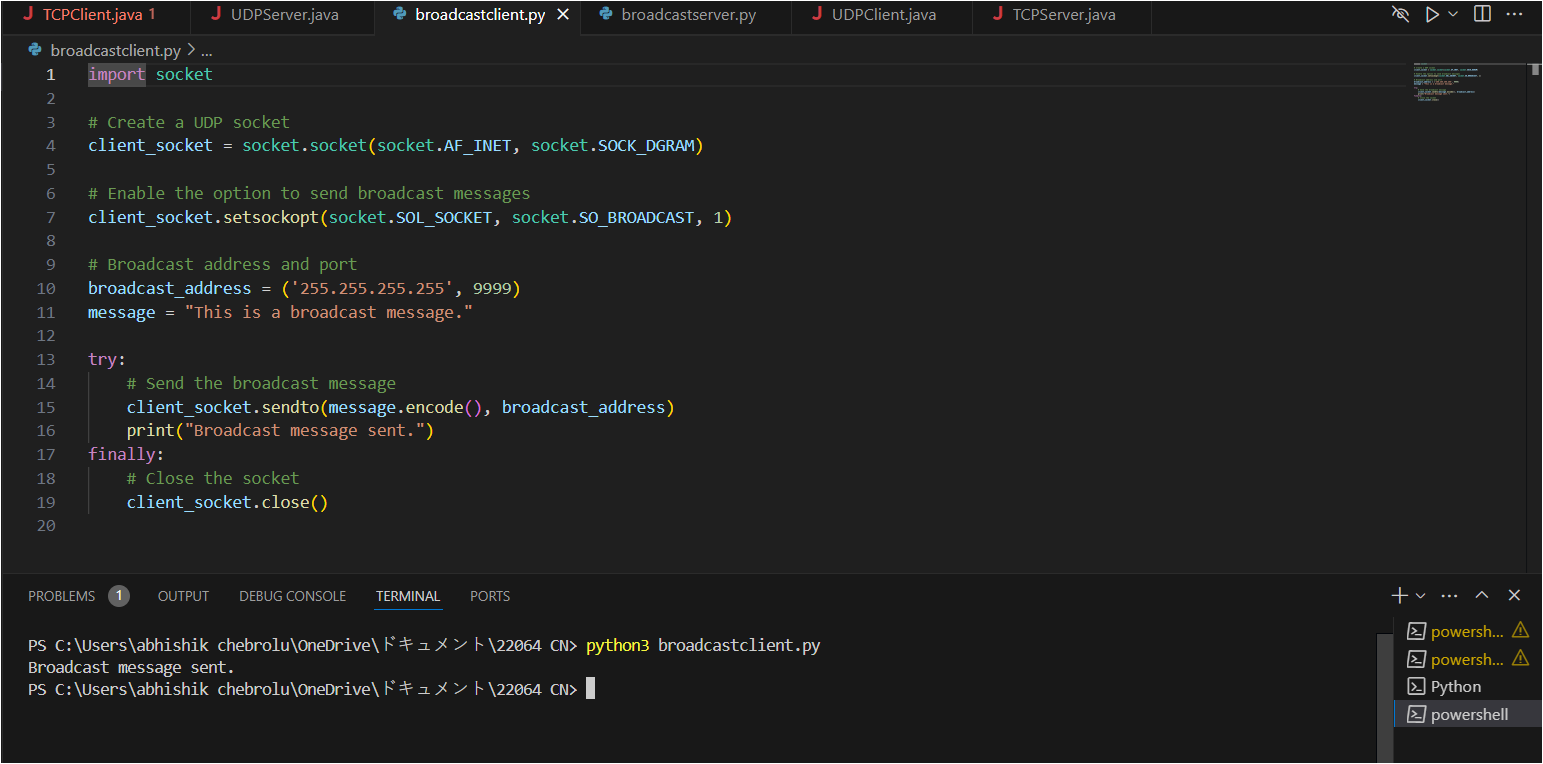
****

1. **Broadcast:**

**Server code and output:**

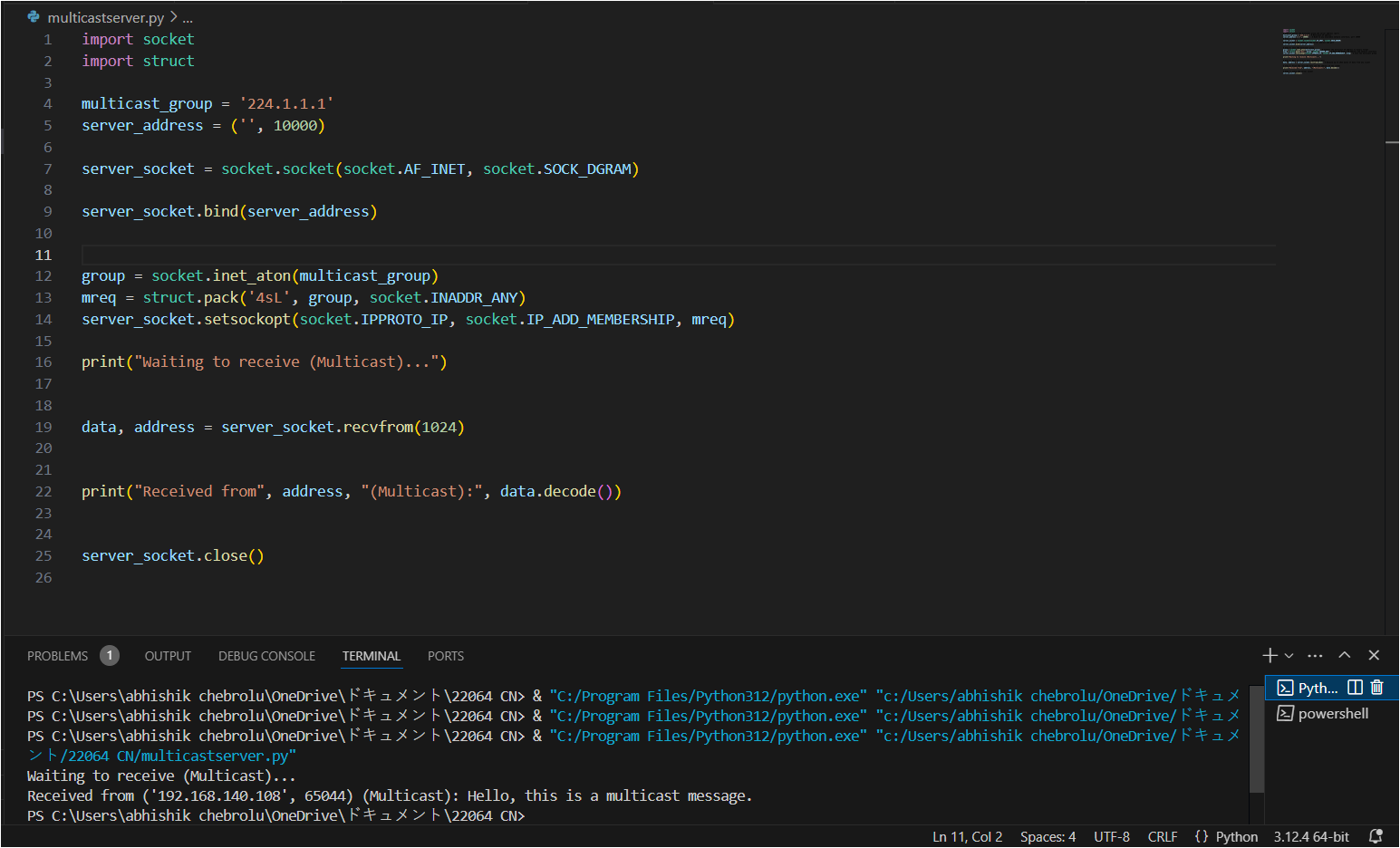
****

**Client code and output:**

****

**Multicast:**

**Server code and output:**

****

**Client code and output:**

****

1. **Implementation of Checksum:**

**Code and Output:**

****

1. **Implementation and Simulation of algorithms for routing protocols, finding the shortest path in the given network using Dijkstra and Link state:**

**Code:**

import networkx as nx

G = nx.Graph()

edges = [("A", "B", 5), ("A", "D", 8), ("B", "D", 7), ("B", "C", 2), ("C", "D", 6)]

G.add\_weighted\_edges\_from(edges)

def link\_state\_routing(graph):

    for node in graph.nodes:

        lengths = nx.single\_source\_dijkstra\_path\_length(graph, node)

        paths = nx.single\_source\_dijkstra\_path(graph, node)

        print(f"Router {node}'s shortest paths:")

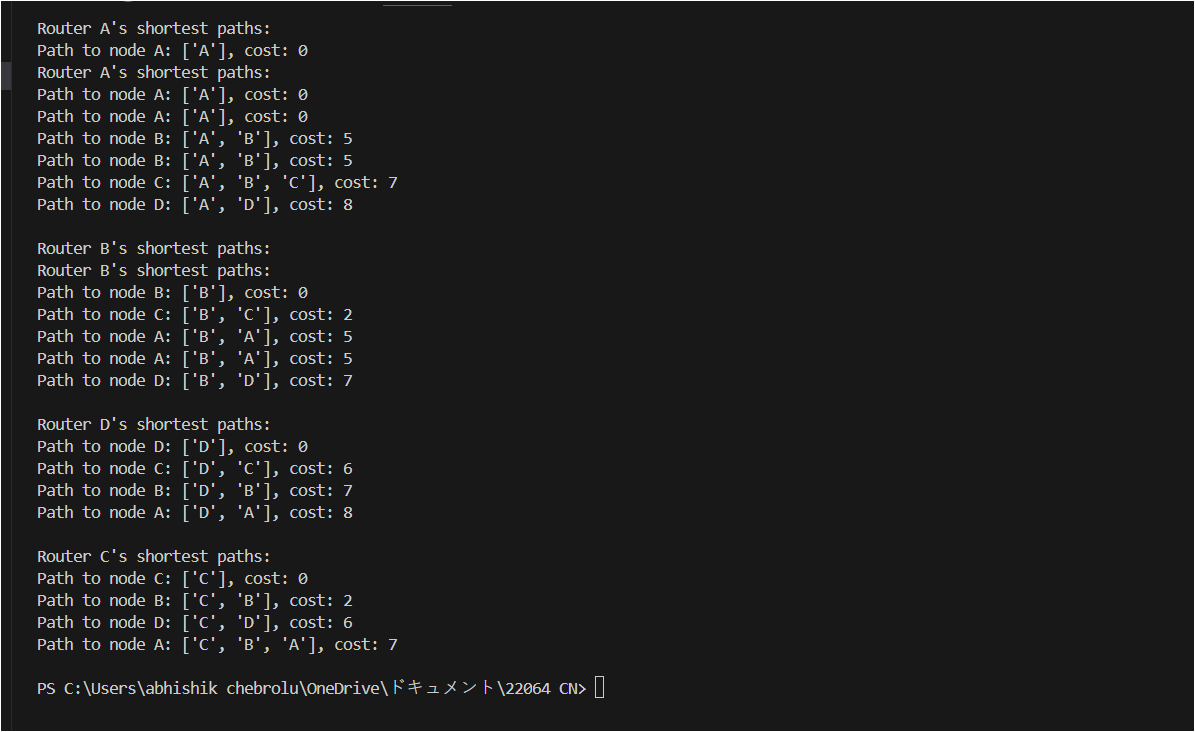
        for dest, length in lengths.items():

            print(f"Path to node {dest}: {paths[dest]}, cost: {length}")

        print()

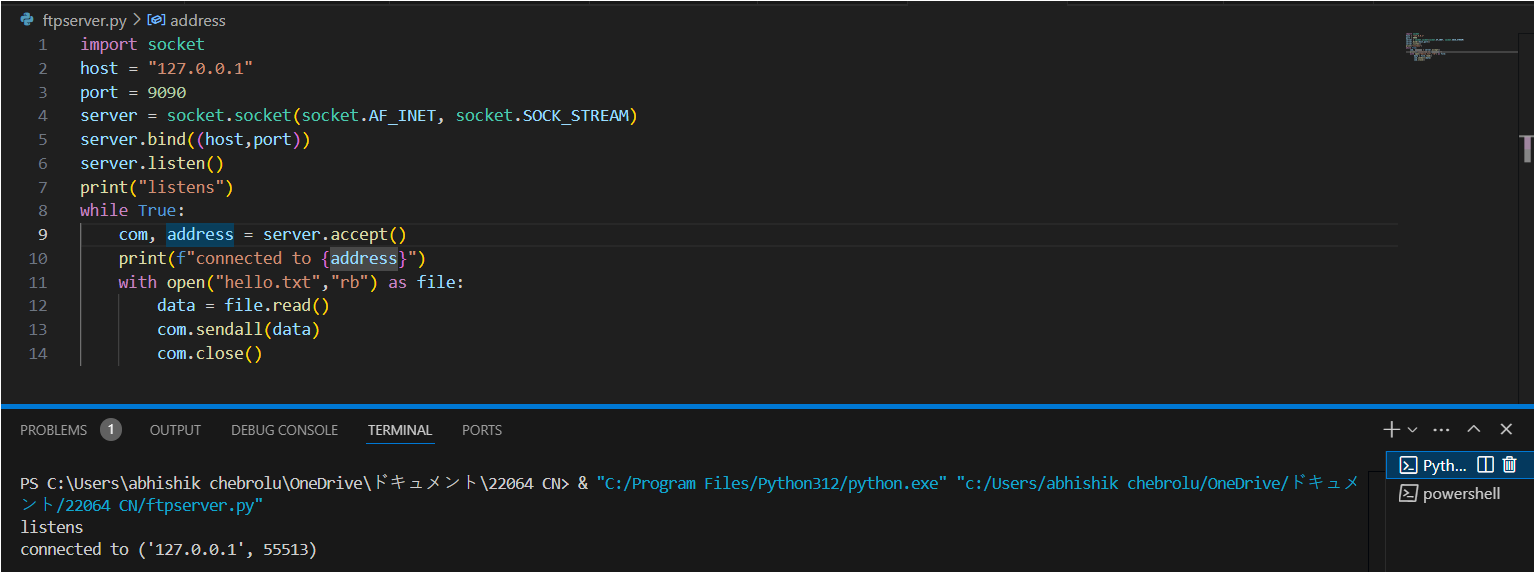
link\_state\_routing(G)

**Output:**

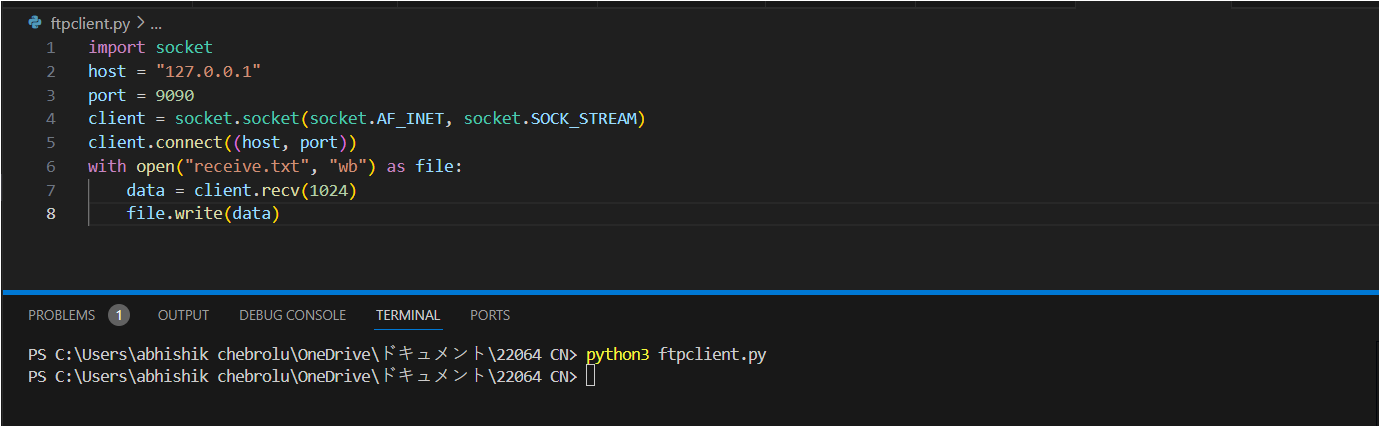
****

1. **FTP:**

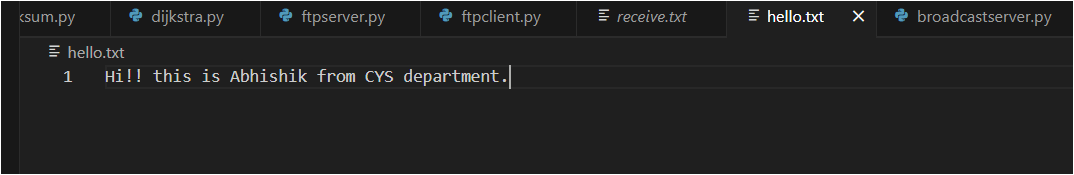
**Server code and output:**

****

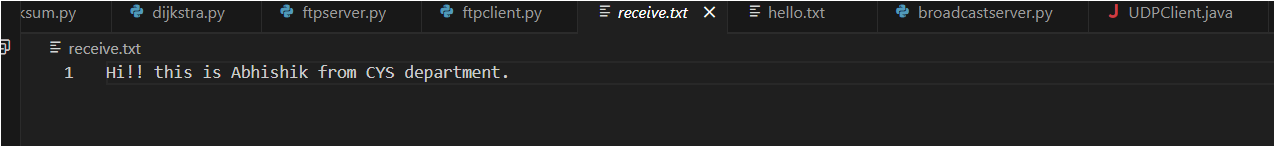
**Client code and output:**

****

**Hello.txt:**

****

**Receive.txt:**

****

1. **IP Spoofing:**

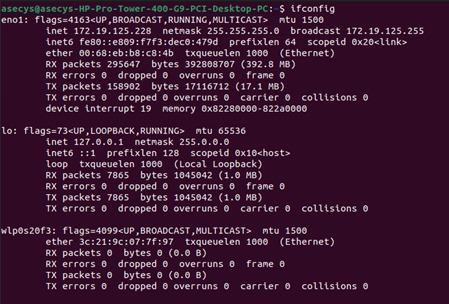
IP Spoofing using nmap

Aim : To spoof the IP address of using nmap

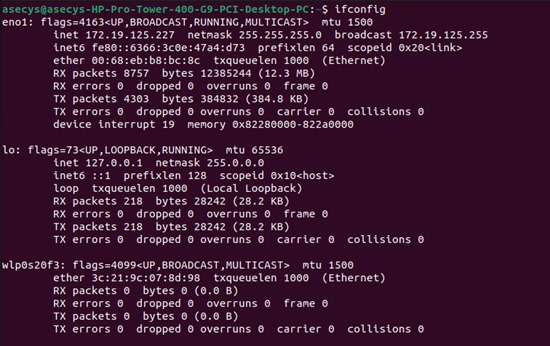
Procedure:

1. First, we need to have two machines for this experiment. We first find out the IP addresses of both the machines.

MACHINE 1:



MACHINE 2:

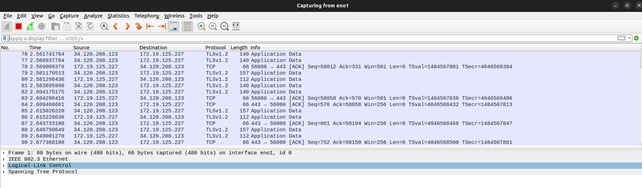


2. In the second machine we open a port using netcat. We use the command

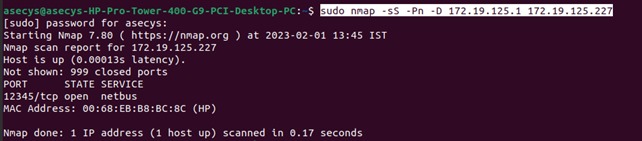
nc lnvp [port-number]



3. We open wireshark and start listening to the traffic coming through.



4. Now we use the nmap to spoof the IP using the below command.



sudo nmap -sS -Pn -D 172.19.125.1 172.19.125.227

where,

-sS – Stealth Scan

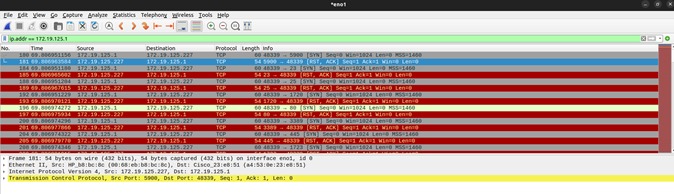
-Pn – skip host discovery

-D – decoy scan

We can also mention use a random IP address instead of mentioning it by using the command -RND.

5. Finally, we check whether the spoofing is done successfully by checking the incoming traffic in wireshark. We can filter the spoofed ip using the filter.

ip.addr == <spoofed IP address>



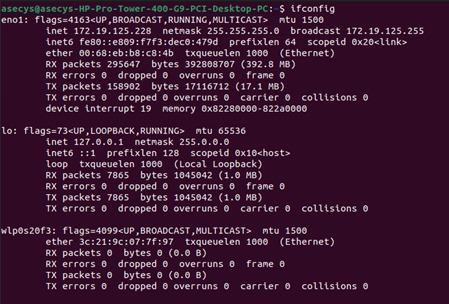
Experiment #6 : DOS attack using hping3

Aim: To perform DOS attack on another machine using hping3

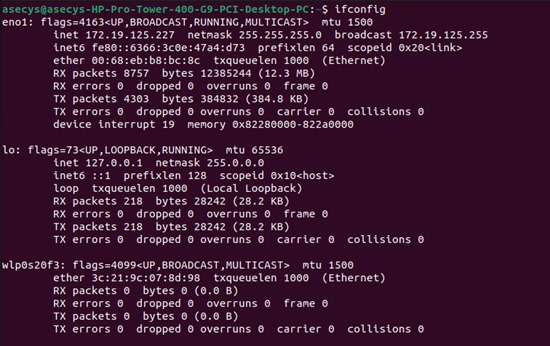
Procedure:

1. First, we need to have two machines for this experiment. We first find out the IP addresses of both the machines.

MACHINE 1:



MACHINE 2:



2. We install hping3 to our system using

sudo apt install hping3

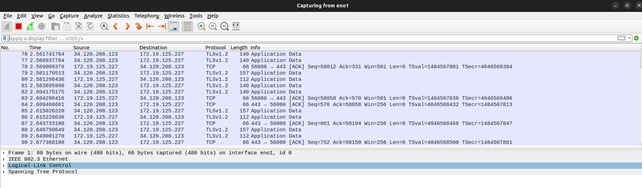


3. In the second machine we open a port using netcat. We use the command

nc lnvp [port-number]



4. We open wireshark and start listening to the traffic coming through.



5. In the first machine we give the following command,

sudo hping3 -S --flood -V -p 12345 172.19.125.227

where

-S: specifies SYN packets.

• –flood: replies will be ignored and packets will be sent as fast as possible.

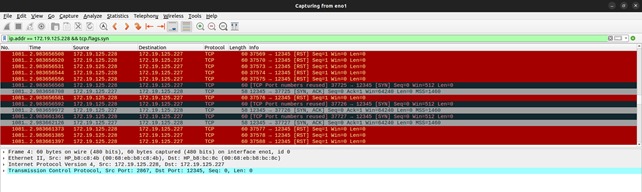
• -V: Verbosity.

• -p <portnumber>: port number

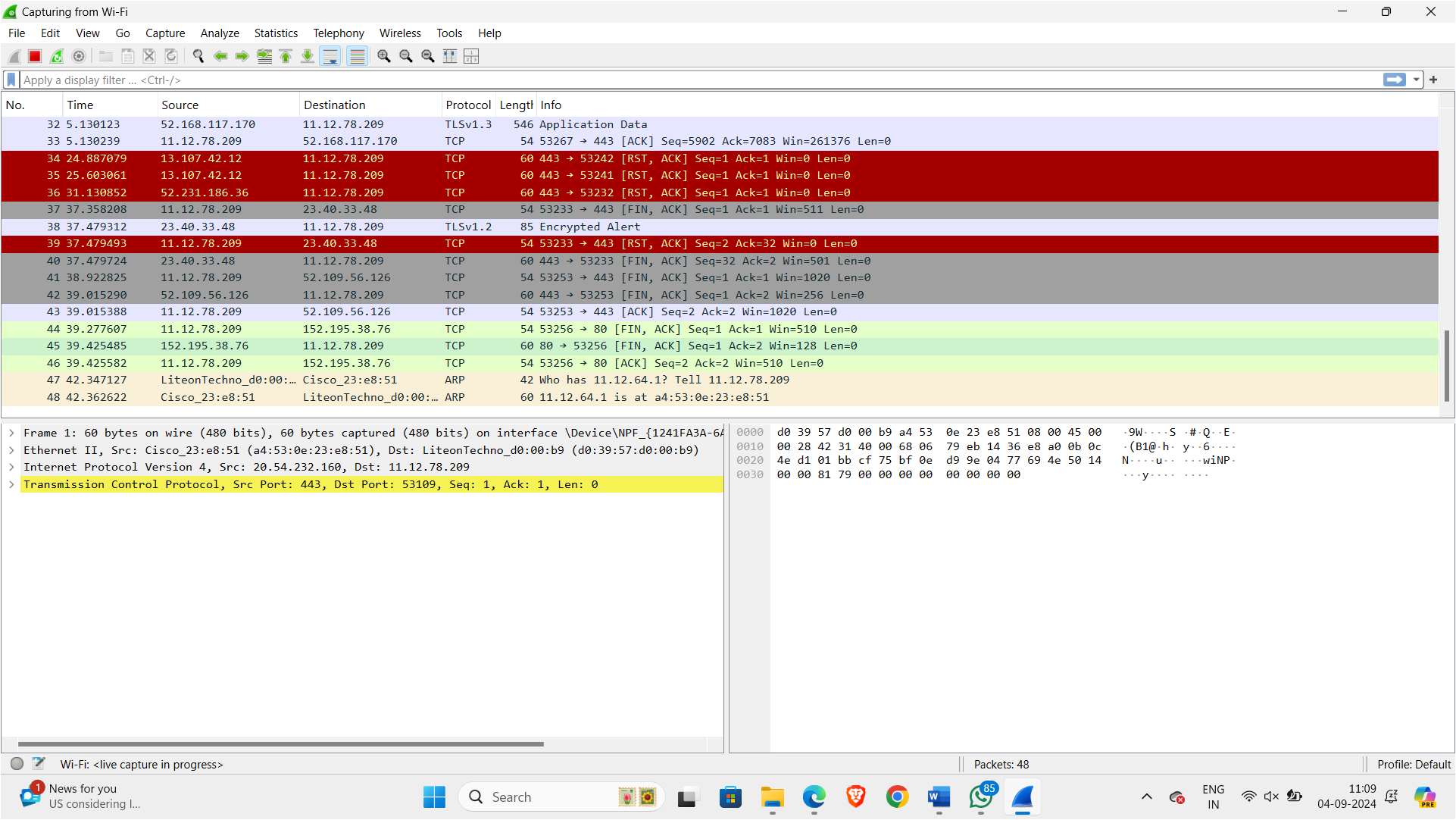


6. In the second machine, in wireshark we will see the flooded SYN packets. We can filter this using the command

ip.addr == <ip address of attacking machine> && tcp.flags.syn



1. **Wireshark:**
2. **Analyze the network traffic captured using Wireshark and identify the most commonly used network protocols.**



Most commonly used network protocols:

1. UDP

2. TCP

3. DNS

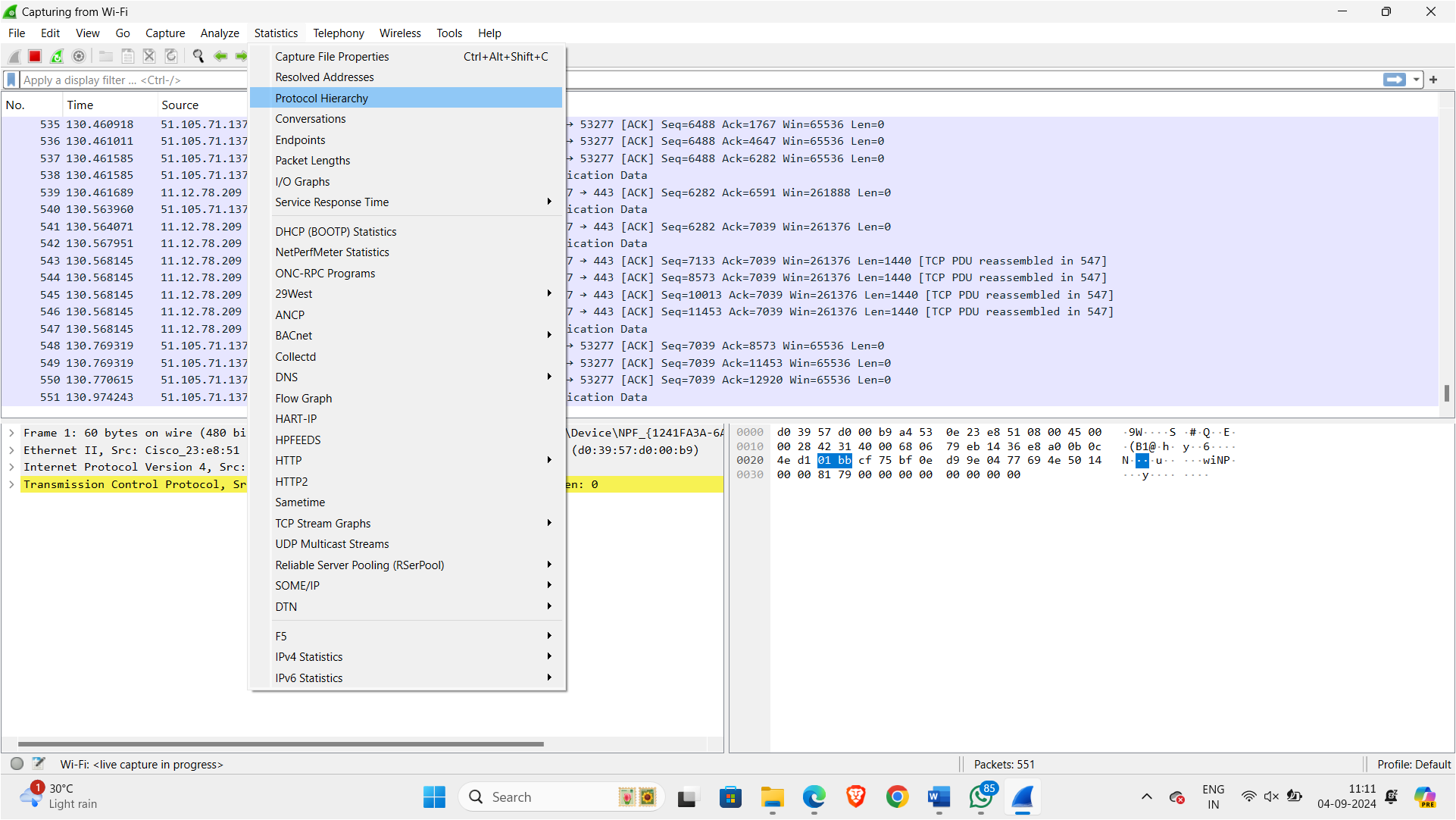
4. ARP

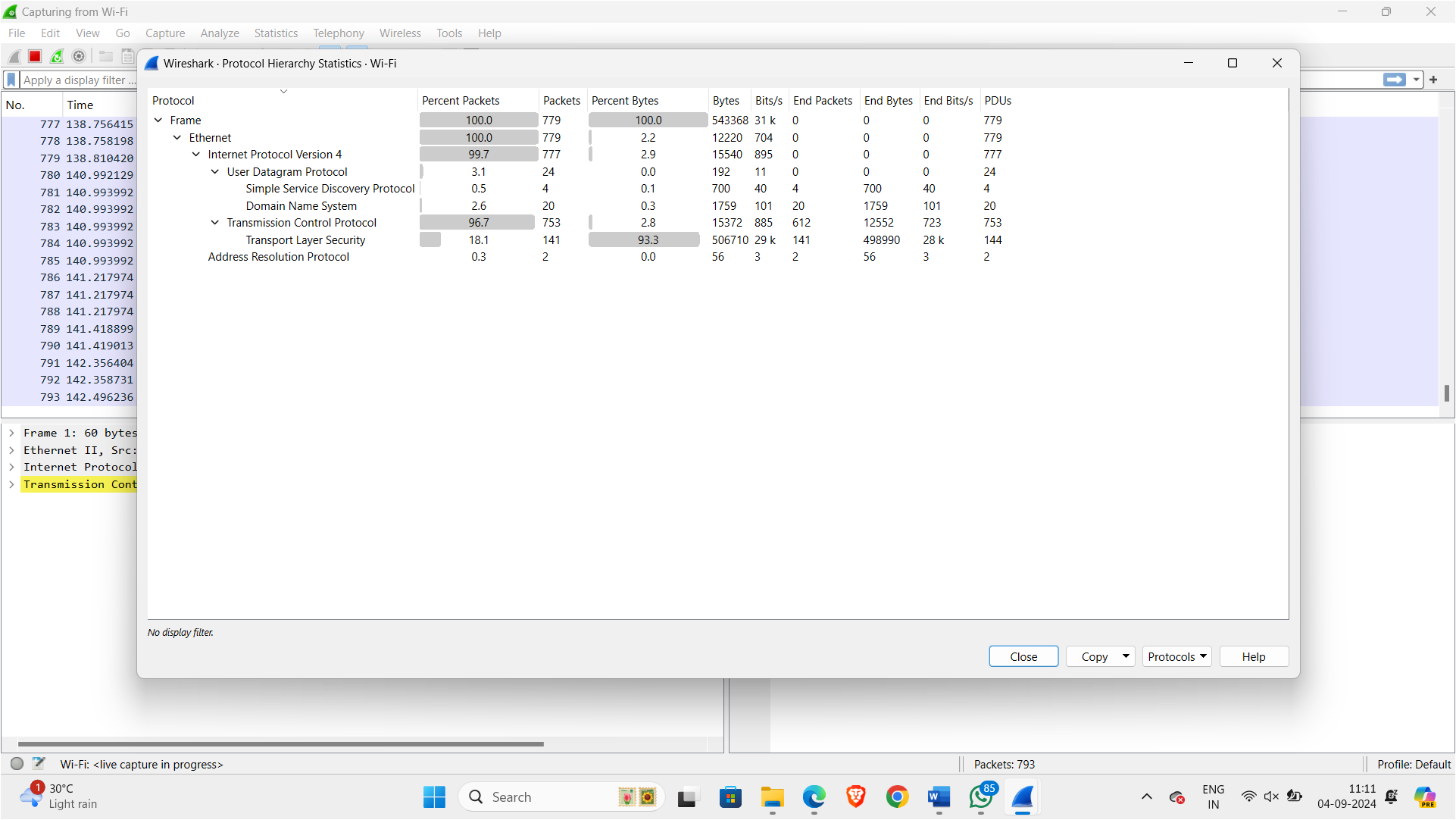
1. **Investigate a specific network issue by analyzing captured packets in Wireshark and propose a solution.**

**Steps:**

1. Start a capture in Wireshark.
2. Reproduce the network issue or wait for it to occur.

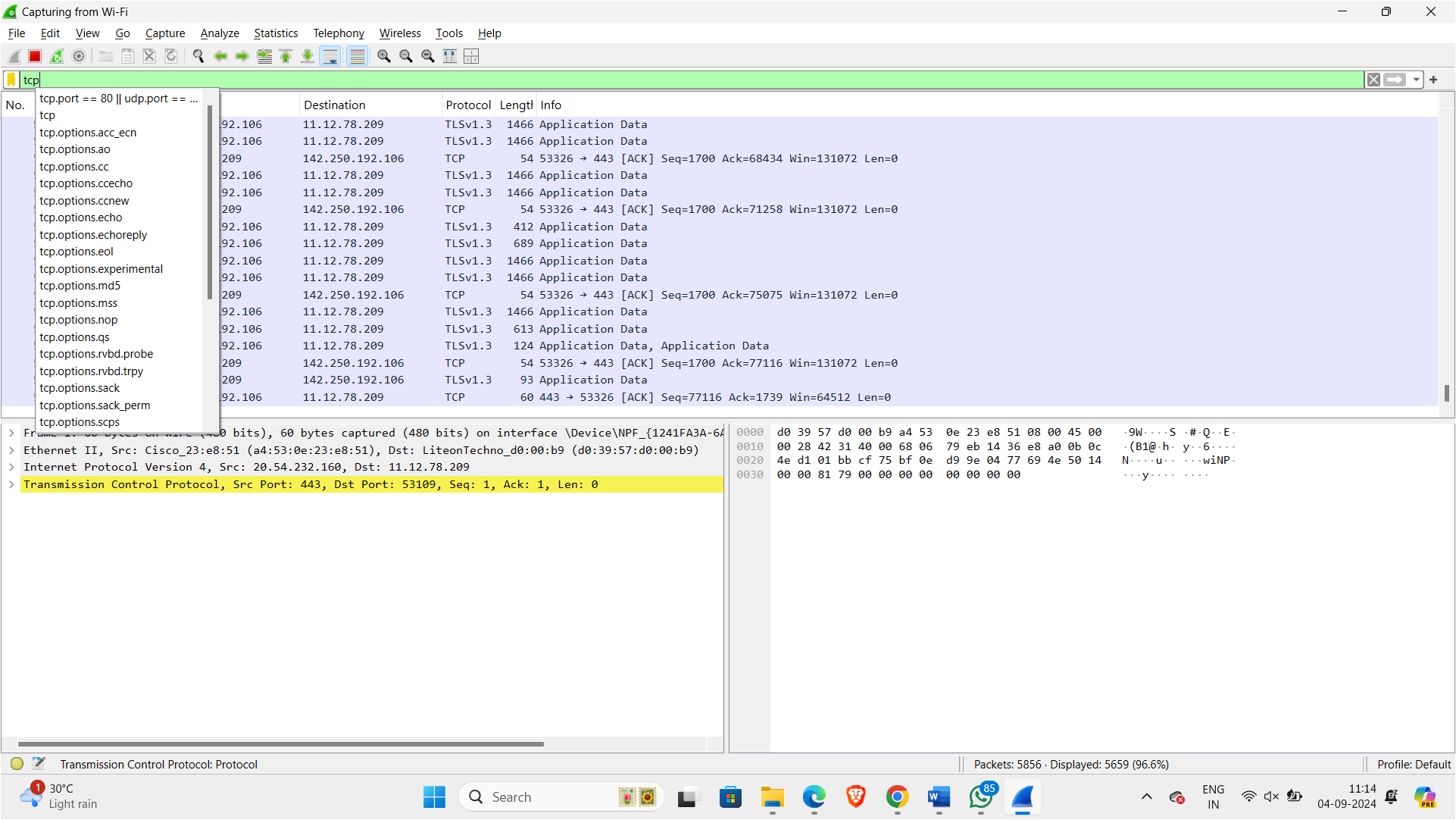
We are creating a dummy problem of network congestion

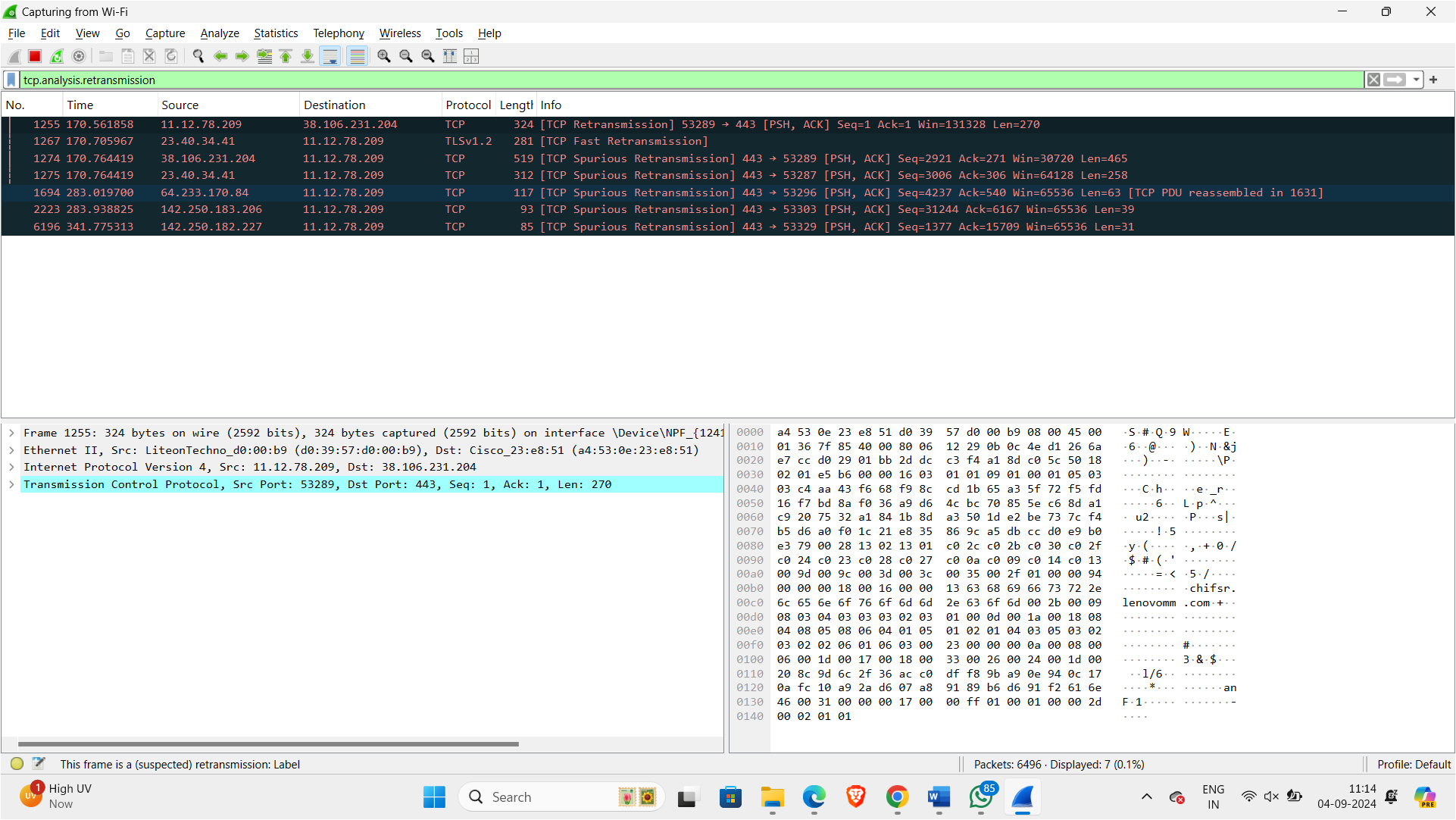




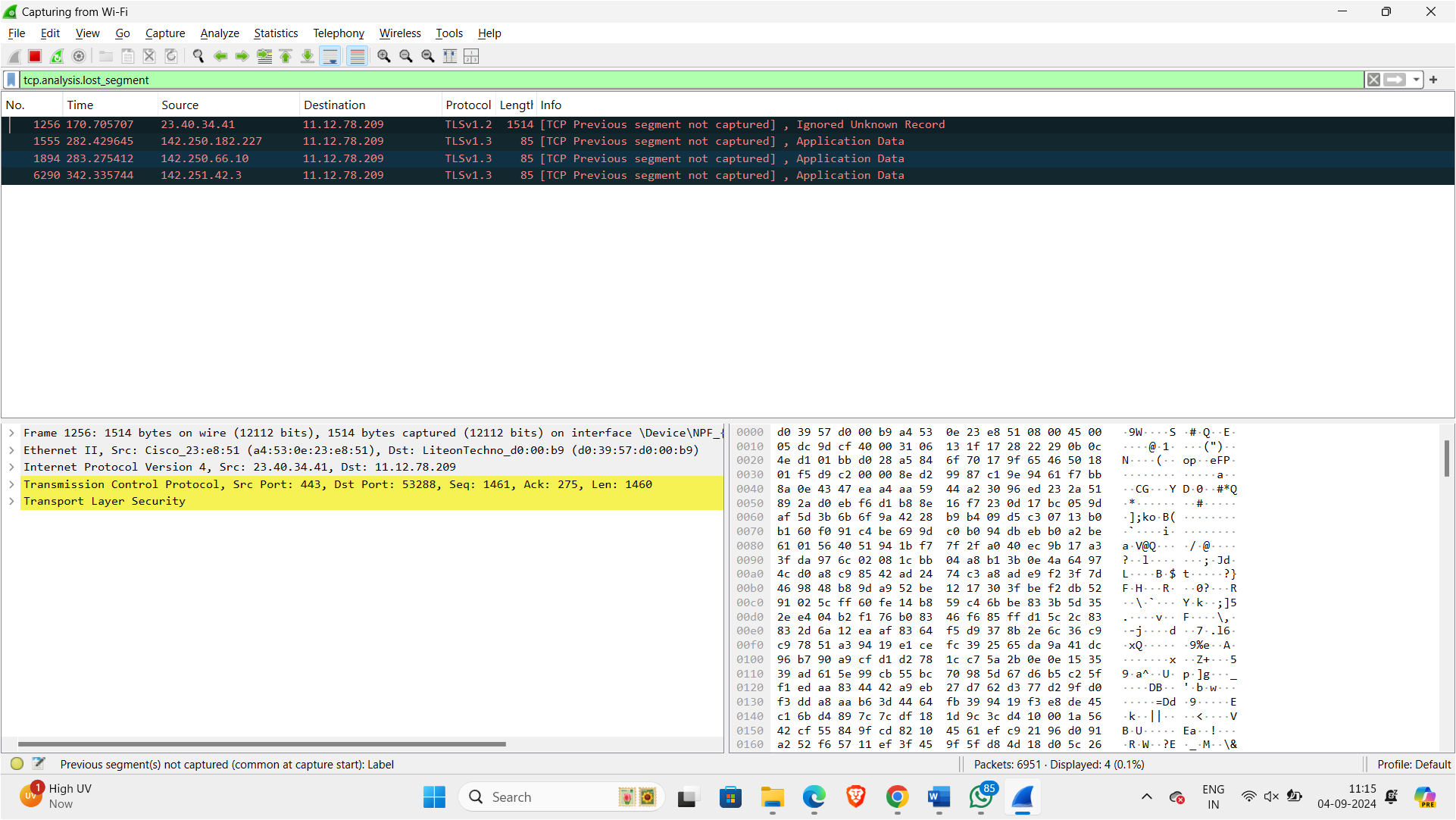
Analyze the packets by looking for anomalies or errors (e.g., retransmissions, TCP RST).

Analysing tcp retransmissions



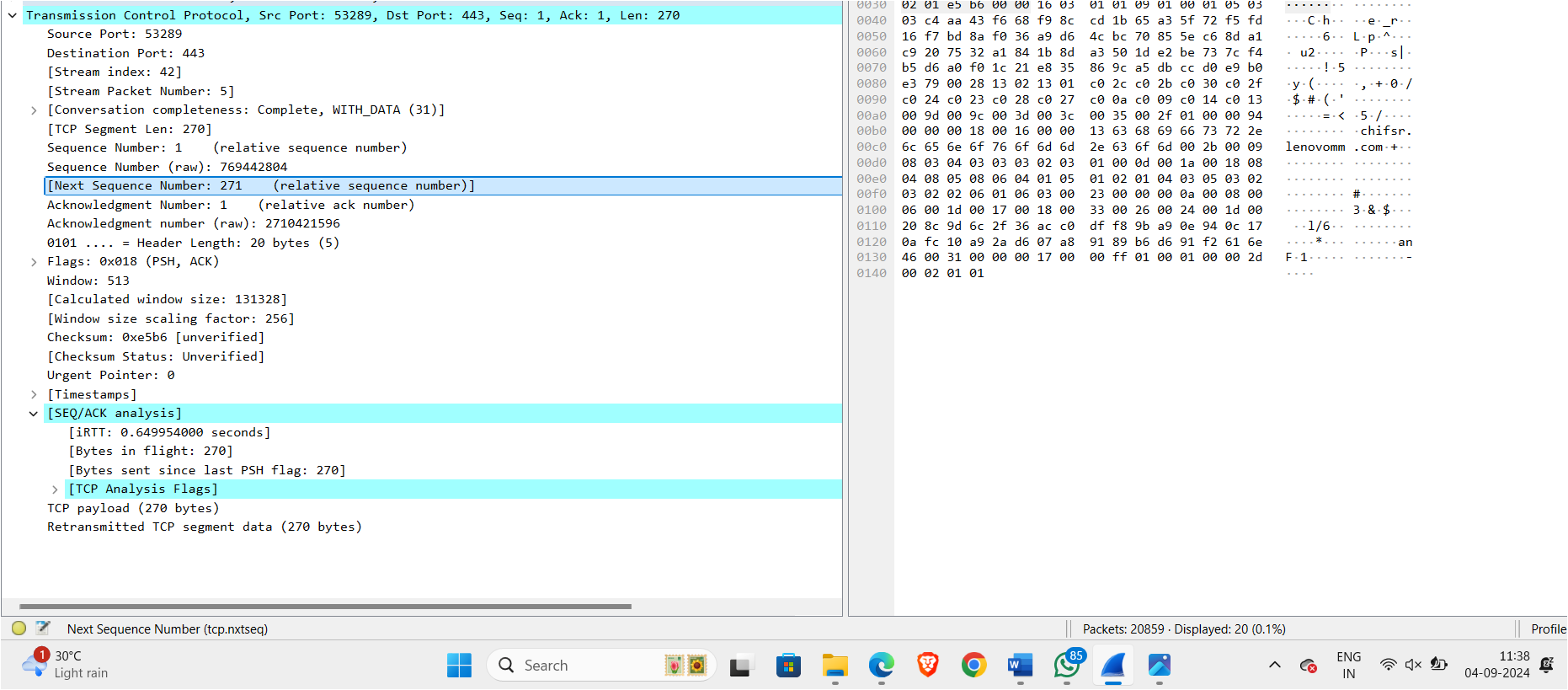


Analysing tcp lost\_segments:

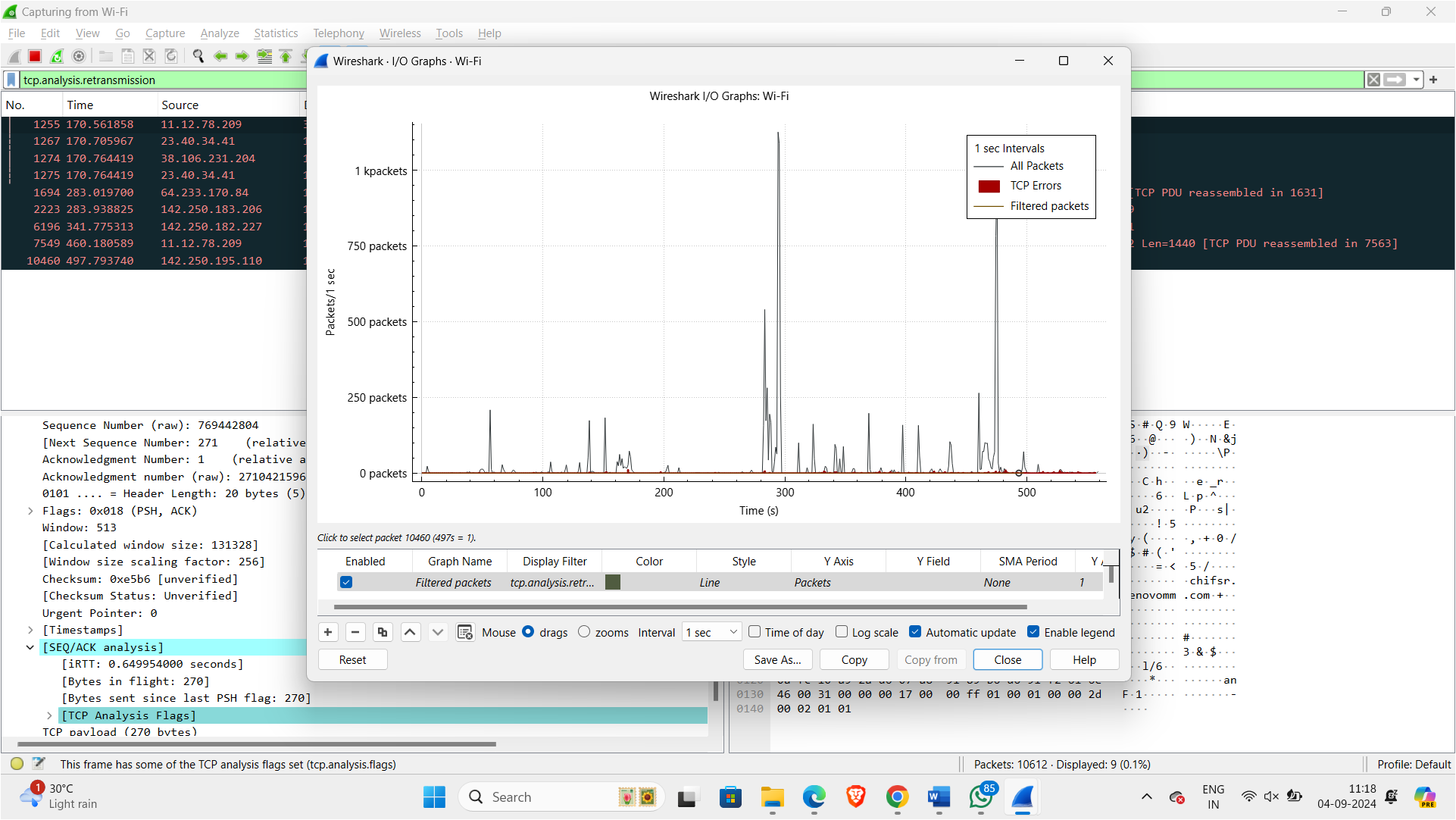


Use filters and follow streams to trace the issue.

Checking for packets with ‘TCP retransmission’ or TCP duppak’



Checking for latency rate



* Propose a solution based on the identified problem (e.g., network congestion, misconfiguration).

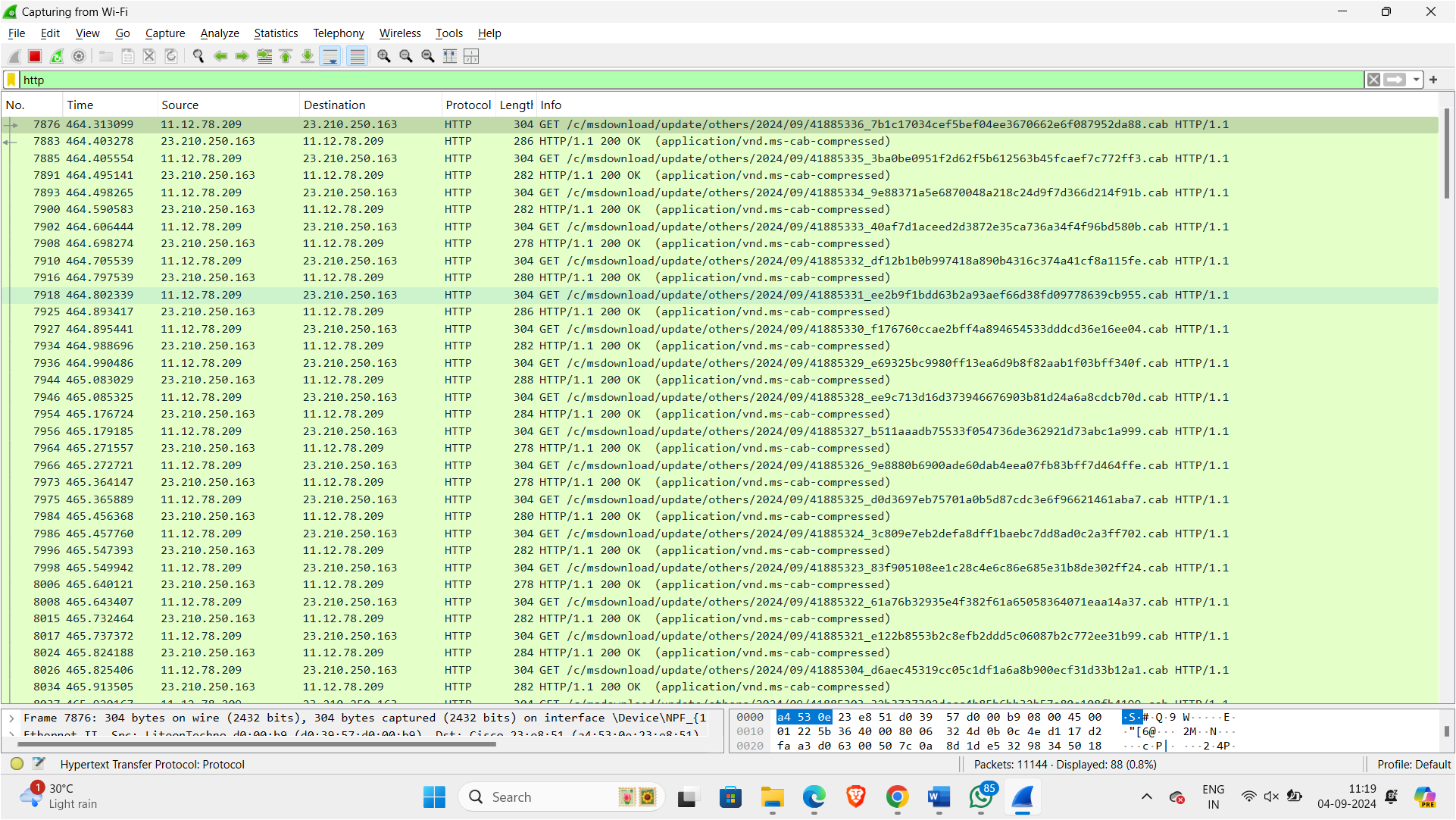
1. Reduce Traffic Load:
   * + Bandwidth Management: Use bandwidth management tools to control the amount of traffic generated by applications.
     + Prioritize Traffic: Implement Quality of Service (QoS) policies to prioritize critical traffic over less important traffic.
2. Optimize Network Configuration:

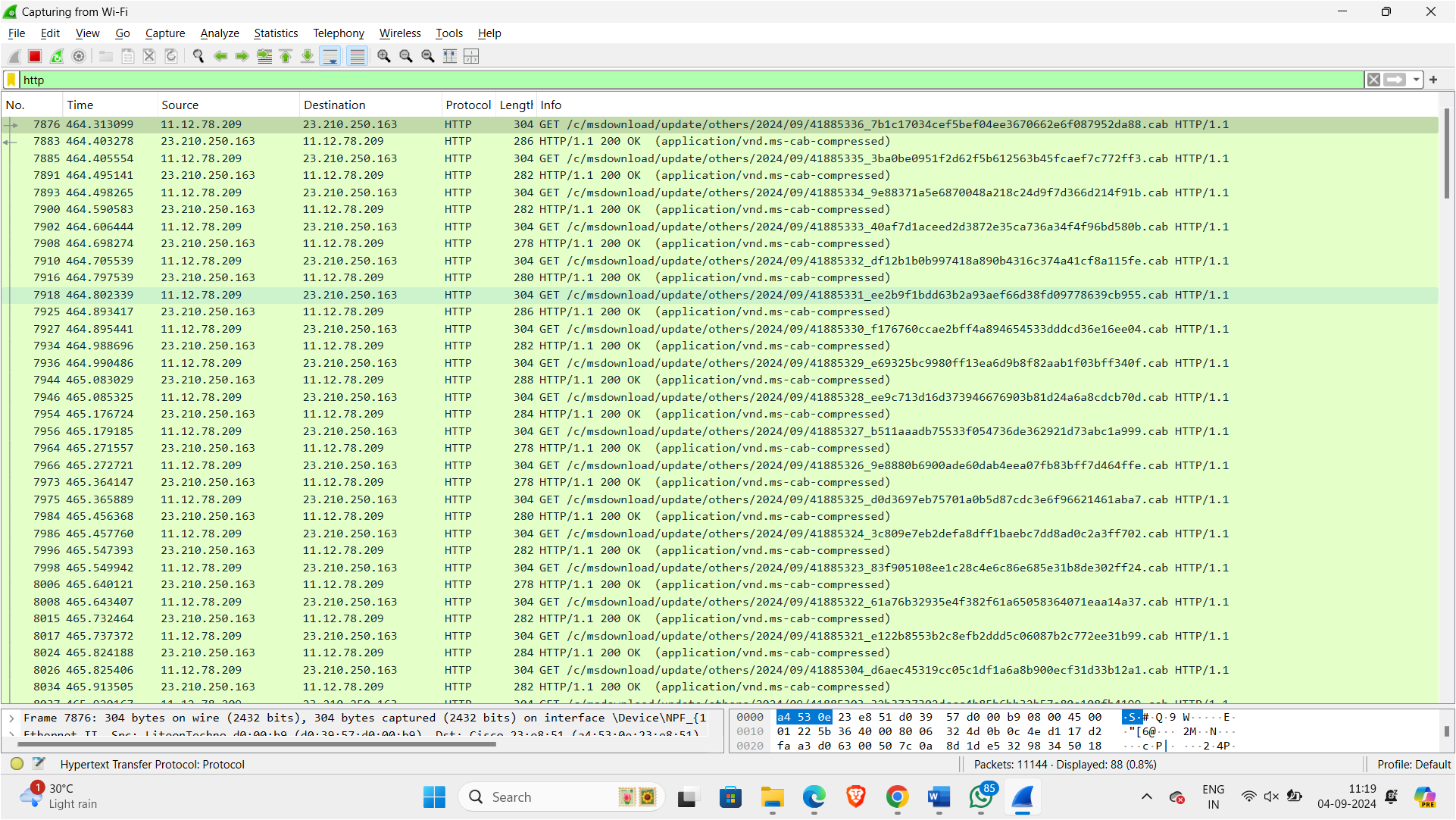
* Remove Latency and Packet Loss: Adjust the network settings to remove the artificial delay and packet loss you introduced. Run sudo tc qdisc del dev eth0 root netem to remove the network emulation.
* Increase Bandwidth: If the congestion is due to insufficient bandwidth, consider upgrading network infrastructure or increasing the bandwidth capacity.
* Monitor Network Performance:
* Continuous Monitoring: Set up regular network performance monitoring using tools like Wireshark, network analyzers, or SNMP-based monitoring tools to catch issues early.
* Test the Solution:
* Re-capture Traffic: After implementing the solution, capture and analyze traffic again to verify that latency and packet loss have been reduced.

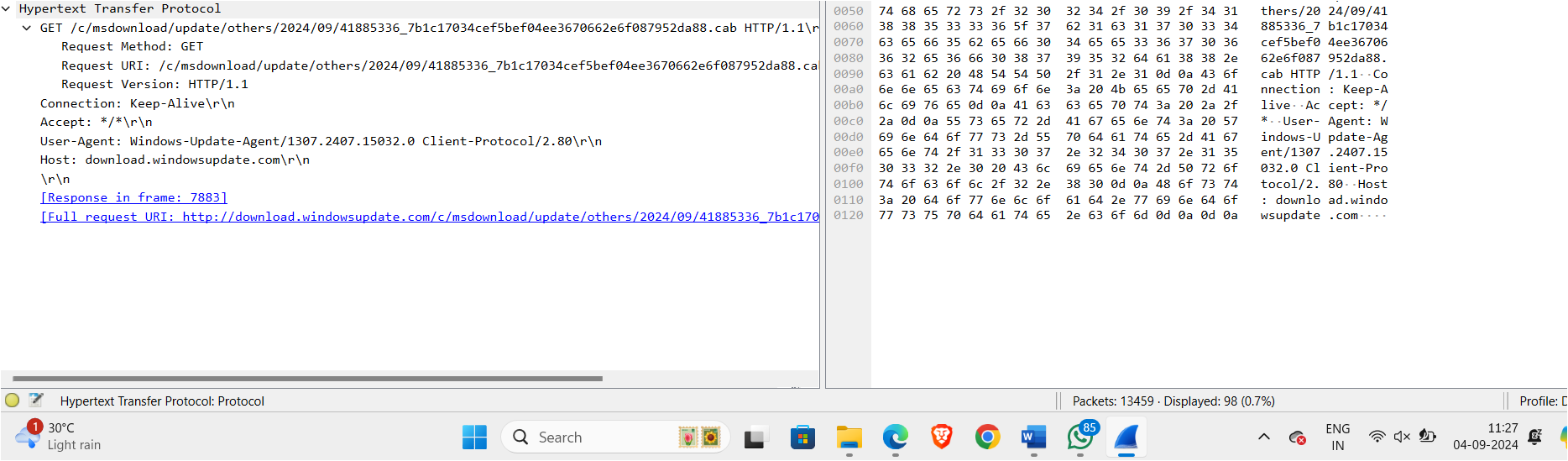
1. **Capture and analyze network packets to determine the source and destination IP addresses of a particular type of traffic (e.g., HTTP, DNS, FTP).**

Start Wireshark and capture traffic.

Apply a filter for the specific type of traffic, e.g., http, dns, or ftp.



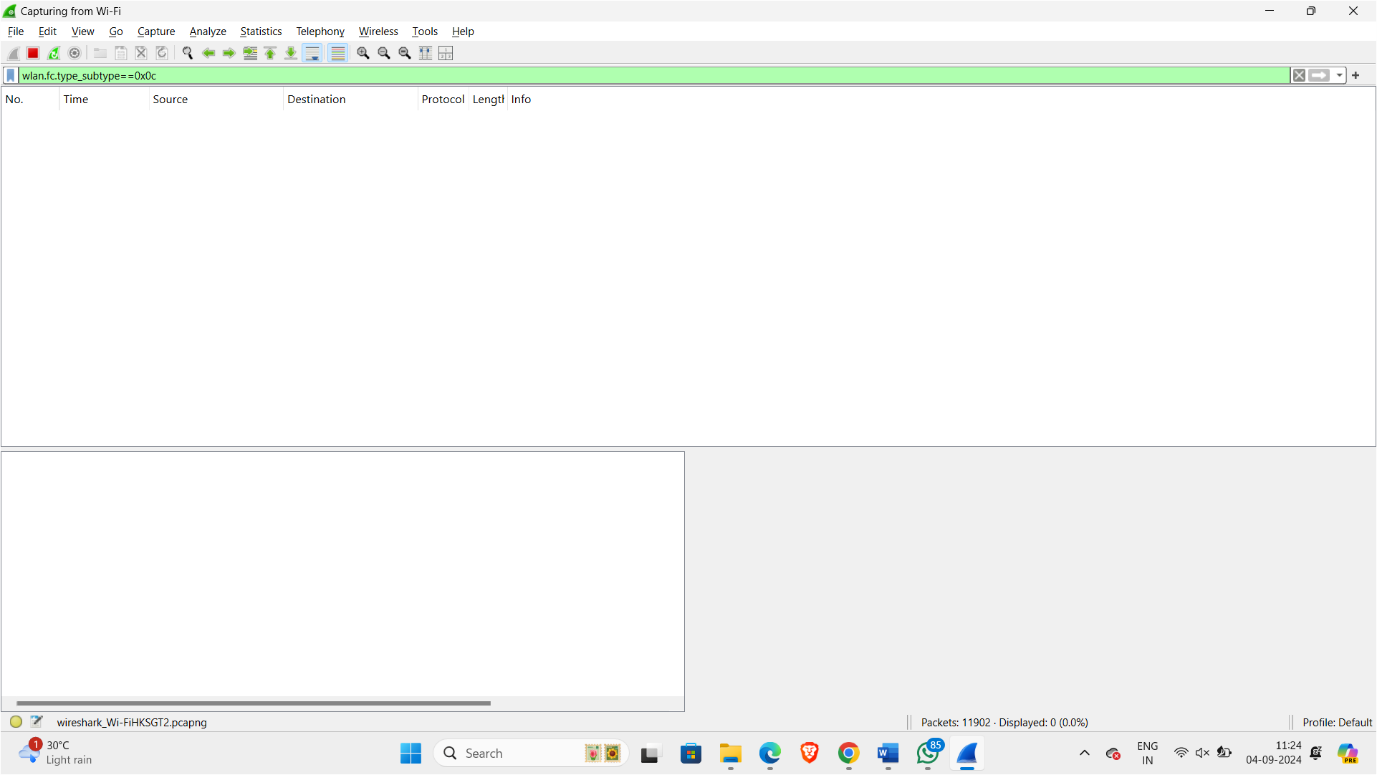
Examine the packets to identify source and destination IP addresses.  


Use the packet details to see the IP addresses in the “IP” or “Ethernet” layer.  


1. **Use Wireshark to capture and analyze wireless network traffic, and identify any security vulnerabilities or potential attacks**

* Start Wireshark and configure it for wireless monitoring (use a compatible wireless adapter).
* Capture wireless traffic on the appropriate channel.
* Stop the capture once you have enough data.
* Look for unencrypted traffic, unusual activity, or known attack patterns (e.g., deauthentication attacks).

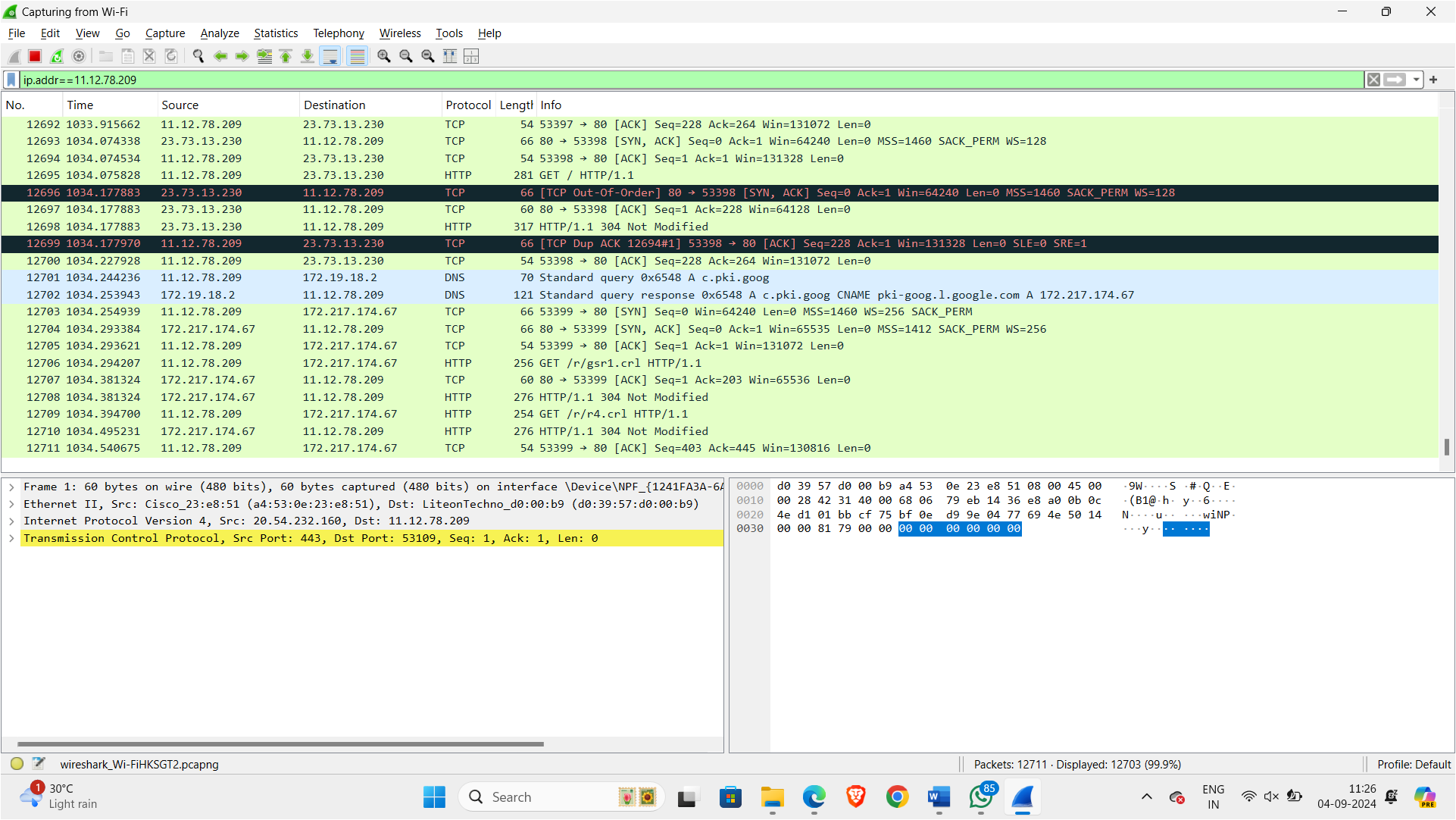
We can filter ‘wlan.fc.type\_subtype == 0x0c’ to find reauthentication attack.



* + Examine packet details to identify vulnerabilities.

1. **You suspect that a certain computer in your network is sending sensitive data to an external server. How would you use Wireshark filters to monitor the traffic of that specific machine and identify any suspicious activities?**

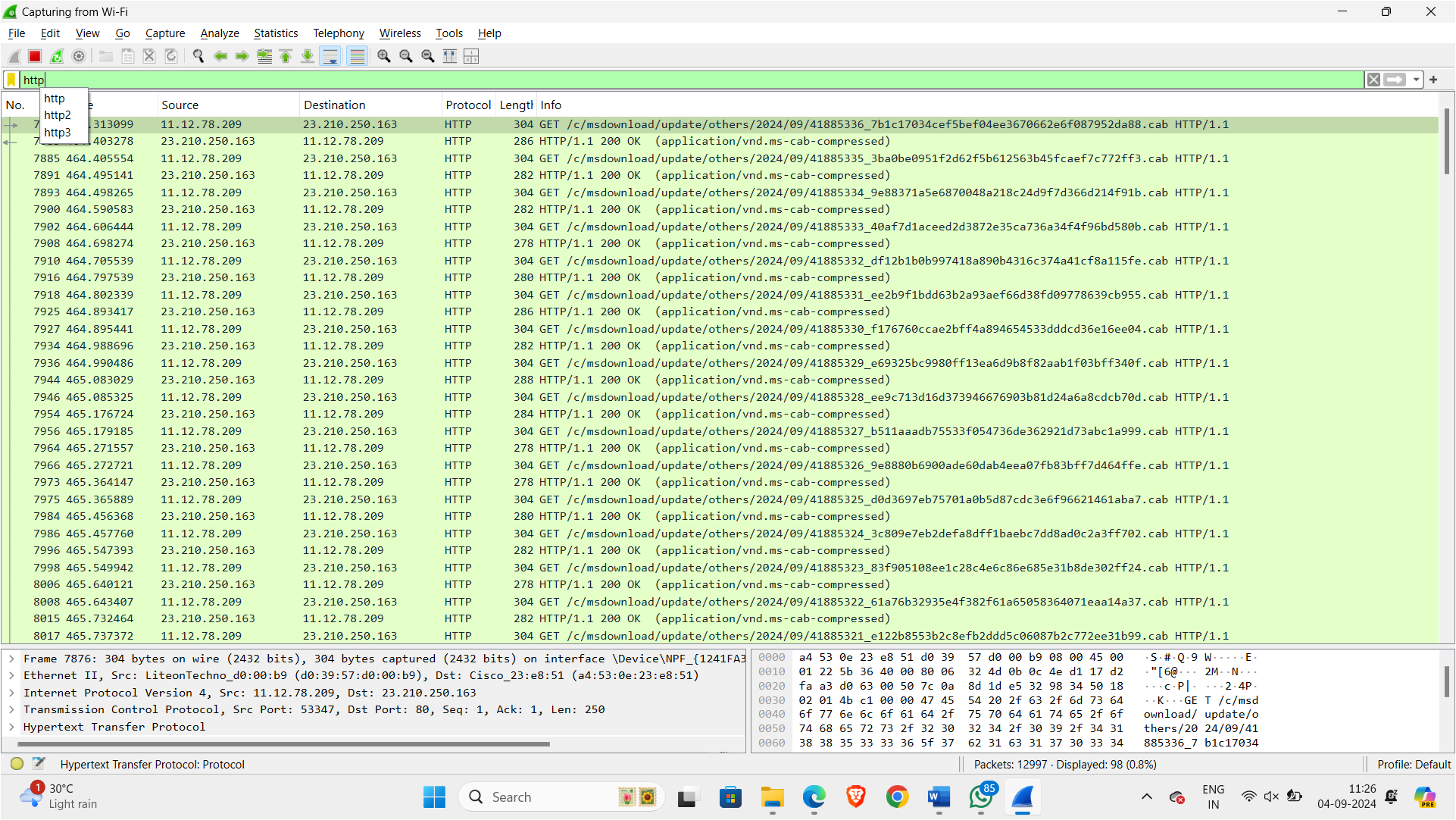
* Start Wireshark and capture all traffic.
* Apply a filter to focus on the specific machine’s IP address, e.g., ip.addr == x.x.x.x.



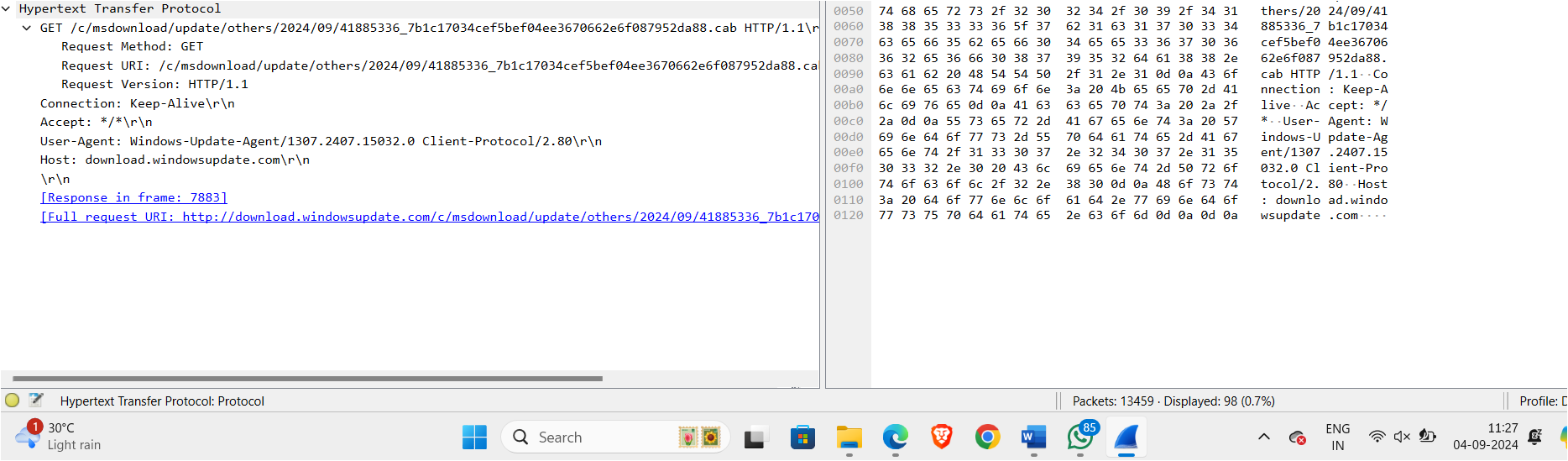
* Examine the captured packets for any sensitive or unusual data being sent.
* Look for signs of data exfiltration, like large volumes of data sent to external IPs.

1. **You need to analyze the HTTP traffic in your network to identify any potential security vulnerabilities. How would you use Wireshark filters to focus specifically on HTTP packets and extract relevant information for analysis?**

* Start Wireshark and capture HTTP traffic.
* Apply an HTTP filter, e.g., http.



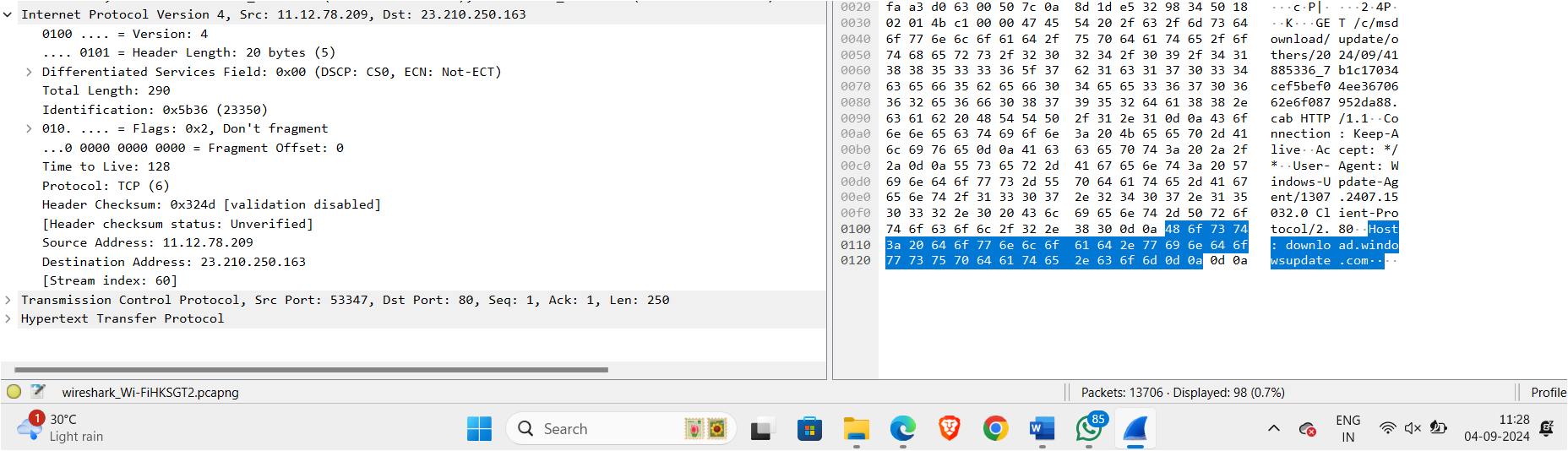
* Examine HTTP packets for sensitive data, improper use of HTTP methods, or security headers (e.g., missing Secure flag in cookies).
* **Inspect Packet Details:** Click on individual HTTP packets in the packet list pane to view their details in the packet details pane. Expand the Hypertext Transfer Protocol section to view the HTTP headers and payload.
* **Look for Sensitive Data:** HTTP Requests: Check the Request URI and Request Body for any sensitive information such as passwords, personal data, or API keys.
* **HTTP Responses**: Inspect the Response Body for sensitive data that might be exposed, such as user data or authentication tokens.

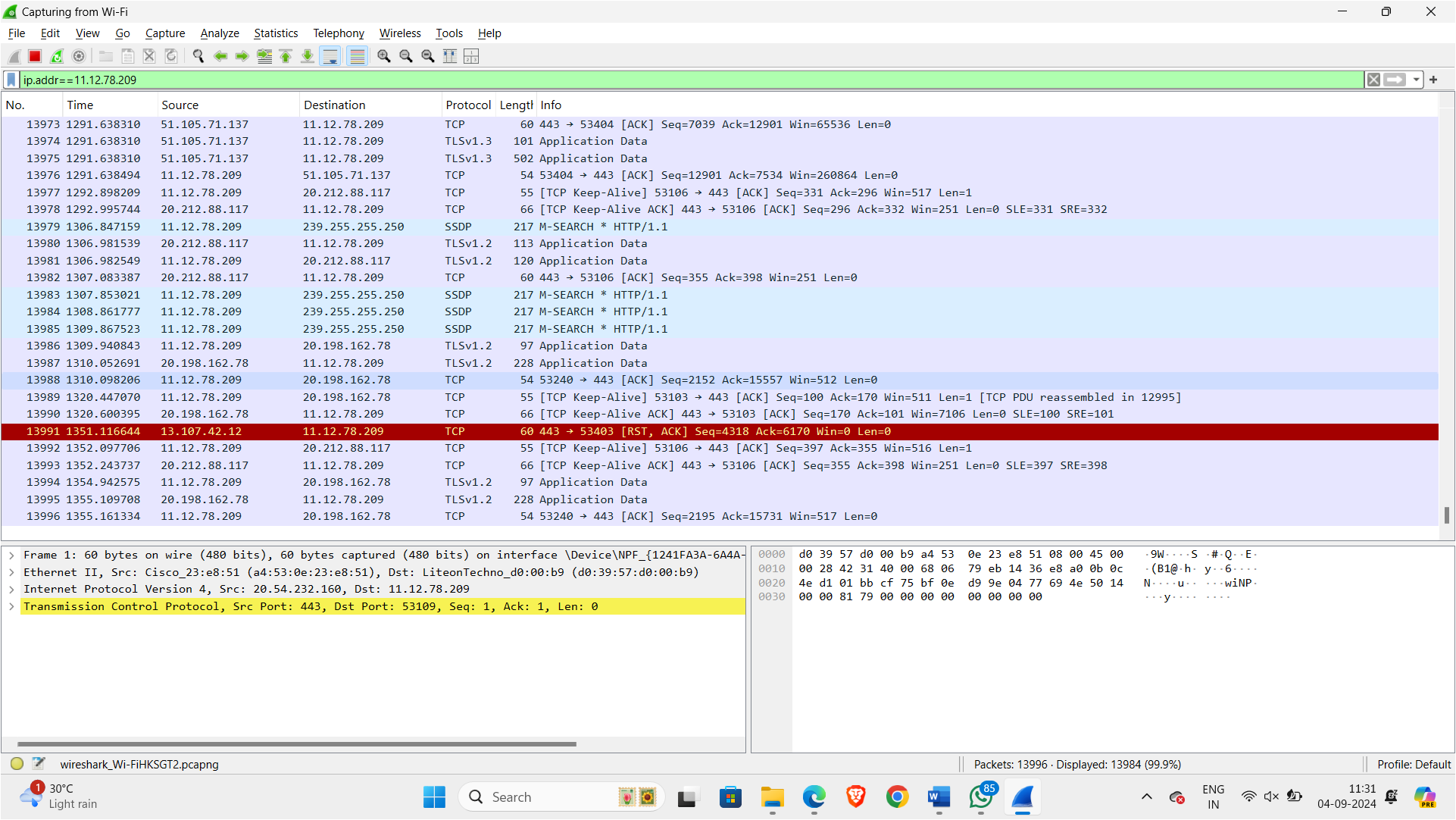


* Analyze response and request payloads for vulnerabilities like SQL injection or XSS.

1. **Your company has recently implemented a new application, and you need to monitor the network traffic to assess its impact on network performance. How would you use Wireshark filters to analyze the traffic related to this specific application?**

* Start Wireshark and capture traffic.
* Identify the application's traffic using known ports or protocols.



1. Apply a filter to capture traffic related to the application, e.g., tcp port 12345 if the app uses that port.
2. Analyze traffic patterns and volume to assess the impact on network performance.
3. **Write a C or C++ program to development of a packet capture and filtering application using raw sockets.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <pcap.h>

#include <arpa/inet.h>

#include <netinet/ip.h>

#include <netinet/tcp.h>

// Define a structure for the packet filter

typedef struct {

char \*ip\_address;

} packet\_filter\_t;

// IP header

struct ip\_hdr {

unsigned char ihl:4, version:4;

unsigned char tos;

unsigned short tot\_len;

unsigned short id;

unsigned short frag\_off;

unsigned char ttl;

unsigned char protocol;

unsigned short check;

unsigned int saddr;

unsigned int daddr;

};

// TCP header

struct tcp\_hdr {

unsigned short source;

unsigned short dest;

unsigned int seq;

unsigned int ack\_seq;

unsigned char doff:4, res:4;

unsigned char flags;

unsigned short window;

unsigned short check;

unsigned short urg\_ptr;

};

// Function to handle captured packets

void packet\_handler(u\_char \*user\_data, const struct pcap\_pkthdr \*pkthdr, const u\_char \*packet) {

packet\_filter\_t \*filter = (packet\_filter\_t \*)user\_data;

struct ip\_hdr \*iph = (struct ip\_hdr \*)(packet + 14); // Assuming Ethernet header length of 14 bytes

// Convert source IP address to string

char src\_ip[INET\_ADDRSTRLEN];

inet\_ntop(AF\_INET, &iph->saddr, src\_ip, sizeof(src\_ip));

// Print packet information if it matches the filter

if (filter->ip\_address != NULL && strcmp(src\_ip, filter->ip\_address) == 0) {

printf("Captured packet from IP: %s\n", src\_ip);

}

}

int main(int argc, char \*argv[]) {

if (argc != 3) {

fprintf(stderr, "Usage: %s <interface> <filter\_ip>\n", argv[0]);

return EXIT\_FAILURE;

}

char \*dev = argv[1];

char \*filter\_ip = argv[2];

char errbuf[PCAP\_ERRBUF\_SIZE];

// Open the network device for packet capture

pcap\_t \*handle = pcap\_open\_live(dev, BUFSIZ, 1, 1000, errbuf);

if (handle == NULL) {

fprintf(stderr, "Could not open device %s: %s\n", dev, errbuf);

return EXIT\_FAILURE;

}

// Set up packet filter

packet\_filter\_t filter;

filter.ip\_address = filter\_ip;

// Start packet capture

if (pcap\_loop(handle, 0, packet\_handler, (u\_char \*)&filter) < 0) {

fprintf(stderr, "Error occurred while capturing packets: %s\n", pcap\_geterr(handle));

pcap\_close(handle);

return EXIT\_FAILURE;

}

pcap\_close(handle);

return EXIT\_SUCCESS;

}

Befor implementing this we need to install some other libraries:

sudo apt-get install libpcap-dev

Then run this code by

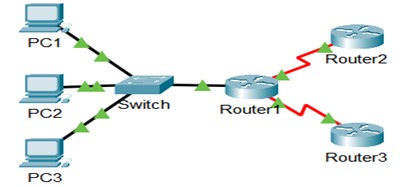
gcc -o packet\_capture packet\_capture.c -lpcap

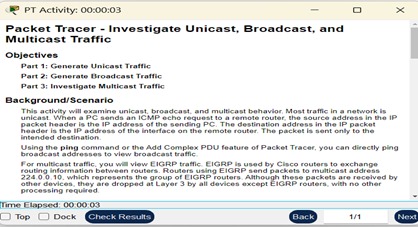
sudo ./packet\_capture <network\_interface> <filter\_ip>

1. **CISCO:**

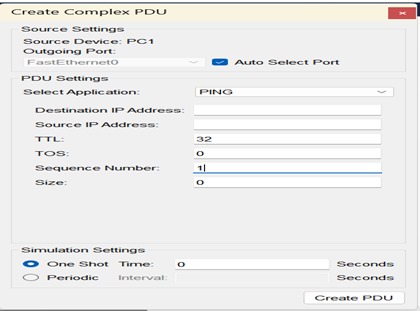
Unicast, Broadcast and Multicast in Cisco Packet Tracer

1. The tasks in the given cisco packet file are visible in the first dialog box. The tasks are to be followed in the order they are told.



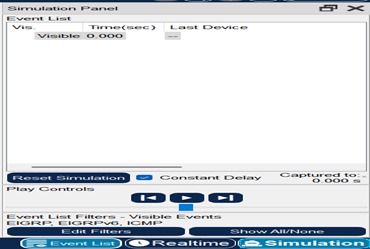


1. Create a complex PDU(Protocol data unit) by using the open envelope and click it on one of the PCs and give details accordingly.

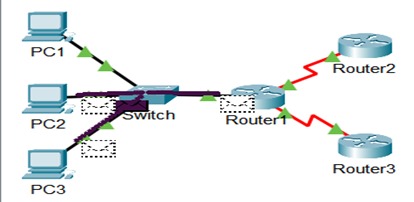


• If unicast is being done, then an IP address of one of the systems are kept in destination IP address. In the case of broadcast, 255.255.255.255 is kept which is broadcast address. After this, click create PDU.

1. Now open simulation window by pressing the button.



• Press the next button to see the PDU moving from devices to devices.



4. On further pressing the next button, the PDU packets are seen moving from different components to different components. The packets move according to the destination IP address typed.

1. **SMTP:**

**Code:**

import socket

import base64

import ssl

# SMTP server details

SMTP\_SERVER = 'mail.example.com'

SMTP\_PORT = 587

USERNAME = 'mail-address@server.com'

PASSWORD = 'password'

# Email details

from\_address = 'mail-address@server.com'

to\_address = 'receiver@server.com'

subject = 'Test Email'

body = 'This is a test email sent using SMTP over TCP.'

def send\_smtp\_command(sock, command):

print(f'Sending: {command}')

sock.sendall((command + '\r\n').encode())

response = sock.recv(1024).decode()

print(f'Response: {response}')

return response

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

sock.connect((SMTP\_SERVER, SMTP\_PORT))

response = sock.recv(1024).decode()

print(f'Server: {response}')

send\_smtp\_command(sock, f'EHLO {socket.gethostname()}')

send\_smtp\_command(sock, 'STARTTLS')

context = ssl.create\_default\_context()

sock = context.wrap\_socket(sock, server\_hostname=SMTP\_SERVER)

send\_smtp\_command(sock, f'EHLO {socket.gethostname()}')

send\_smtp\_command(sock, 'AUTH LOGIN')

send\_smtp\_command(sock, base64.b64encode(USERNAME.encode()).decode())

send\_smtp\_command(sock, base64.b64encode(PASSWORD.encode()).decode())

send\_smtp\_command(sock, f'MAIL FROM: <{from\_address}>')

send\_smtp\_command(sock, f'RCPT TO: <{to\_address}>')

send\_smtp\_command(sock, 'DATA')

email\_data = f'From: {from\_address}\r\nTo: {to\_address}\r\nSubject: {subject}\r\n\r\n{body}\r\n.\r\n'

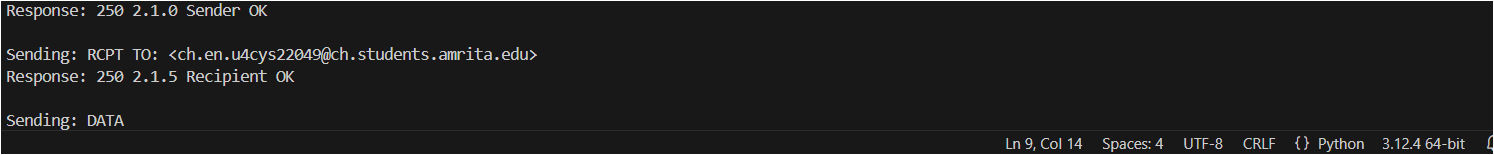
send\_smtp\_command(sock, email\_data)

send\_smtp\_command(sock, 'QUIT')

sock.close()

**Output:**

****

****

**Explanation:**

• Establishes a connection to the SMTP server: The script creates a TCP socket and connects to the SMTP server (mail.example.com) on port 587, a common port for email submission. It initiates communication with the SMTP server and receives the server's initial response.

• Performs TLS encryption and authentication: After sending the EHLO command to the server, the script uses the STARTTLS command to initiate a secure TLS connection. It then wraps the socket in SSL using Python's ssl module to encrypt the communication. The script re-sends the EHLO command and performs SMTP authentication using the AUTH LOGIN command, followed by sending the base64-encoded username and password.

• Composes and sends the email: After authentication, the script sends the MAIL FROM command to specify the sender's email address and the RCPT TO command to specify the recipient's email address. It then sends the DATA command, followed by the email's subject, body, and ending with a period (.) on a new line to signal the end of the message content.

• Closes the connection: Once the email is sent, the script sends the QUIT command to terminate the SMTP session and closes the socket, completing the process of sending the email.