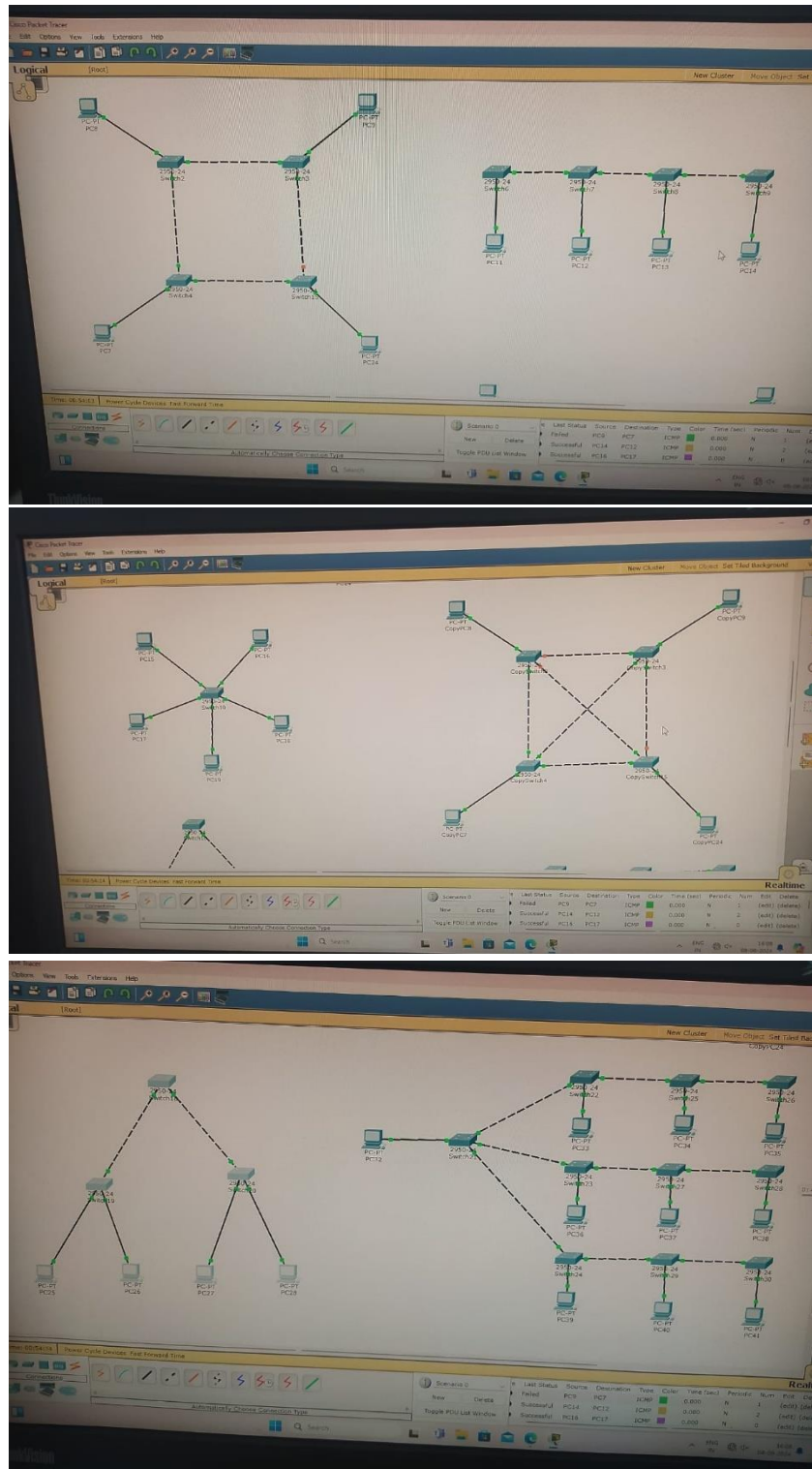


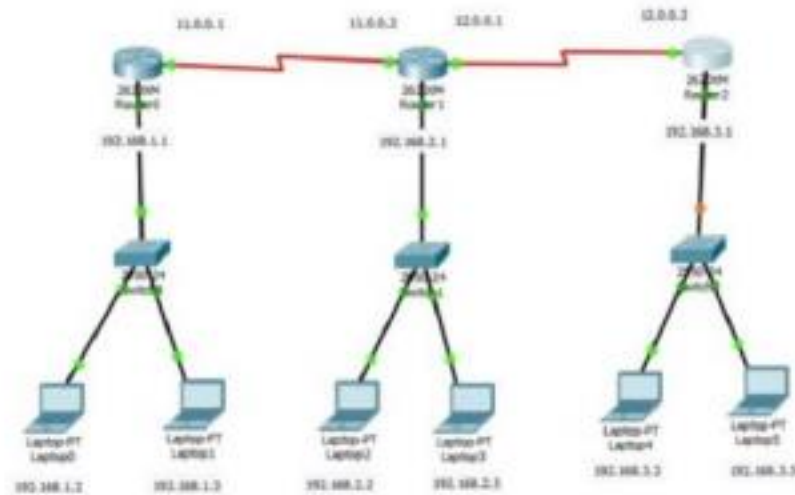
Name: Saanika Wani

Roll no: 32568

ASSIGNMENT 1(LA 1)



ASSIGNMENT 2 (LA 4)



ASSIGNMENT 3 (LA 5)

```
import java.util.Scanner;

class SimpleSubnetting {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        // Prompt for IP address input
        System.out.print("Enter the IP address (format: xxx.xxx.xxx.xxx): ");
        String ipAddress = sc.nextLine();

        // Prompt for subnet mask input
        System.out.print("Enter the number of subnet bits (1-30): ");
        int subnetBits = sc.nextInt();

        if (subnetBits < 1 || subnetBits > 30) {
```

```

        System.out.println("Subnet bits must be between 1 and 30.");
        sc.close();
        return; // Exit the program if invalid
    }

    // Calculate subnet mask
    int subnetMask = (int) (Math.pow(2, subnetBits) - 1) << (32 - subnetBits);
    String subnetMaskString = convertToDottedDecimal(subnetMask);

    // Calculate the number of host bits
    int hostBits = 32 - subnetBits;
    int numberOfHosts = (int) Math.pow(2, hostBits) - 2; // -2 for network and
broadcast addresses

    System.out.println("\nIP Address: " + ipAddress);
    System.out.println("Subnet Mask: " + subnetMaskString);
    System.out.println("Total number of hosts per subnet: " + numberOfHosts);

    sc.close();
}

// Function to convert a subnet mask integer to dotted decimal format
public static String convertToDottedDecimal(int mask) {
    return ((mask >> 24) & 0xFF) + "." +
        ((mask >> 16) & 0xFF) + "." +
        ((mask >> 8) & 0xFF) + "." +
        (mask & 0xFF);
}
}

```

ASSIGNMENT 4 (LA 6)

class NetworkDV:

```
def __init__(self, num_nodes):
    self.num_nodes = num_nodes
    self.graph = [[float('inf')] * num_nodes for _ in range(num_nodes)]
    for i in range(num_nodes):
        self.graph[i][i] = 0 # Distance to itself is always 0
```

```
def add_link(self, u, v, weight):
    # Add a link (edge) between u and v
    self.graph[u][v] = weight
    self.graph[v][u] = weight
```

```
def bellman_ford(self, start):
    distances = [float('inf')] * self.num_nodes
    distances[start] = 0

    for _ in range(self.num_nodes - 1):
        for u in range(self.num_nodes):
            for v in range(self.num_nodes):
                if self.graph[u][v] != float('inf'):
                    new_distance = distances[u] + self.graph[u][v]
                    if new_distance < distances[v]:
                        distances[v] = new_distance

    return distances
```

Example usage

```
if __name__ == "__main__":  
    network = NetworkDV(5) # Create a network with 5 nodes (0 to 4)  
  
    # Add links (edges) between nodes  
    network.add_link(0, 1, 2)  
    network.add_link(0, 2, 4)  
    network.add_link(1, 2, 1)  
    network.add_link(1, 3, 7)  
    network.add_link(2, 3, 3)  
    network.add_link(3, 4, 1)  
  
    start_node = 0  
    distances = network.bellman_ford(start_node)  
  
    print(f"Shortest paths from node {start_node}:")  
    for node, distance in enumerate(distances):  
        print(f"Node {node} -> Distance {distance}")
```

ASSIGNMENT 5 (LA 7)

TCP Client:

```
import java.net.*;
```

```
import java.io.*;
```

```
public class TCPClient {  
    public static void main(String[] args) {  
        try {  
            // Connect to the server running on localhost, port 5000  
            Socket client = new Socket("localhost", 5000);  
            System.out.println("Client is connected");  
  
            // Input stream to read data from the server  
            BufferedReader br = new BufferedReader(new  
InputStreamReader(client.getInputStream()));  
  
            // Output stream to send data to the server  
            PrintWriter out = new PrintWriter(client.getOutputStream(), true);  
  
            BufferedReader userInput = new BufferedReader(new  
InputStreamReader(System.in));  
  
            new Thread(() -> {  
                String serverData;  
                try {  
                    while ((serverData = br.readLine()) != null) {  
                        System.out.println("Data from server: " + serverData);  
                    }  
                } catch (IOException e) {  
                    System.out.println("Server disconnected.");  
                }  
            })
```

```

    }).start();

    String message;
    while (true) {
        System.out.print("Enter message to send to server (or 'exit' to quit): ");
        message = userInput.readLine();
        if ("exit".equalsIgnoreCase(message)) {
            break;
        }
        out.println(message); // Send the message to the server
    }

    br.close();
    out.close();
    client.close();
} catch (IOException e) {
    System.err.println("Connection error: " + e.getMessage());
}
}
}

```

TCP Server

```

import java.net.*;
import java.io.*;
import java.util.*;

public class TCPServer {
    public static void main(String[] args) {
        ServerSocket ss = null;
        Socket server = null;
    }
}

```

```

    PrintWriter pw = null;

    Scanner sc = new Scanner(System.in);

    try {

        ss = new ServerSocket(5000); // Create server socket on port 5000

        System.out.println("Waiting for client...");

        server = ss.accept(); // Wait for the client to connect

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

        pw = new PrintWriter(server.getOutputStream(), true); // Auto-flush enabled

        BufferedReader br = new BufferedReader(new
InputStreamReader(server.getInputStream()));

        new Thread(() -> {

            String clientMessage;

            try {

                while ((clientMessage = br.readLine()) != null) {

                    System.out.println("Message from client: " + clientMessage); // Print
client messages

                }

            } catch (IOException e) {

                System.err.println("Client disconnected.");

            }

        }).start();

        String data;

        while (true) {

            System.out.println("Enter data (type 'exit' to quit):");

```



```

        data = sc.nextLine();
        if ("exit".equalsIgnoreCase(data)) {
            break; // Exit the loop if the user types "exit"
        }
        pw.println(data); // Send data to client
    }

} catch (IOException e) {
    System.err.println("Error: " + e.getMessage());
} finally {

    try {
        if (pw != null) pw.close();
        if (server != null) server.close();
        if (ss != null) ss.close();
        sc.close();
    } catch (IOException e) {
        System.err.println("Error closing resources: " + e.getMessage());
    }
}
}
}

```

UDP Client

```

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

```

```

public class UDPClient {
    public static void main(String[] args) {
        DatagramSocket socket = null;
        Scanner sc = new Scanner(System.in);

        try {
            // Create a DatagramSocket
            socket = new DatagramSocket();
            InetAddress serverAddress = InetAddress.getByName("localhost");

            byte[] sendBuffer;
            byte[] receiveBuffer = new byte[1024];

            while (true) {
                System.out.print("Enter message to send to server (or 'exit' to quit): ");
                String message = sc.nextLine();
                sendBuffer = message.getBytes();

                // Send the message to the server
                DatagramPacket sendPacket = new DatagramPacket(sendBuffer,
sendBuffer.length, serverAddress, 5000);
                socket.send(sendPacket);

                if ("exit".equalsIgnoreCase(message.trim())) {
                    System.out.println("Client is exiting.");
                    break; // Exit the loop if the user types "exit"
                }

                // Prepare to receive the response from the server
            }
        }
    }
}

```

```

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

        socket.receive(receivePacket); // Blocking call to receive data

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

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

} catch (Exception e) {
    System.err.println("Error: " + e.getMessage());
} finally {
    if (socket != null && !socket.isClosed()) {
        socket.close(); // Close the socket when done
    }
    sc.close();
}
}
}
}

```

UDP Server

```

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class UDPServer {

    public static void main(String[] args) {

        DatagramSocket socket = null;
        try {

```

```
// Create a DatagramSocket to listen on port 5000
socket = new DatagramSocket(5000);
System.out.println("UDP Server is running...");

byte[] receiveBuffer = new byte[1024];

while (true) {
    // Prepare to receive data from client
    DatagramPacket receivePacket = new DatagramPacket(receiveBuffer,
receiveBuffer.length);

    socket.receive(receivePacket); // Blocking call to receive the packet

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

    System.out.println("Data received from client: " + receivedData);

    if ("exit".equalsIgnoreCase(receivedData.trim())) {
        System.out.println("Server shutting down.");
        break; // Exit the loop if the client sends "exit"
    }

    // Sending a response back to client
    String response = "Server received: " + receivedData;
    byte[] sendBuffer = response.getBytes();
    InetAddress clientAddress = receivePacket.getAddress();
    int clientPort = receivePacket.getPort();

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

    socket.send(sendPacket); // Send the response to the client
}
```

```

    }

    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    } finally {
        if (socket != null && !socket.isClosed()) {
            socket.close(); // Close the socket when done
        }
    }
}
}
}

```

ASSIGNMENT 6 (LA 8)

TCP client

```
import java.io.*;
```

```
import java.net.*;
```

```

public class TCPClient {
    public static void main(String[] args) {
        Socket socket = null;
        PrintWriter out = null;
        BufferedReader in = null;

        try {
            // Connect to the server at localhost on port 5000
            socket = new Socket("localhost", 5000);
            System.out.println("Connected to the server.");

```

```

// Setup input and output streams
out = new PrintWriter(socket.getOutputStream(), true);
in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

// Read the greeting from the server
String serverMessage = in.readLine();
System.out.println("Received from server: " + serverMessage);

// Respond back to the server
out.println("Hello from the client!");

} catch (IOException e) {
    e.printStackTrace();
} finally {
    // Close resources
    try {
        if (in != null) in.close();
        if (out != null) out.close();
        if (socket != null) socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}

```

TCP server

```

import java.io.*;
import java.net.*;

```

```
public class TCPServer {  
    public static void main(String[] args) {  
        ServerSocket serverSocket = null;  
        Socket clientSocket = null;  
        PrintWriter out = null;  
        BufferedReader in = null;  
  
        try {  
            // Create a server socket listening on port 5000  
            serverSocket = new ServerSocket(5000);  
            System.out.println("Server is listening on port 5000...");  
  
            // Accept a client connection  
            clientSocket = serverSocket.accept();  
            System.out.println("Client connected.");  
  
            // Setup input and output streams  
            out = new PrintWriter(clientSocket.getOutputStream(), true);  
            in = new BufferedReader(new  
InputStreamReader(clientSocket.getInputStream()));  
  
            // Greet the client  
            out.println("Hello from the server!");  
  
            // Read the message from the client  
            String clientMessage = in.readLine();  
            System.out.println("Received from client: " + clientMessage);  
        }  
    }  
}
```

```

    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        // Close resources
        try {
            if (in != null) in.close();
            if (out != null) out.close();
            if (clientSocket != null) clientSocket.close();
            if (serverSocket != null) serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}

```

ASSIGNMENT 7 (LA 9)

UDP Client

```
import java.io.*;
```

```
import java.net.*;
```

```

public class UDPClient {
    public static void main(String[] args) {
        DatagramSocket socket = null;
        try {
            socket = new DatagramSocket();
            InetAddress serverAddress = InetAddress.getByName("localhost");

            // Change the file name to send different types of files

```



```

        String filePath = "D:\\Saanu\\Academics\\TE\\CN
LAB\\Assignment9\\File.txt"; // Specify the path to your file

        File file = new File(filePath);

        byte[] sendData = new byte[(int) file.length()];

        FileInputStream fis = new FileInputStream(file);
        fis.read(sendData);
        fis.close();

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

        socket.send(sendPacket);

        System.out.println("File sent: " + file.getName());
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (socket != null && !socket.isClosed()) {
            socket.close();
        }
    }
}
}
}

```

UDP Server

```

import java.io.*;
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];

        System.out.println("Server is running...");

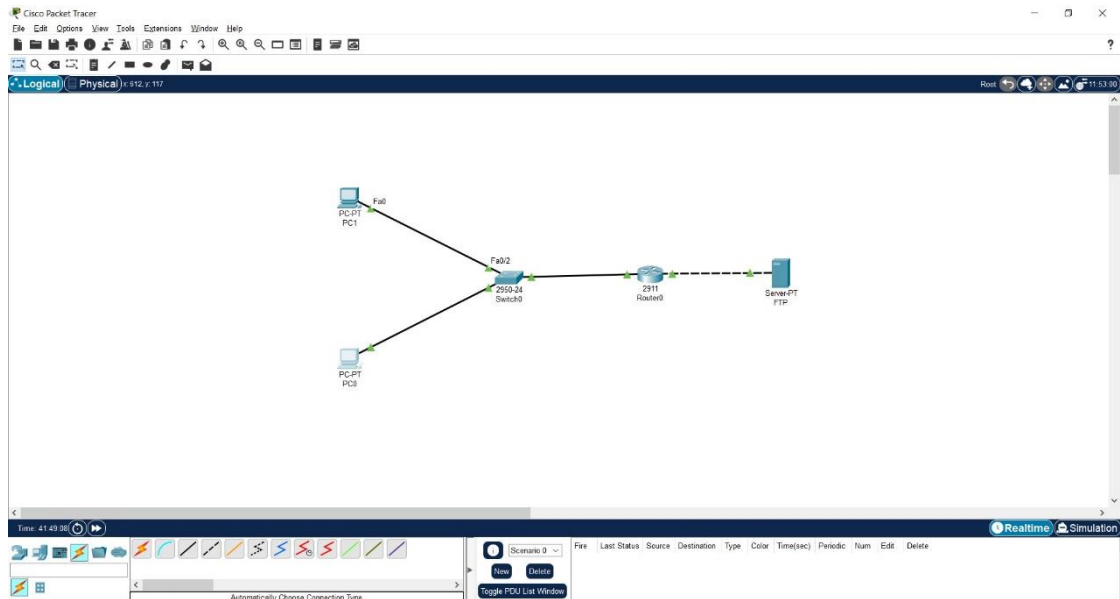
        while (true) {
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
            socket.receive(receivePacket);

            String fileName = "received_file";
            FileOutputStream fos = new FileOutputStream(fileName);
            fos.write(receivePacket.getData(), 0, receivePacket.getLength());
            fos.close();

            System.out.println("File received: " + fileName);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (socket != null && !socket.isClosed()) {
            socket.close();
        }
    }
}

```

ASSIGNMENT 8 (LA 11)



PC0

Physical Config Desktop Programming Attributes

Command Prompt

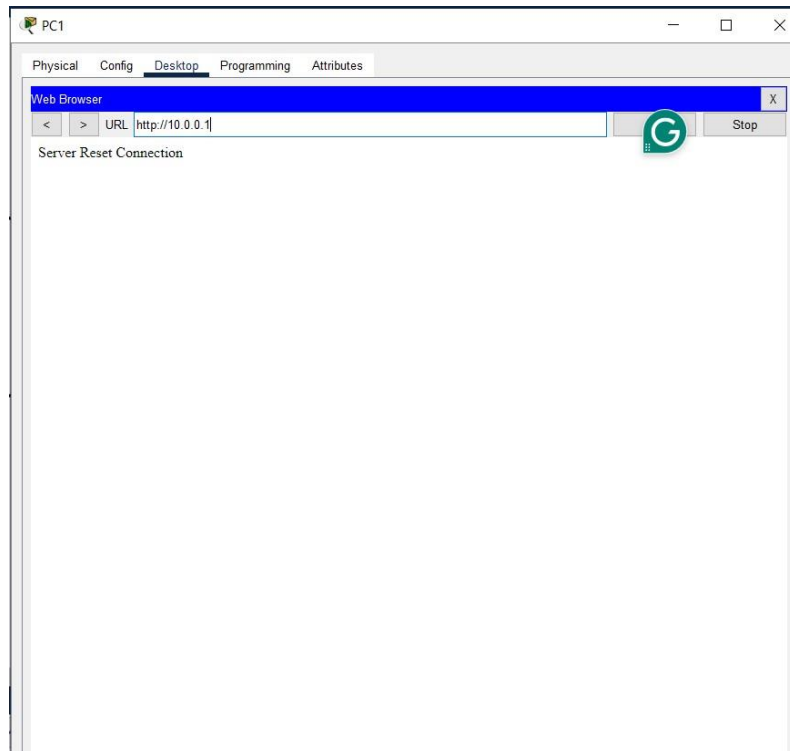
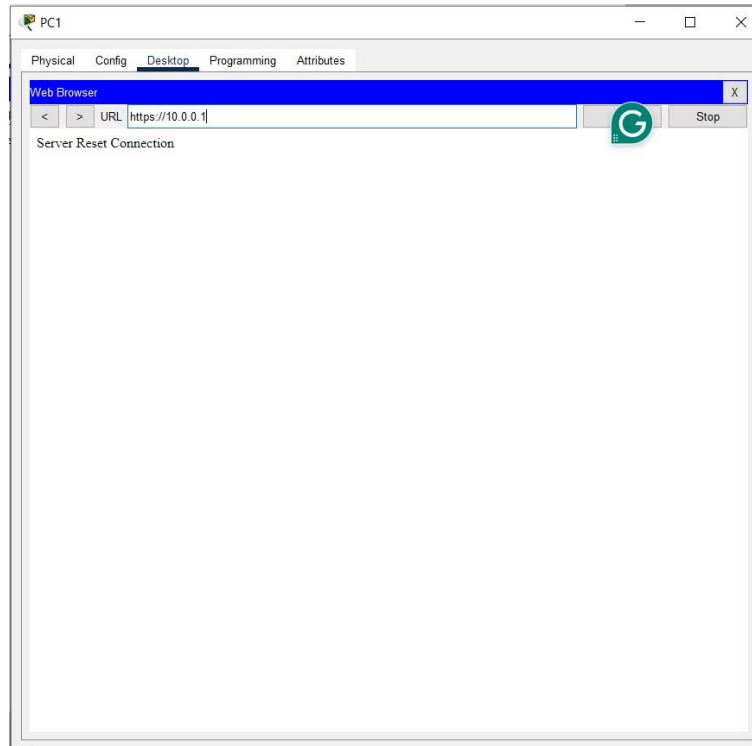
```
Cisco Packet Tracer PC Command Line 1.0
C:\>ftp 10.0.0.2
Trying to connect...10.0.0.2
Connected to 10.0.0.2
220- Welcome to FT Ftp server
Username:cisco
331- Username ok, need password
Password:
230- Logged in
(passive mode On)
ftp>get hello.txt

Reading file hello.txt from 10.0.0.2:
File transfer in progress...

[Transfer complete - 41 bytes]

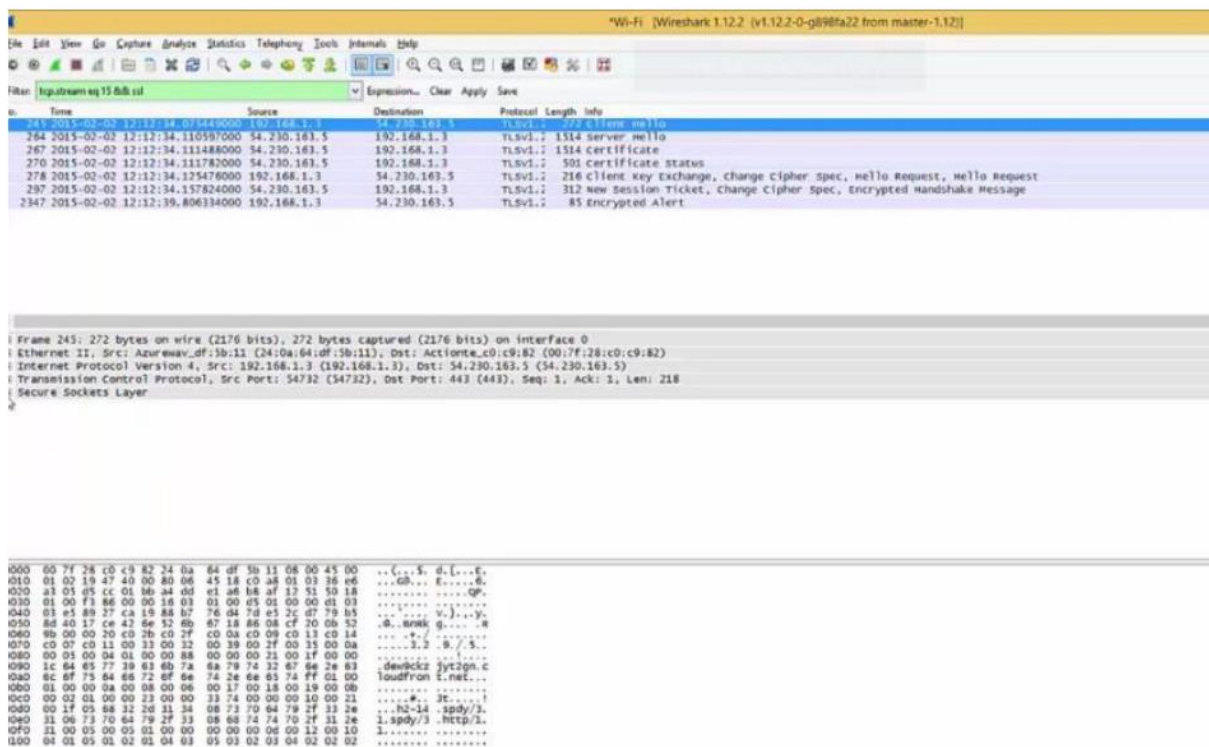
41 bytes copied in 0.01 secs (4100 bytes/sec)
ftp>
```

☐ Top





ASSIGNMENT 9 (LA 12)



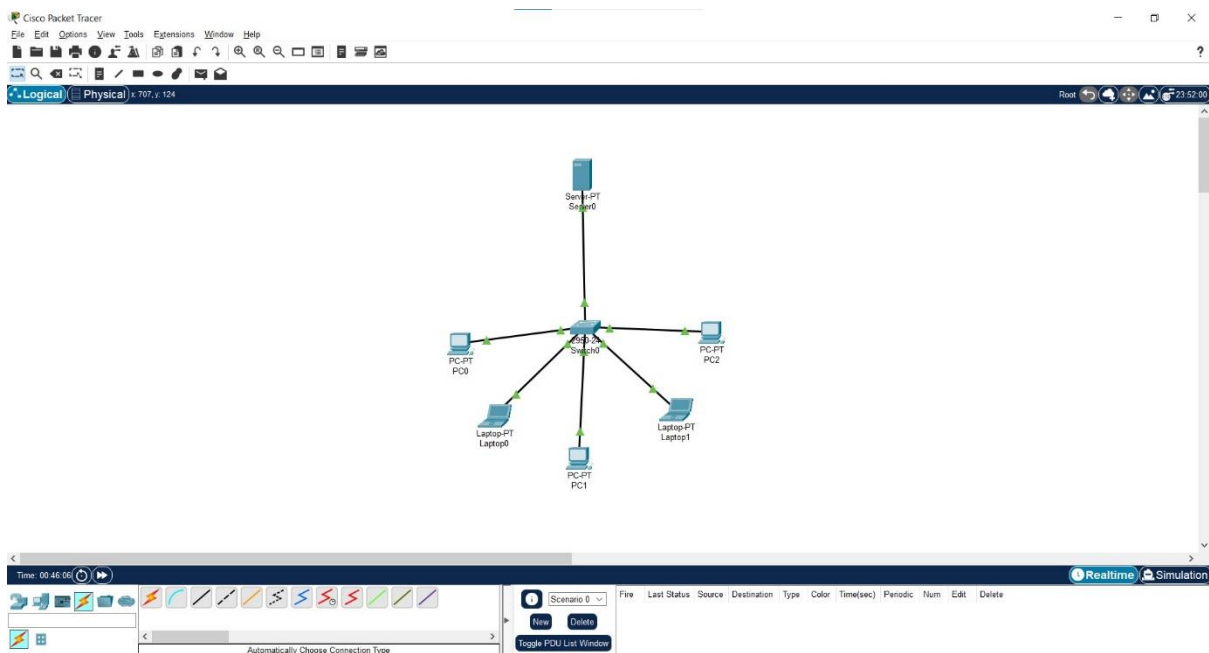
ASSIGNMENT 10 (LA 14)

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-----------|----------------|----------------|----------|--------|--|
| 964 | 26.215745 | 192.168.43.188 | 52.206.144.119 | TCP | 66 | [TCP Retransmission] 52332 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |
| 965 | 26.308351 | 192.168.43.188 | 13.32.251.53 | TCP | 66 | [TCP Retransmission] 52330 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |
| 966 | 26.340328 | 192.168.43.188 | 52.3.147.230 | TCP | 66 | [TCP Retransmission] 52307 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |
| 967 | 26.983453 | 192.168.43.188 | 52.39.16.251 | TCP | 66 | [TCP Retransmission] 52334 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |
| 968 | 27.046847 | 192.168.43.188 | 52.37.61.255 | TCP | 66 | [TCP Retransmission] 52337 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |
| 969 | 27.139266 | 192.168.43.188 | 20.42.65.90 | TCP | 66 | [TCP Retransmission] 52335 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM |
| 970 | 27.186809 | 192.168.43.188 | 18.165.189.87 | TCP | 66 | [TCP Retransmission] 52336 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |
| 971 | 27.216579 | 192.168.43.188 | 52.200.112.105 | TCP | 66 | [TCP Retransmission] 52290 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |
| 972 | 27.210318 | 192.168.43.188 | 34.197.204.218 | TCP | 66 | [TCP Retransmission] 52278 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |
| 973 | 27.342224 | 192.168.43.188 | 52.3.147.230 | TCP | 66 | [TCP Retransmission] 52307 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |
| 974 | 27.359276 | 192.168.43.188 | 13.32.251.66 | TCP | 66 | [TCP Retransmission] 52338 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |
| 975 | 27.648331 | 192.168.43.188 | 192.168.43.1 | DNS | 96 | Standard query 0x562a A win-extension.femetrics.grammarly.io |
| 976 | 27.649368 | 192.168.43.188 | 192.168.43.1 | DNS | 96 | Standard query response 0x56a0 AAAA win-extension.femetrics.grammarly.io |
| 977 | 27.657137 | 192.168.43.1 | 192.168.43.188 | DNS | 96 | Standard query response 0x562a A win-extension.femetrics.grammarly.io A 52.200.112.105 A 44.215.95.64 A 34.200.200.233 A 52.206. |
| 978 | 27.749187 | 192.168.43.1 | 192.168.43.188 | DNS | 224 | Standard query response 0x562a A win-extension.femetrics.grammarly.io A 52.200.112.105 A 44.215.95.64 A 34.200.200.233 A 52.206. |
| 979 | 27.751334 | 192.168.43.188 | 52.200.112.105 | TCP | 66 | [TCP Retransmission] 52339 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |
| 980 | 28.062161 | 192.168.43.188 | 52.37.61.255 | TCP | 66 | [TCP Retransmission] 52337 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |

> Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{504A5066-BE-0000-0000-0000-000000000000} on 0:00:00:00:00:00
> Ethernet II, Src: Intel_b0:99:88 (28:16:ad:b0:99:88), Dst: d6:15:57:99:ab:d5 (d6:15:57:99:ab:d5)
> Internet Protocol Version 4, Src: 192.168.43.188, Dst: 44.236.110.42
> Transmission Control Protocol, Src Port: 52294, Dst Port: 443, Seq: 0, Len: 0

0000 d6 15 57 99 ab d5 28 16 ad b0 99 88 00 00 45 00 --W...(:E:
0010 00 34 3c d6 40 00 00 06 36 6c c0 a8 2b bc 2c ec -4C @ - - 6L + + + +
0020 6e 2a cc 46 01 bb 83 60 63 4e 00 00 00 00 02 m+P . . . cH
0030 fa f0 37 f4 00 00 02 04 05 b4 01 03 03 08 01 01 --7.....
0040 04 02 ..

ASSIGNMENT 11 (LA 15)



ASSIGNMENT 12 (LA 16)

```
import java.net.InetAddress;
```

```
import java.util.Scanner;
```

```
public class DNS
```

```
{
```

```
    public static void main(String[] args) throws Exception {
```

```
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.println("1. IP to URL\n2. URL to IP");
```

```
        int choice = scanner.nextInt();
```

```
        scanner.nextLine();
```

```
        switch (choice) {
```

```
            case 1:
```

```
                System.out.println("Enter the IP address: ");
```

```
                String ip = scanner.nextLine();
```

```
                InetAddress inetAddressIP = InetAddress.getByName(ip);
```

```
                System.out.println("The URL associated with the IP address " + ip +  
" is: " + inetAddressIP.getHostName());
```

```
                break;
```

```
            case 2:
```

```
                System.out.println("Enter the URL: ");
```

```
                String url = scanner.nextLine();
```

```
                InetAddress inetAddressURL = InetAddress.getByName(url);
```

```
                System.out.println("The IP address associated with the URL " + url +  
" is: " + inetAddressURL.getHostAddress());
```

```
break;
```

```
default:
```

```
    System.out.println("Invalid choice! Please enter '1' or '2'.");
```

```
}
```

```
scanner.close();
```

```
}
```

```
}
```