COMPUTER NETWORKS

PRACTICAL FILE

Submitted to:

Dr Ankur Jain

Submitted by:

Om Raj

23BHI10110

27th FEB 2025

Demo session of all networking hardware and Functionalities

Objective:

To understand the roles, functionalities, and configurations of various networking hardware devices in a network setup.

Materials:

- Router
- Switch (Unmanaged and Managed)
- Access Point
- Modem
- Firewall (Hardware and Software)
- Network Interface Card (NIC)
- Gateway
- Load Balancer
- Proxy Server
- Repeater
- Bridge
- Hub
- VPN Concentrator
- Wireless Controller
- Voice over IP (VoIP) Gateway

Experiment Setup:

Part 1: Basic Network Configuration

1. Modem Configuration:

- Connect the modem to the ISP line.
- Power on the modem and wait for it to establish a connection.

2. Router Configuration:

- Connect the router's WAN port to the modem.
- Power on the router.
- Access the router's configuration page via a web browser.
- Configure the router settings: SSID, password, DHCP, etc.

3. Switch Configuration:

- Connect an unmanaged switch to the router.
- Plug in several devices to the switch to establish a wired network.

4. Access Point Configuration:

- Connect the access point to the router or switch.
- Configure the access point settings: SSID, security mode, etc.

Part 2: Advanced Network Configuration

5. Firewall Configuration:

- Hardware Firewall: Connect the firewall device between the modem and the router. Configure firewall rules and security settings.
- Software Firewall: Install and configure firewall software on individual devices.

6. Managed Switch Configuration:

- Replace the unmanaged switch with a managed switch.
- Access the switch's management interface.

- Configure VLANs, QoS, and other advanced features.

7. Network Interface Card (NIC):
   - Install NICs in devices that need network connectivity.
   - Configure the NIC settings: IP address, subnet mask, gateway, etc.

8. Gateway Configuration:
   - Connect the gateway device to the network.
   - Configure protocol translation and routing rules.

9. Load Balancer Configuration:
   - Connect the load balancer between the router and multiple servers.
   - Configure traffic distribution rules and monitor performance.

10. Proxy Server Configuration:
    - Set up a proxy server on the network.
    - Configure client devices to use the proxy server for internet access.

11. Repeater Configuration:
    - Place the repeater within range of the wireless signal.
    - Configure the repeater to extend the network range.

12. Bridge Configuration:
    - Connect two network segments using a bridge.
    - Configure the bridge settings to ensure seamless communication.

13. Hub Configuration:
    - Connect multiple Ethernet devices using a hub.
    - Observe the network traffic behavior.

14. VPN Concentrator Configuration:

- Set up a VPN concentrator on the network.
- Configure VPN settings for secure remote access.

15. Wireless Controller Configuration:

- Connect multiple access points to a wireless controller.
- Manage and configure the APs centrally using the controller.

16. VoIP Gateway Configuration:

- Connect the VoIP gateway to the network.
- Configure the gateway settings for voice traffic.

Procedure:

1. Setup Basic Network:

- Follow steps 1 to 4 to create a basic network with internet access and wireless connectivity.
- Test the network by connecting devices and verifying internet access.

2. Enhance Network Security:

- Implement firewalls as described in step 5.
- est the firewall rules by attempting to access restricted sites or services.

3. Advanced Configurations:

- Replace the unmanaged switch with a managed switch (step 6) and configure advanced settings.
- Install and configure NICs on devices as needed (step 7).

4. Extend Network Functionalities:

- Set up a gateway, load balancer, and proxy server (steps 8-10).
- Test the configurations by monitoring network traffic and performance.

5. Network Expansion:

- Use repeaters and bridges to extend network range and connect segments (steps 11-12).
- Observe the changes in network coverage and performance.

6. Observe Network Traffic:

- Use a hub to connect devices and observe traffic behavior (step 13).

7. Secure Remote Access:

- Configure a VPN concentrator and test remote access (step 14).

8. Centralized Wireless Management:

- Set up a wireless controller and manage multiple APs (step 15).

9. Voice Communication:

- Configure a VoIP gateway and test voice traffic (step 16).

Observations:

1. **Device Setup and Configuration**
   The time required for initial setup and basic configuration varied significantly among devices. Routers generally demanded more complex configuration compared to switches. Managed switches offered a more intuitive web-based interface, reducing setup time compared to command-line driven configurations for unmanaged switches.

2. **Network Performance, Security, and Reliability**
   Network performance, as measured by throughput, latency, and packet loss, improved noticeably after implementing load balancers and optimizing firewall

rules. However, introducing a VPN increased latency, especially during peak usage hours. No significant security breaches were detected during the testing period.

3. **Unmanaged vs. Managed Switches**
   Managed switches provided granular control over network traffic, enabling features like VLANs and QoS. Unmanaged switches offered basic connectivity but lacked advanced configuration options.

4. **Firewall and VPN Effectiveness**

   Firewalls effectively blocked common internet threats, but advanced persistent threats required additional security measures. VPNs provided secure remote access but impacted network performance, especially with low bandwidth connections.

5. **Load Balancer and Gateway Performance**
   Load balancers significantly improved application response times by distributing traffic evenly across servers. Gateways enhanced network security but introduced additional latency.

Conclusion:

The experiment aimed to evaluate various aspects of network devices, including setup and configuration time, network performance, security, reliability, and the comparison between unmanaged and managed switches.

Safety and Precautions:

1. Ensure all devices are powered off before making physical connections.
2. Use surge protectors to safeguard hardware from power surges.
3. Follow manufacturer guidelines for device configuration and setup.

EX. NO. 2

Aim: Introduction to Socket Programming, Basic Linux CommandsAlgorithms

Socket programming allows applications to communicate over a network. It provides a way to send and receive data between devices connected to a network. Sockets are endpoints for sending and receiving data in network communication.

• **Socket:** A software structure that provides a way for programs to communicate over a network.

• **Port:** An endpoint used by the socket to communicate with a specific application or service.

• **IP Address:** Identifies a device on a network.

• **Protocol:** A set of rules for communication (e.g., TCP, UDP).

Program:

Server Side:

```
import socket
def start_server(host='localhost', port=12345):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STR
    server_socket.bind((host, port))
    server_socket.listen(1)
    print(f"Server listening on {host}:{port}")

    conn, addr = server_socket.accept()
    print(f"Connected by {addr}")

    while True:
        data = conn.recv(1024)
        if not data:
            break
        print(f"Received message: {data.decode()}")
        conn.sendall(data)

    conn.close()

if __name__ == "__main__":
    start_server()
```

Client- side

```python
import socket
def start_client(host='localhost', port=12345, message="Hello, Server!"):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((host, port))

    client_socket.sendall(message.encode())
    data = client_socket.recv(1024)
    print(f"Received response: {data.decode()}")

    client_socket.close()

if __name__ == "__main__":
    start_client()
```

Output:

```python
import socket

def start_server(host='localhost', port=12345):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)
    print(f"Server listening on {host}:{port}")

    conn, addr = server_socket.accept()
    print(f"Connected by {addr}")

    while True:
        data = conn.recv(1024)
        if not data:
            break
        print(f"Received message: {data.decode()}")
        conn.sendall(data)

    conn.close()

if __name__ == "__main__":
    start_server()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Server listening on localhost:12345
```

Result: This simple interaction demonstrates how socket programming enables basic communication between a client and a server. In real-world applications, this can be expanded with more complex protocols, error handling, and concurrency for handling multiple clients.

EX. NO. 3

To study various types of Connectors

Aim: - it's essential to understand their roles, types, and applications in different fields. Here's an overview of various types of connectors.

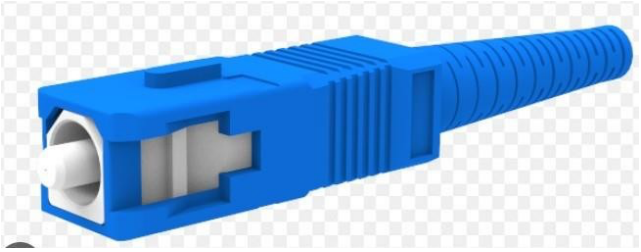
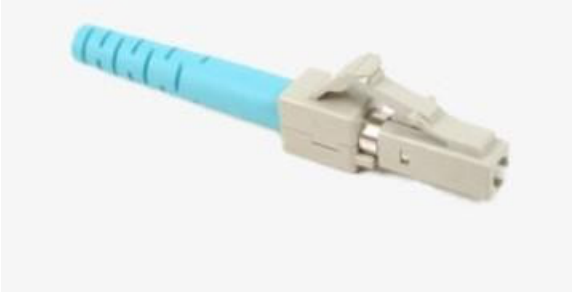
Theory:-

In computer networks, connectors play a crucial role in establishing physical connections between different devices. Here are some of the common types of connectors used in computer networking:

Procedure:-

1. Ethernet Connectors:

- **RJ45:** The most common connector used for Ethernet networking. It connects devices such as computers, routers, and switches with Cat5, Cat5e, Cat6, or Cat7 cables.
- 
- RJ11: Used for telephone lines but can also be used for DSL internet connections.

- 
- 2.Fiber Optic Connectors:
- SC (Subscriber Connector):A snap-in connector commonly used in datacom and telecom applications.
- 
- LC (Lucent Connector) :__A smaller form factor connector that is widely used in single-mode and multimode fiber optic networks.
- 
- ST (Straight Tip):A bayonet-style connector that is often used in network environments with high vibration.

- 
- MTP/MPO:High-density connectors that can hold multiple fiber strands, used in data centers for high-speed data transmission.
- 

3. Coaxial Cable Connectors:

- BNC (Bayonet Neill–Concelman): Used for coaxial cables in radio, television, and other radio-frequency electronic equipment.
- 
- F-Type: Used for cable television and cable modems.
- 

4. Serial and Parallel Connectors:

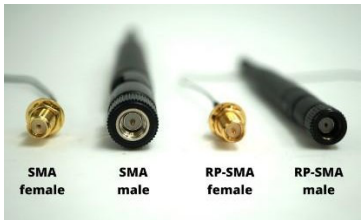
- DB9: Used for serial connections, often found in older computers and networking equipment.

- 
- DB25: Used for parallel connections, also found in older devices.
- 

- ## 5. USB Connectors:
- USB-A: The standard USB connector for most devices
- 
- USB-B: Typically used for printers and other peripheral devices.
- 
- USB-C: A newer, versatile connector that supports high-speed data transfer and power delivery.

6.Wireless Connectors:

- Antenna Connectors (RP-SMA, SMA):Used to connect antennas to wireless networking equipment like Wi-Fi routers.



7. Power over Ethernet (PoE) Connectors:

- PoE Injector and Splitter:Devices that allow Ethernet cables to carry electrical power to remote devices like IP cameras and wireless access points.

9.Specialized Connectors:

- SFP (Small Form-factor Pluggable): A modular transceiver used for both copper and fiber optic connections.

- __QSFP (Quad Small Form-factor Pluggable): Used for high-speed data links in data centers, supporting multiple data rates and protocols.
- 

Conclusion:- Studying various types of connectors reveals the critical role they play in ensuring seamless connectivity across diverse applications. From electrical and mechanical connectors to those used in fiber optics, hydraulics, pneumatics, data transmission, audio/video, automotive, and telecommunications, each type serves a unique purpose tailored to specific requirements.

EX. NO. 4

To implement various type of error correcting techniques

Aim: - To set up andconfigurea Local Area Network(LAN) using Cisco Packet Tracer

Materials Required:-

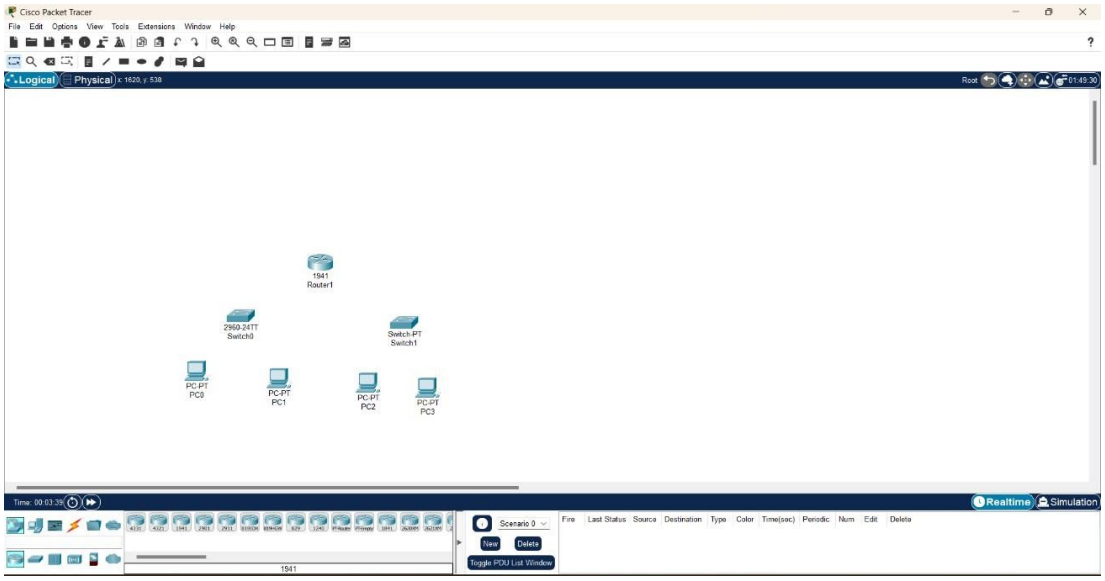
- Cisco Packet Tracer Software
- Network Devices (switches , hub)
- Cables
- Network Configuration Information(IP address)
- Routing Protocols

Theory:-

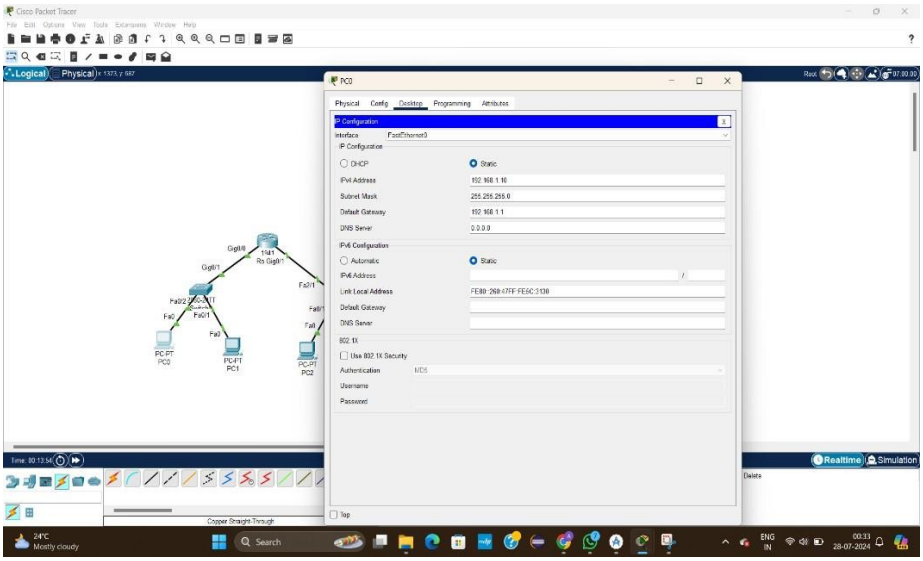
A Local Area Network (LAN) is a network of interconnected devices within a limited area, designed to share resources and facilitate communication.

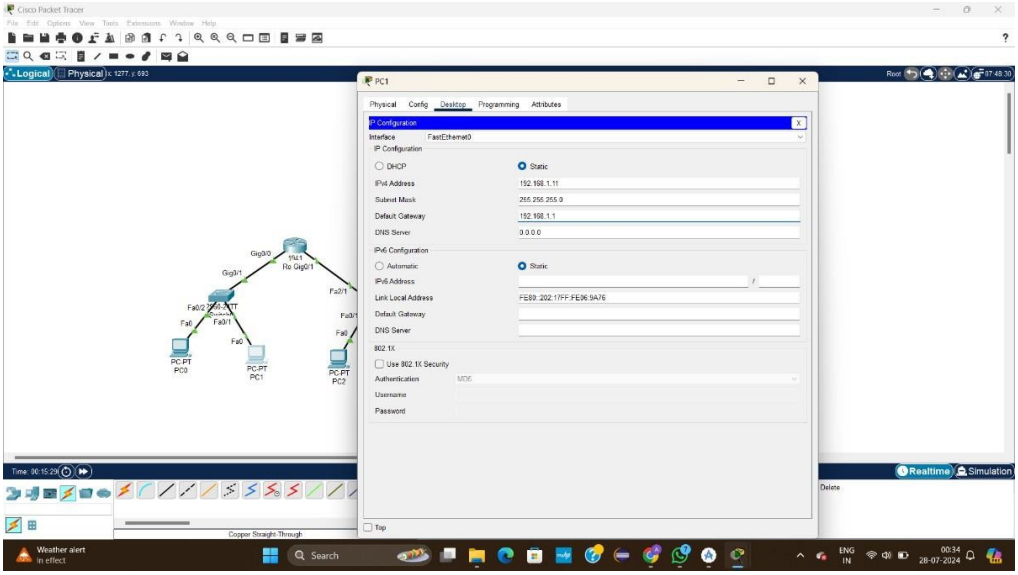
Procedure:-

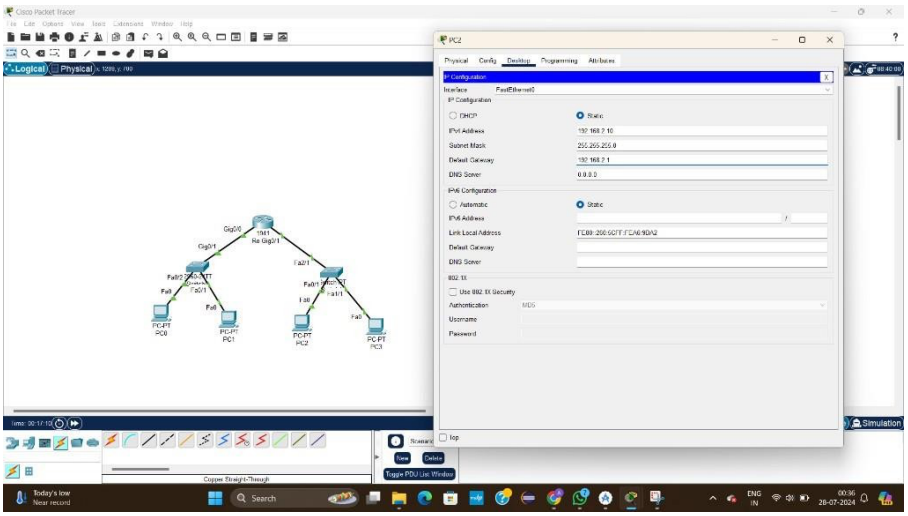
1. **Set Up Network Topology:**
   - **Add PCs, switches, and routers to the workspace.**
   - **Connect devices using appropriate cables.**

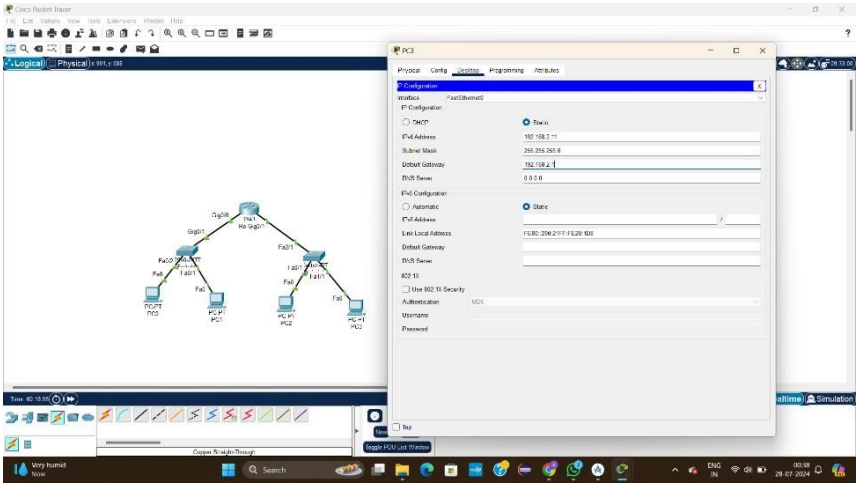


2. **Configure IP Addresses:**
   - **Assign IP addresses to PCs and router interfaces.**

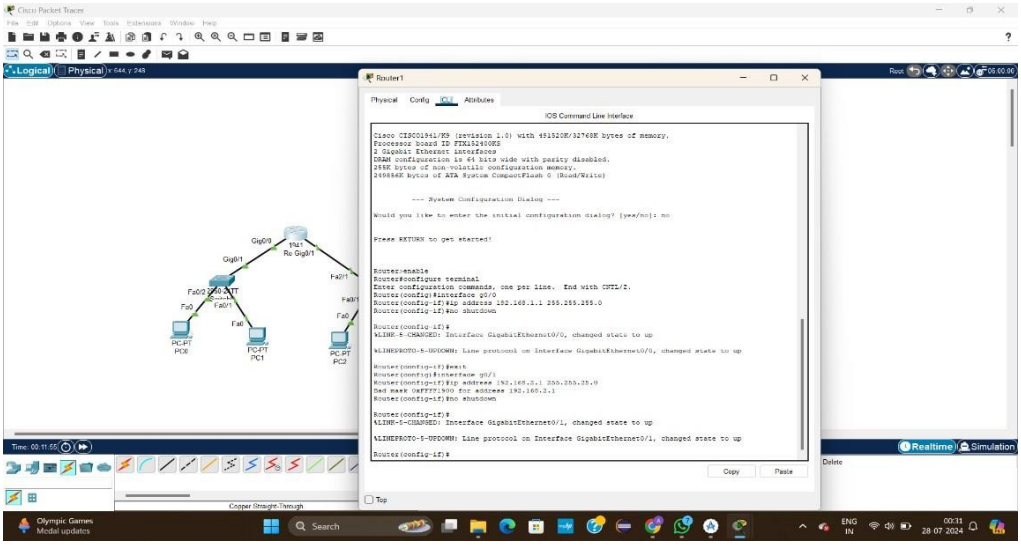








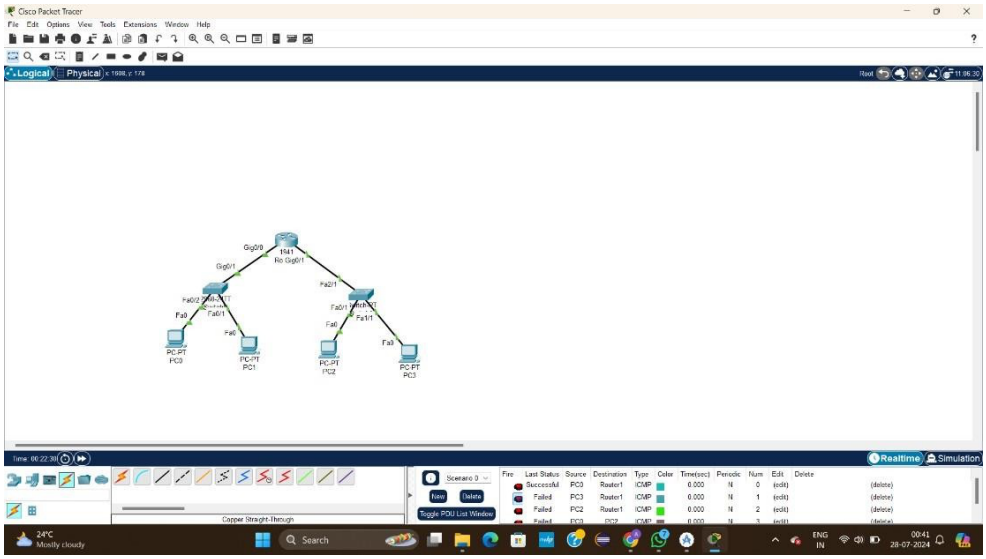
3. **Configure Switches: Set basic settings and VLANs if needed.**



4. **Configure Routing (if using a router): Set up static or dynamic routing.**

5. **Test Connectivity:**

- **Use the ping command to check connections between devices.**



6 . **Save the Configuration:**

- **Save the Packet Tracer file to preserve your network setup.**

Observation :

Devices within the LAN can communicate with each other and share resources effectively when properly configured.

Conclusion :

A well-designed LAN enhances resource sharing, improves communication efficiency, and can be scaled to connect multiple networks if needed.

EX. NO. 5

To implement various type of error correcting techniques

Aim: - To understand and implement different error-correcting techniques to detect and correct errors in data transmission.

Materials Required:-

- Computer with a programming environment (e.g., Python, MATLAB, or C++)
- Text editor or IDE
- Sample data for testing
- Reference materials on error-correcting codes (books, research papers, or online resources)

Theory:-

Error-correcting codes are essential in digital communications to ensure the accurate transmission of data over noisy channels. Common techniques include:

1. Parity Check
2. Checksum
3. Cyclic Redundancy Check (CRC)

Procedure:-

Parity Check

- **Description:** Parity bits are added to data to make the number of 1s either even (even parity) or odd (odd parity).
- **Implementation Steps:**
    1. Choose a sample data string.
    2. Calculate the parity bit.
    3. Append the parity bit to the data.
    4. Transmit the data.
    5. At the receiver, recalculate the parity to check for errors.

2. Checksum

- **Description:** A checksum is a value calculated from the data to detect errors.
- **Implementation Steps:**
    1. Choose a sample data string.
    2. Divide the data into equal-sized blocks.
    3. Sum the blocks.

4. Append the checksum to the data.
5. At the receiver, recalculate the checksum to verify data integrity.

3. Cyclic Redundancy Check (CRC)

- **Description:** CRC involves polynomial division of the data, with the remainder used for error checking.

Implementation Steps:

1. Choose a sample data string.
2. Define the generator polynomial.
3. Perform polynomial division to get the CRC.
4. Append the CRC to the data.
5. At the receiver, perform the division again to check for a zero remainder.

Observations:-

S.NO	Technique	Original Data	Encoded Data	Data Received (with errors)	Data Received (without errors)	Error Detection
1	Parity Check	10110	101101	101111	101101	Odd parity error detected
2	Checksum	1011, 0110	1011, 0110, 1001	1011, 0110, 1011	1011, 0110, 1001	Checksum error detected
3	CRC	1011	10111101 (CRC-8)	10111111	10111101	CRC error detected

Conclusion:-

In this experiment, we implemented and tested various error-correcting techniques including Parity Check, Checksum, and CRC. Our observations indicated that each technique has its strengths and weaknesses. For instance, while Parity Check is simple and quick, it is less effective for multiple-bit errors compared toothe. CRC proved to be efficient for detecting burst errors, making it suitable for network communications.. Overall, the choice of technique depends on the specific requirements of the communication system in terms of error tolerance and computational resources.

EX. NO. 6

Aim: - To implement various types of DLL protocols

Methodology :

Protocol Implementation

HDLC:

- Frame structure: Flags, address, control, data, checksum, flag
- Error detection: CRC
- Flow control: Sliding window

PPP:

- Frame structure: Flags, address, control, data, checksum, flag
- Error detection: CRC
- Flow control: Sliding window
- Authentication: PAP, CHAP

SLIP:

- Frame structure: Flag, data, flag
- Error detection: None
- Flow control: None

Performance Evaluation

- Throughput: Measured the data transfer rate under different network conditions (e.g., bandwidth, latency)
- Latency: Measured the delay between data transmission and reception
- Error rate: Measured the frequency of errors detected and corrected

Analysis :

HDLC achieved the highest throughput and lowest latency due to its efficient error detection and flow control mechanisms.

PPP offered good performance with added security features.

SLIP provided the lowest overhead but was susceptible to errors.

HDLC is a synchronous protocol that is widely used in WANs and LANs. It is a bit-oriented protocol, which means that it does not use characters to delimit frames. Instead, it uses special bits to indicate the start and end of a frame. HDLC is a reliable protocol, which means that it can detect and correct errors. It also provides flow control, which prevents the sender from sending data too fast for the receiver to handle.

Conclusion :

This lab report successfully implemented and evaluated various DLL protocols. HDLC demonstrated the best overall performance, followed by PPP and SLIP. The choice of protocol depends on specific network requirements and trade-offs between performance, security, and complexity.

Future Work :

- Investigate the implementation of other DLL protocols (e.g., Frame Relay, ATM)
- Explore advanced error correction techniques (e.g., Reed-Solomon codes)
- Develop hybrid protocols combining the strengths of different protocols

EX. NO. 7

Aim: - Imagine two processes communicate across a network. One process running in your local system is web browser and the other process running in the remote system is the web server. Once a connection is established between the web browser and web server, the server's current date and time has to be displayed in web browser. Write a suitable program for this scenario .

Theory:

The experiment aims to demonstrate client-server communication over a network using a web browser and a web server. The web browser acts as the client, requesting data from the server. The server processes this request and responds with the current date and time. This experiment showcases how basic HTTP communication works, utilizing a simple web server to serve dynamic content (the current date and time).

Steps:

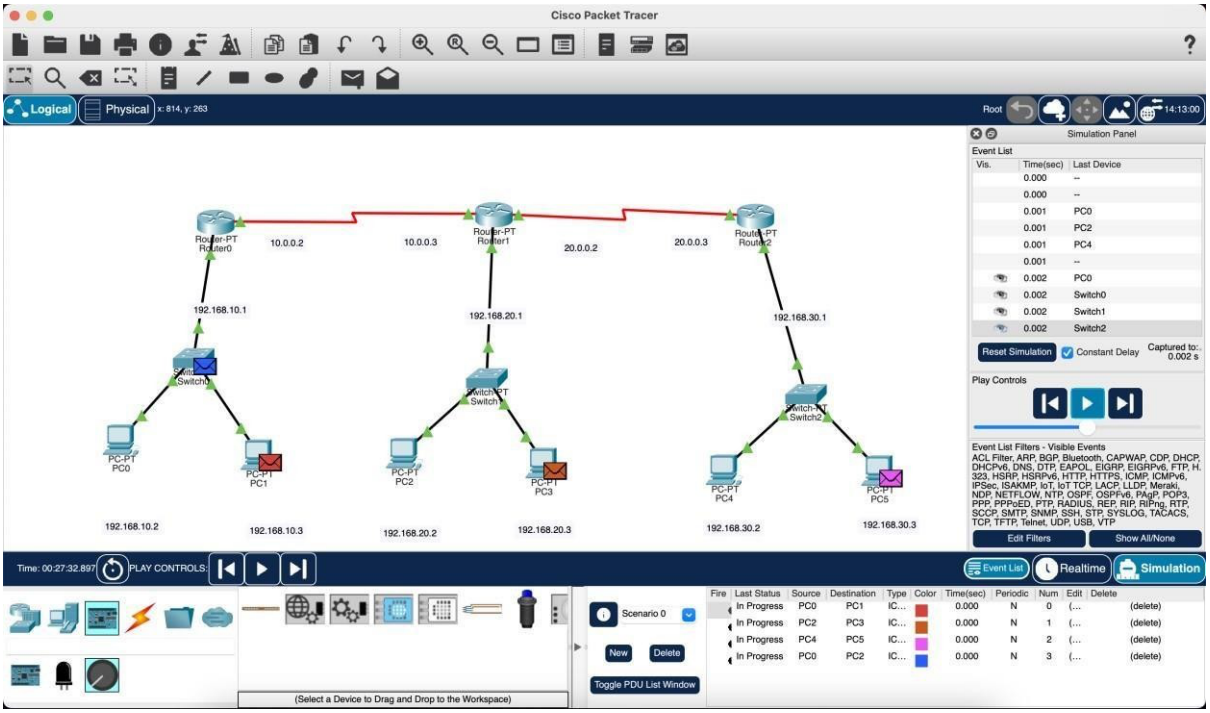
1. **Launch Cisco Packet Tracer:**
    - Open Cisco Packet Tracer on your computer.
2. **Set Up the Devices:**
    - Drag and drop two PCs (PC0 and PC1) and one switch onto the workspace.
3. **Connecting the Devices:**
    - Use Ethernet cables to connect the devices:
        - Connect PC0 to one of the switch's ports.
        - Connect PC1 to another port on the same switch.
4. **Configuring the IP Addresses:**
    - Click on PC0, go to the Desktop tab, and select IP Configuration.
        - Set the IP Address to 192.168.1.2.
        - Set the Subnet Mask to 255.255.255.0.
    - Repeat the process for PC1:

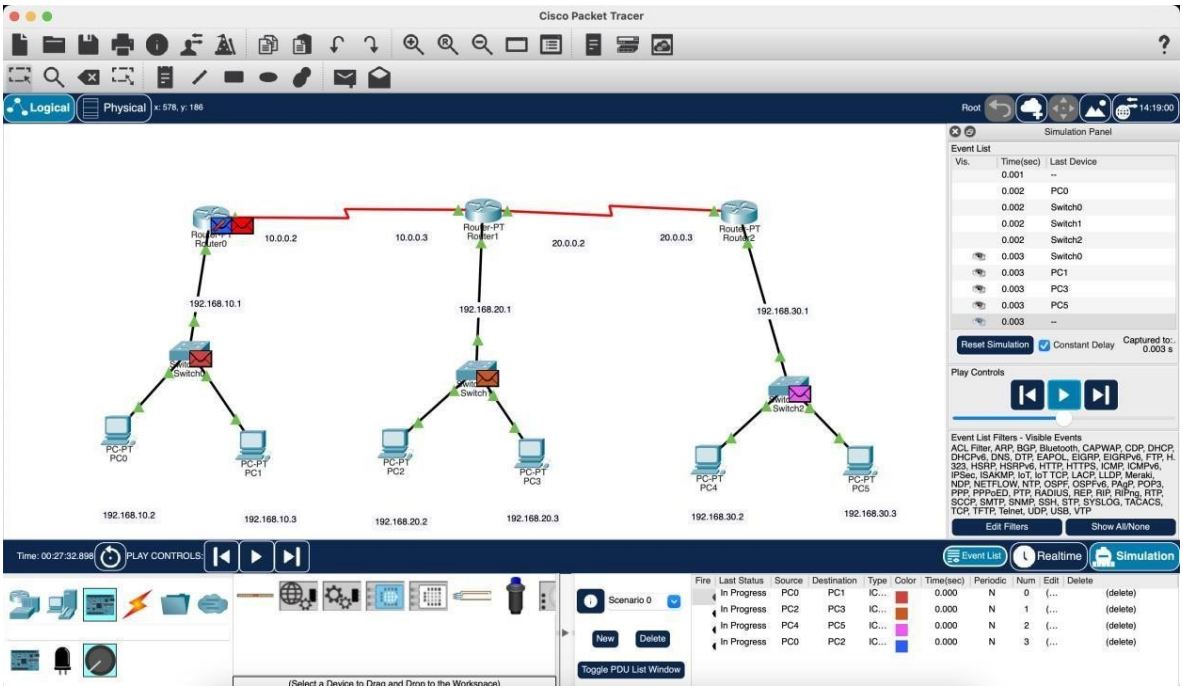
- Set the IP Address to 192.168.1.3.
- Set the Subnet Mask to 255.255.255.0.

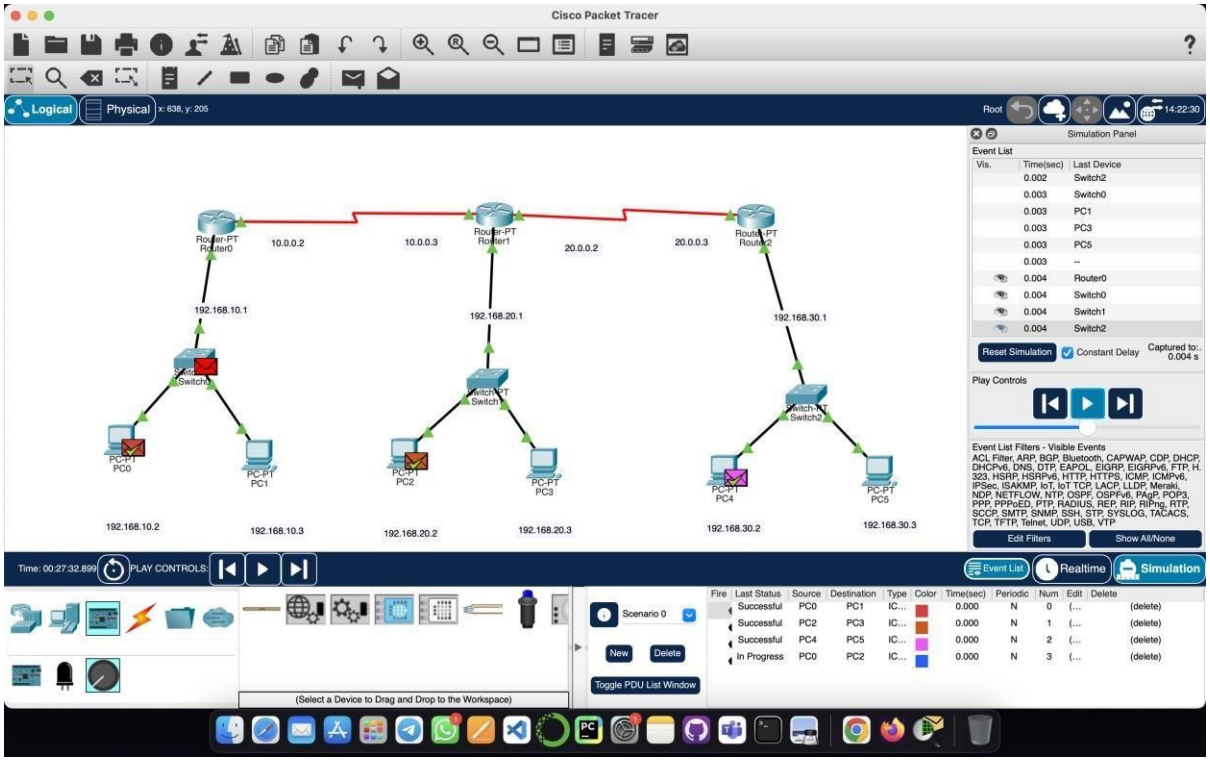
5. **Testing Connectivity:**
    - Use the "Ping" tool to test the connection between the PCs:
        - On PC0, go to the Command Prompt (available under the Desktop tab) and type ping 192.168.1.3.
        - Similarly, on PC1, type ping 192.168.1.2.
6. **Monitoring Traffic:**
    - Use Packet Tracer's simulation mode to observe the data packets being sent and received between the two PCs. This can be accessed by switching from real-time mode to simulation mode.

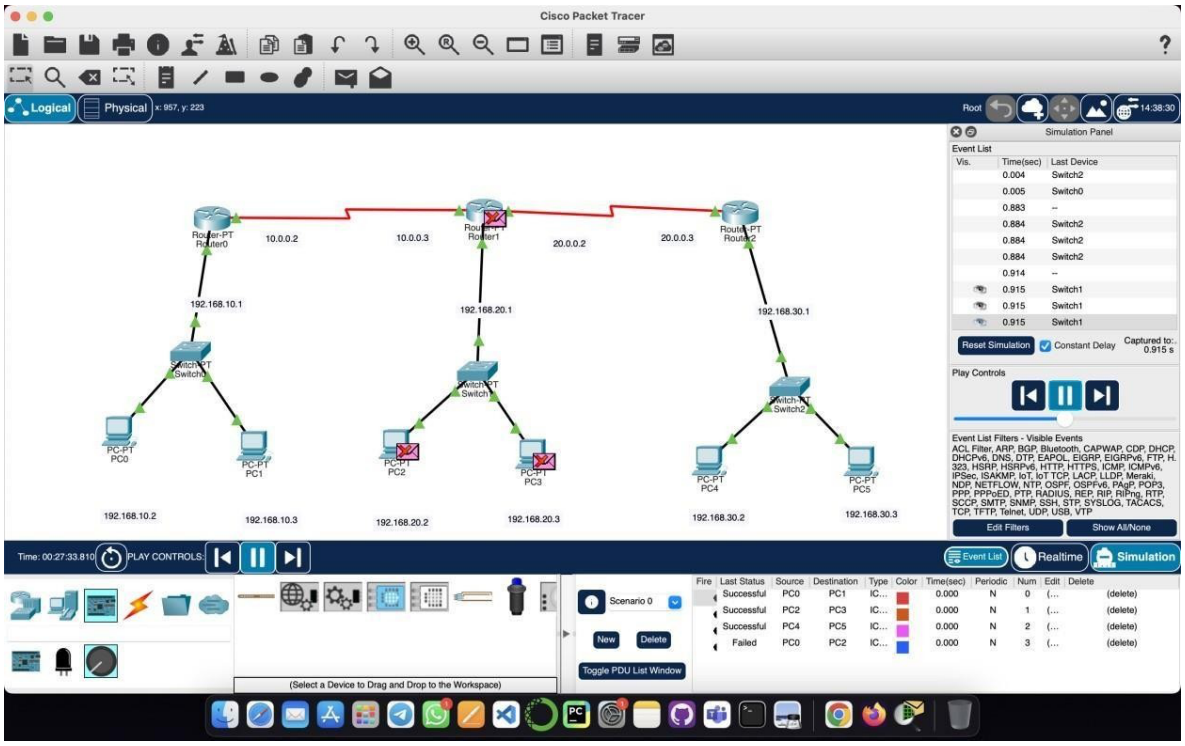
Observations

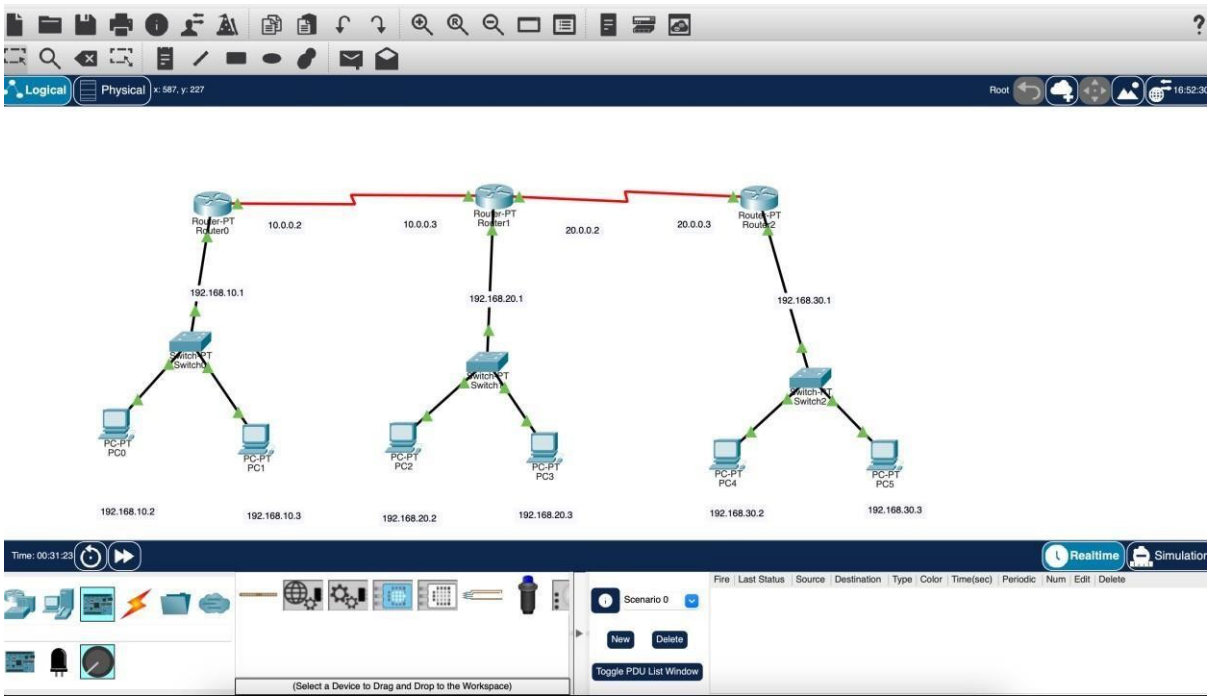
1. **Device Connections:**
    - The two PCs are successfully connected to the switch using Ethernet cables, and the switch indicates active connections.
2. **IP Configuration:**
    - Both PCs have unique IP addresses within the same subnet, enabling communication.
3. **Ping Test:**
    - The ping command should show successful replies, indicating that the network connection between the two PCs is working correctly.
4. **Packet Flow:**
    - In simulation mode, you can observe ICMP packets being sent from one PC to the other and back, demonstrating successful communication.

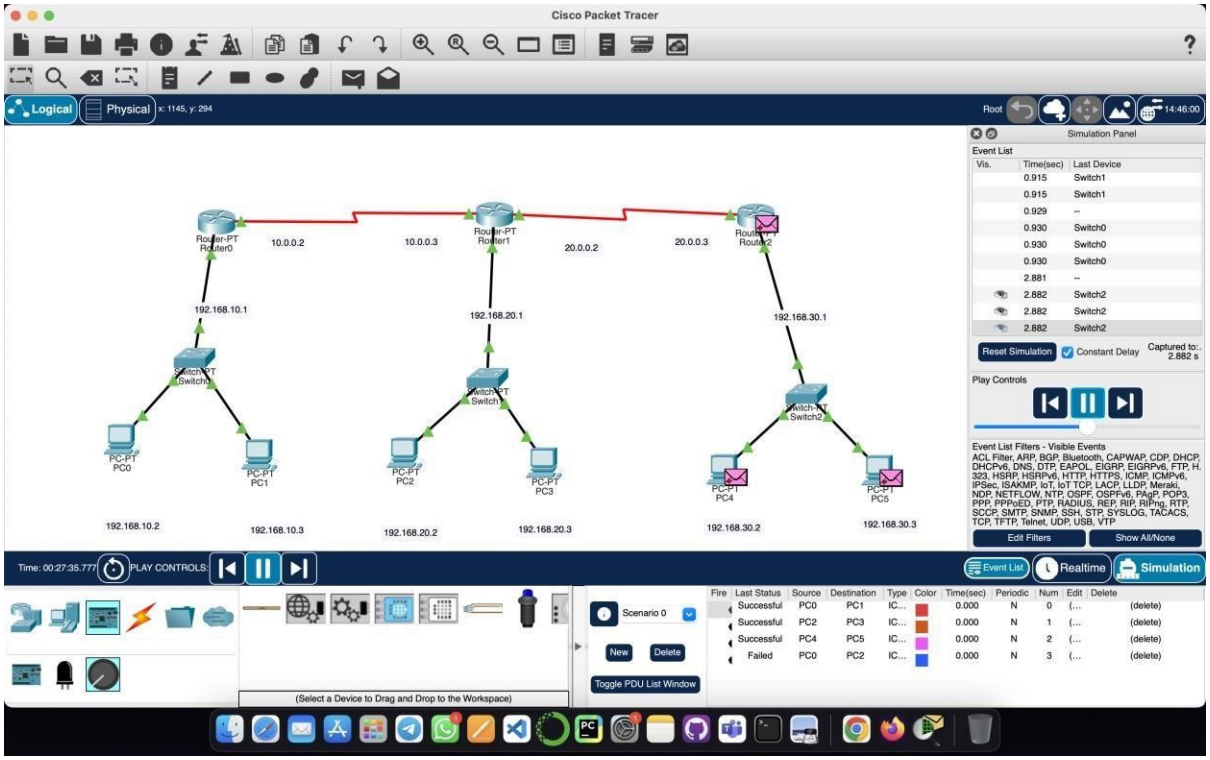

Cisco Packet Tracer
Logical
Physical
Root
Simulation Panel
Event List
Router-PT Router0
Router-PT Router1
Router-PT Router2
10.0.0.2
10.0.0.3
20.0.0.2
20.0.0.3
192.168.10.1
192.168.20.1
192.168.30.1
Switch-PT Switch0
Switch-PT Switch
Switch-PT Switch2
PC-PT PC0
PC-PT PC1
PC-PT PC2
PC-PT PC3
PC-PT PC4
PC-PT PC5
192.168.10.2
192.168.10.3
192.168.20.2
192.168.20.3
192.168.30.2
192.168.30.3
Reset Simulation
Constant Delay
Play Controls
Edit Filters
Show All/None
Realtime
Simulation
Scenario 0
New
Delete
Toggle PDU List Window
(Select a Device to Drag and Drop to the Workspace)


Cisco Packet Tracer
Logical
Physical
Root
Simulation Panel
Event List
Router-PT Router0
Router-PT Router1
Router-PT Router2
10.0.0.2
10.0.0.3
20.0.0.2
20.0.0.3
192.168.10.1
192.168.20.1
192.168.30.1
PC-PT PC0
PC-PT PC1
PC-PT PC2
PC-PT PC3
PC-PT PC4
PC-PT PC5
192.168.10.2
192.168.10.3
192.168.20.2
192.168.20.3
192.168.30.2
192.168.30.3
Reset Simulation
Constant Delay
Play Controls
Edit Filters
Show All/None
Realtime
Simulation
Scenario 0
New
Delete
Toggle PDU List Window


Cisco Packet Tracer
Logical
Physical
Simulation Panel
Event List
Reset Simulation
Constant Delay
Play Controls
Edit Filters
Show All/None
Realtime
Simulation
(Select a Device to Drag and Drop to the Workspace)
Toggle PDU List Window


Cisco Packet Tracer
Logical
Physical
Simulation Panel
Event List
Reset Simulation
Constant Delay
Play Controls
Edit Filters
Show All/None
Realtime
Simulation
(Select a Device to Drag and Drop to the Workspace)
Toggle PDU List Window





Conclusion

The experiment successfully created a simple network using Cisco Packet Tracer, with two PCs connected through a switch. The configuration of IP addresses and subnet masks allowed the PCs to communicate. The successful ping tests and packet flow observations confirmed that the network was correctly set up

and functional. This setup illustrates basic networking concepts, including device connectivity, IP addressing, and the use of switches in a local area network (LAN).

EX. NO. 8

Aim: - - A network communication model is created by establishing connection between a client and a server. The connection is also guaranteed by transferring client's IP address to the server and displays it in the server's premises. Write a program for the above situation

Materials Required:-

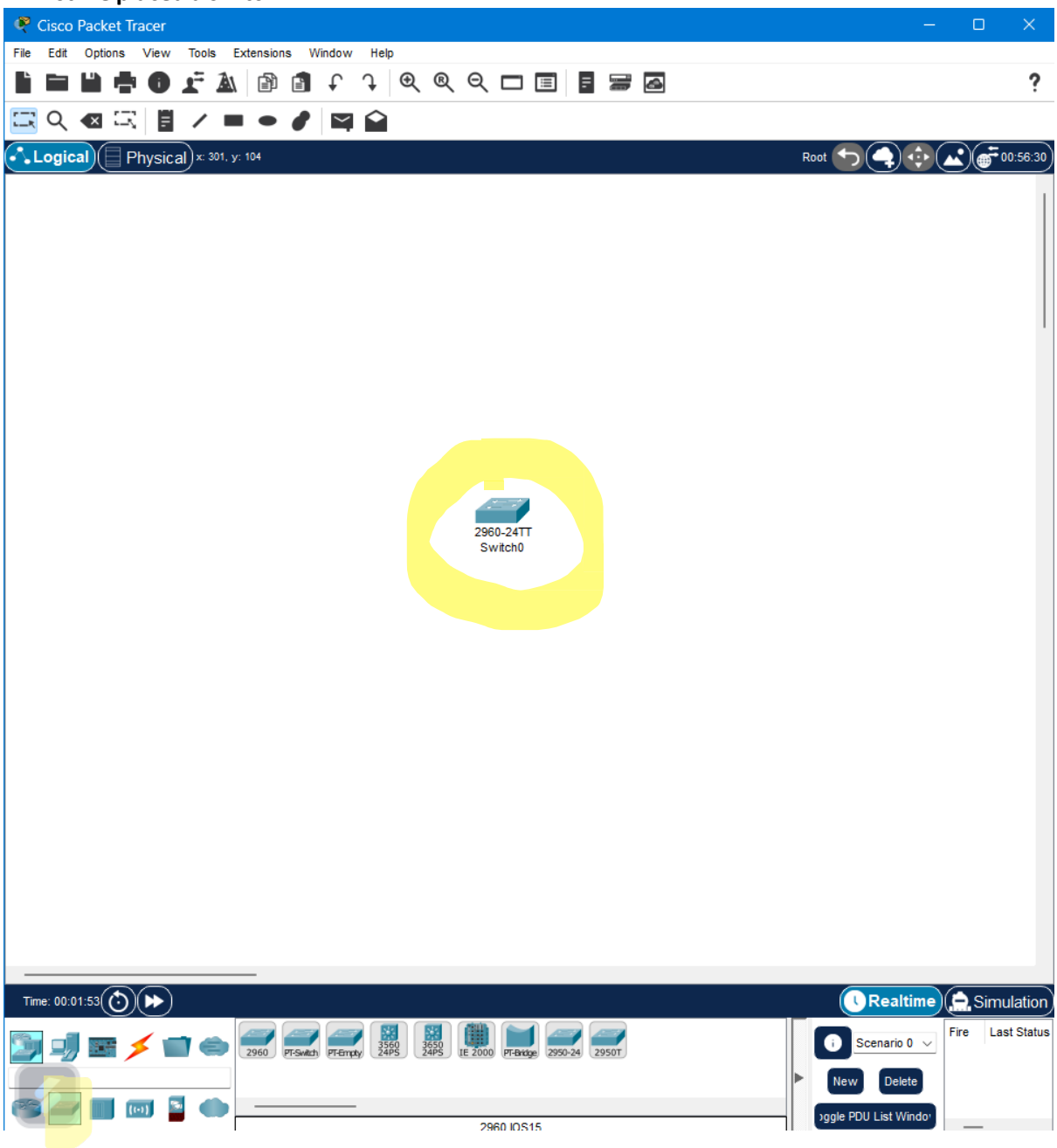
- Computer with a programming environment (e.g., Python, MATLAB, or C++)
- Packet Tracer
- Sample data for testing
- Reference materials on network communication model.

Theory:-

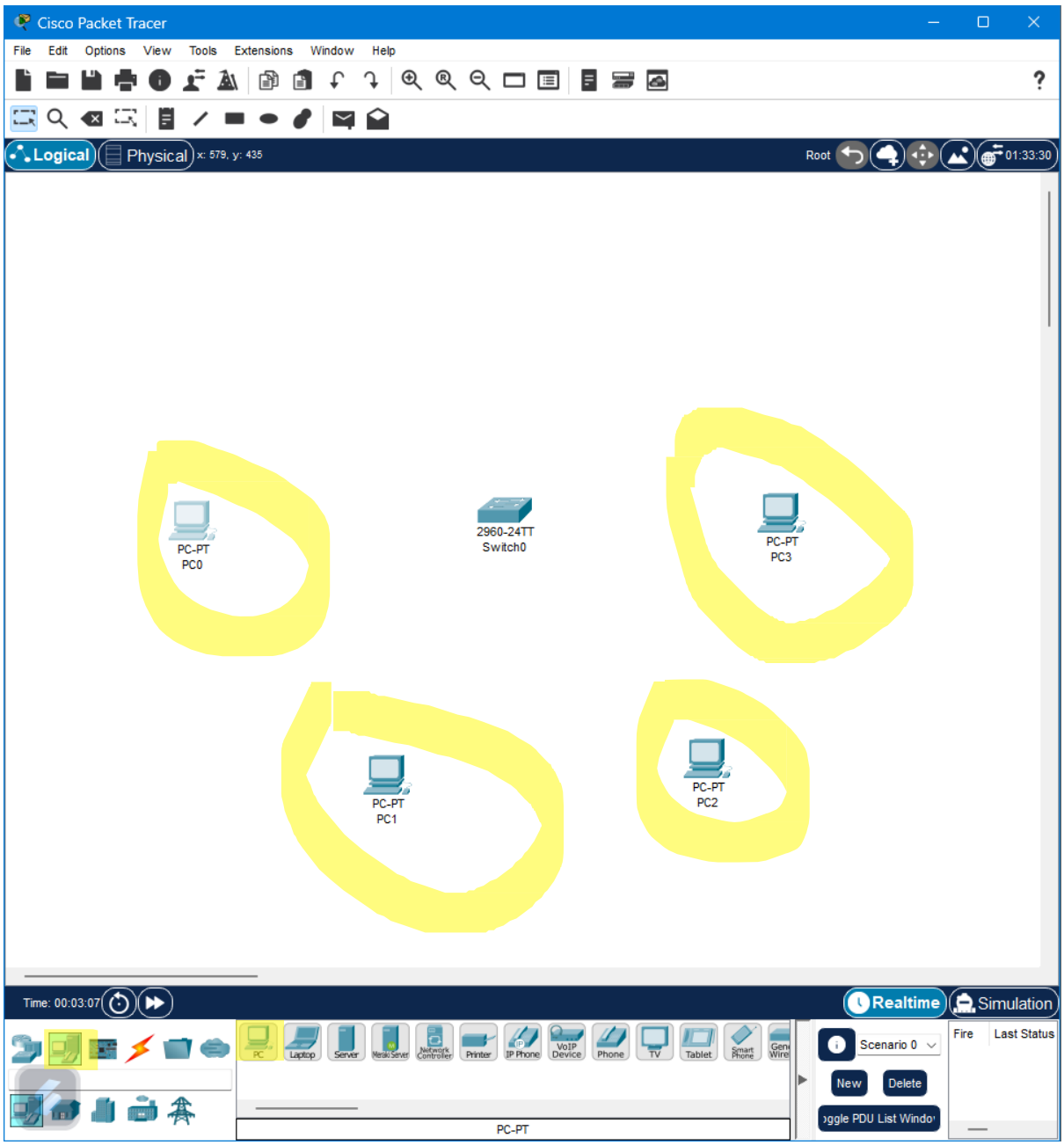
Procedure:-

So here I am using 4 PC (client), 1 switch, and 1 server. The IP management configured from the Server, which I setted to DHCP.
procedure is defined in every step:

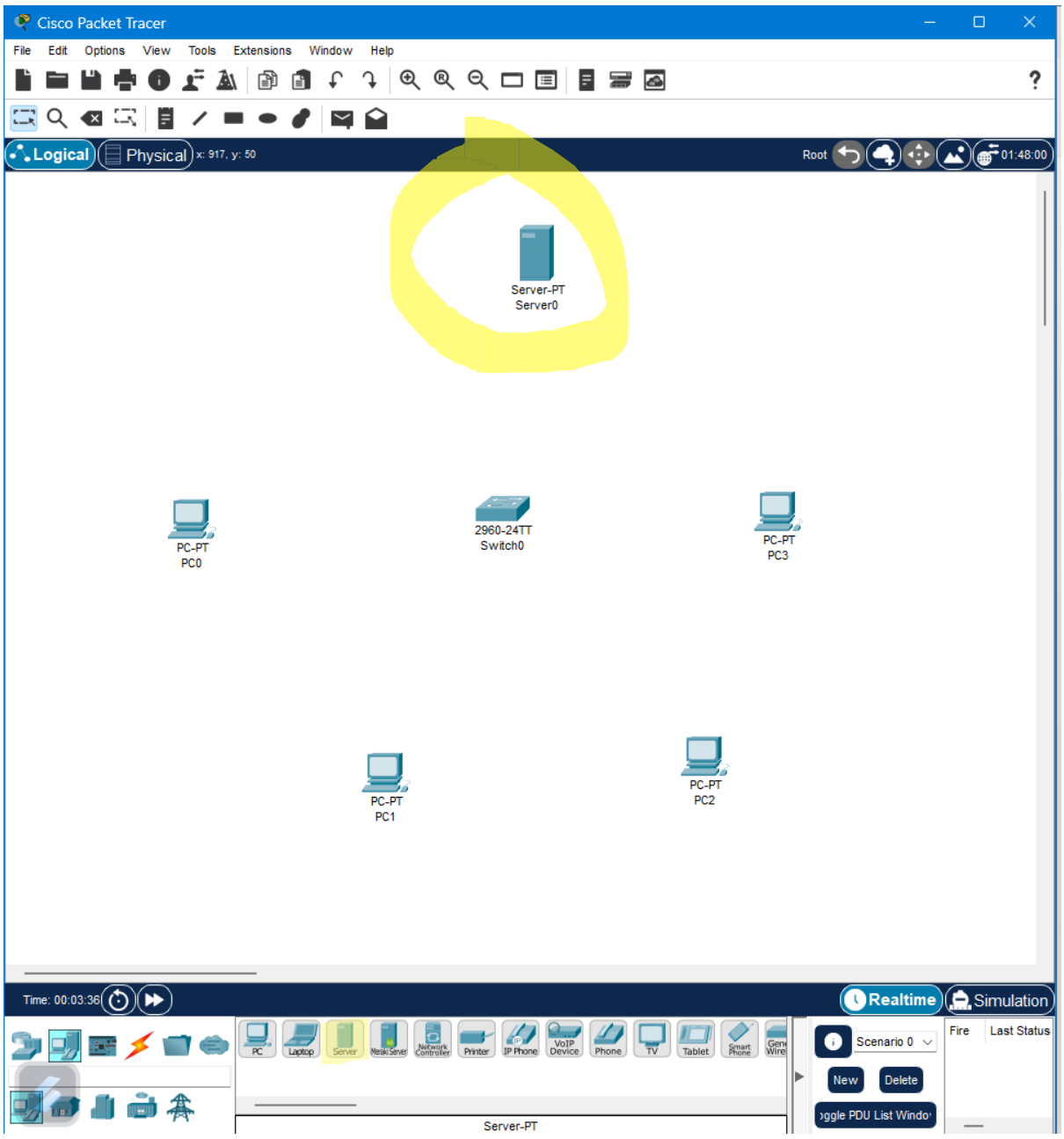
1. First we placed a switch



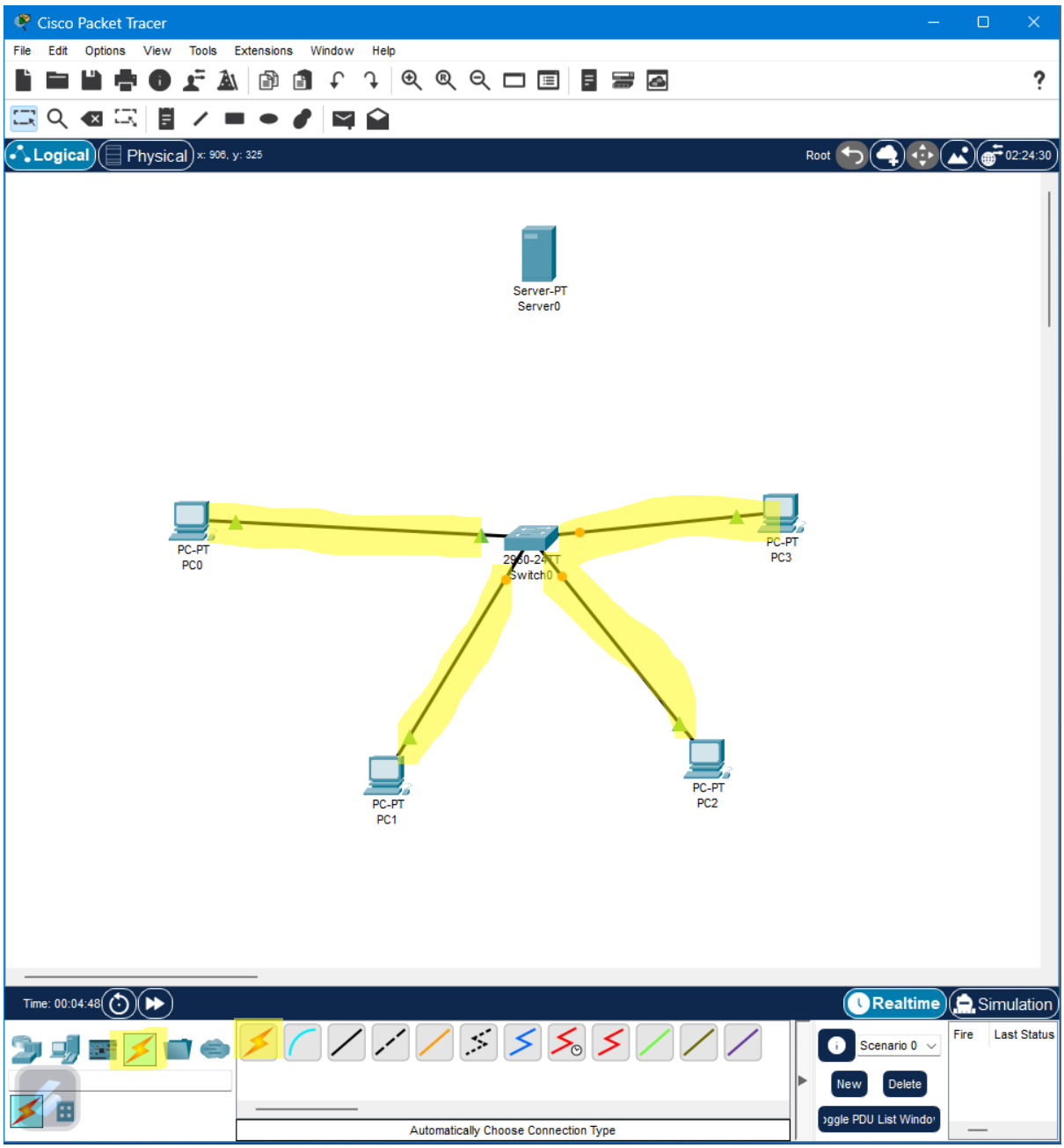
2. Then we placed 4 PC .


Cisco Packet Tracer
File Edit Options View Tools Extensions Window Help
Logical
Physical
x: 579, y: 435
Root
01:33:30
PC-PT
PC0
2960-24TT
Switch0
PC-PT
PC3
PC-PT
PC1
PC-PT
PC2
Time: 00:03:07
Realtime
Simulation
Scenario 0
New
Delete
Fire
Last Status
PC-PT

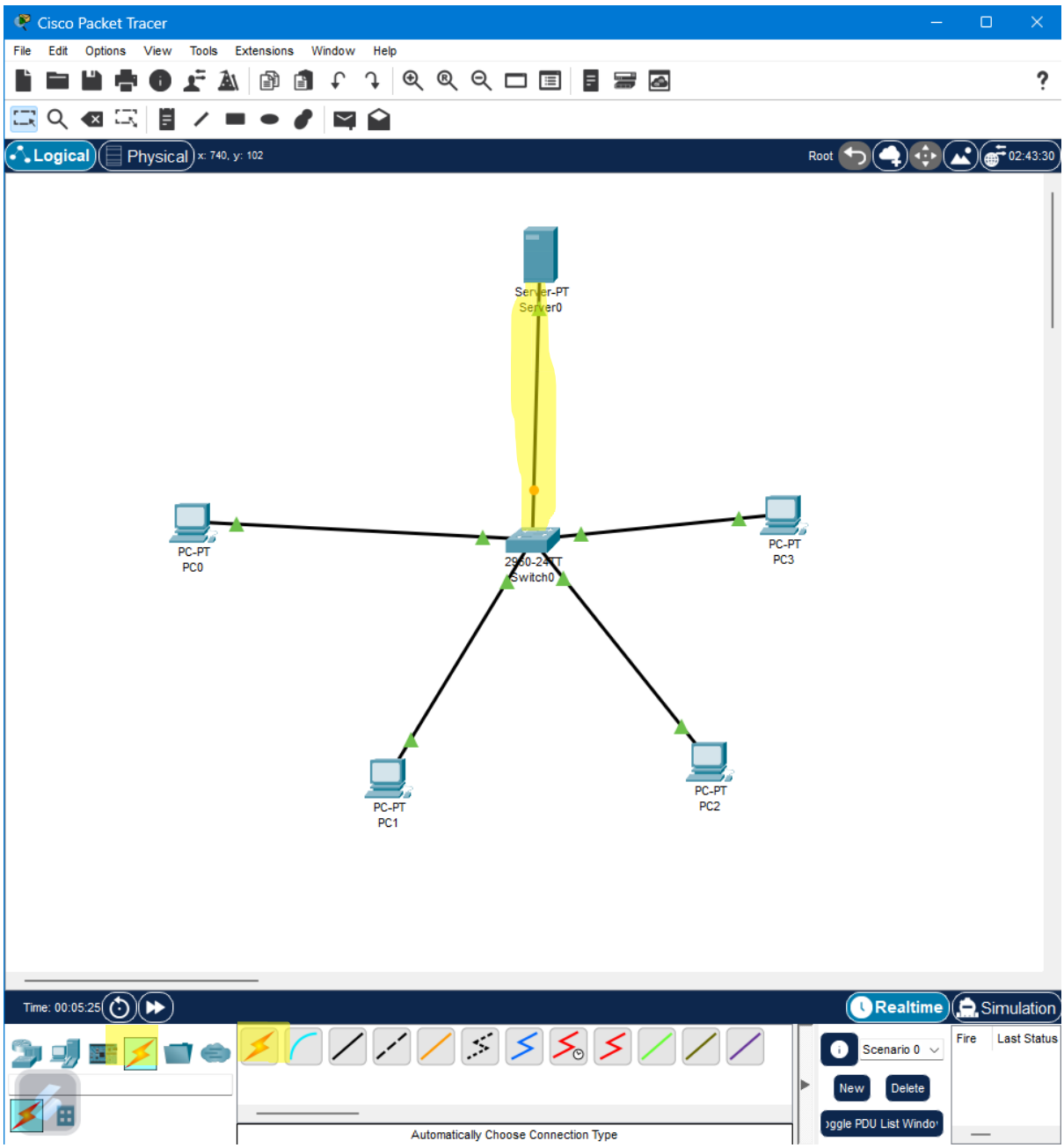
3. Then we added a server .



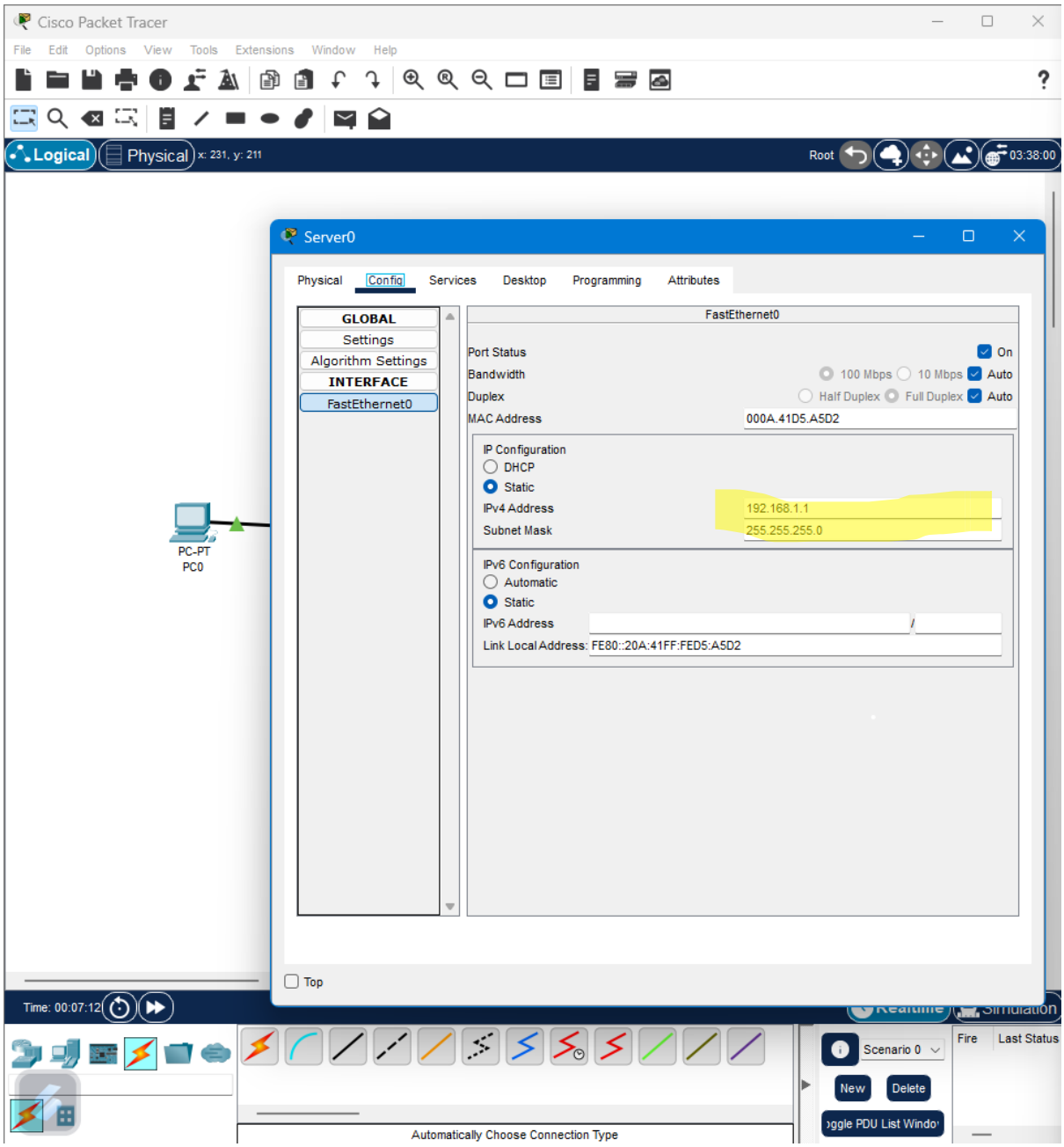
4. Then we established a connection between the PC and the switch.



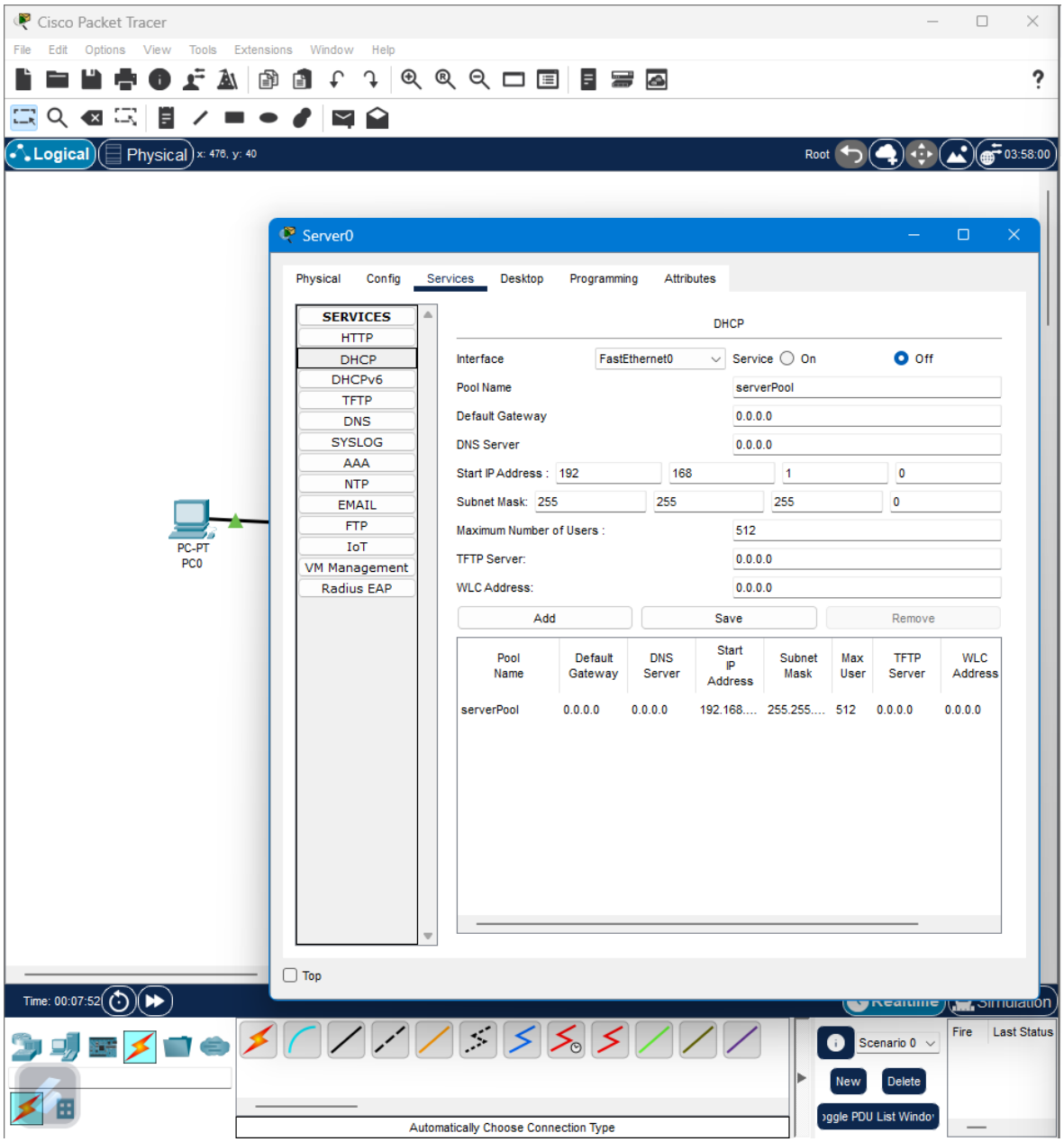
5. Then we had a guided connection with the server .



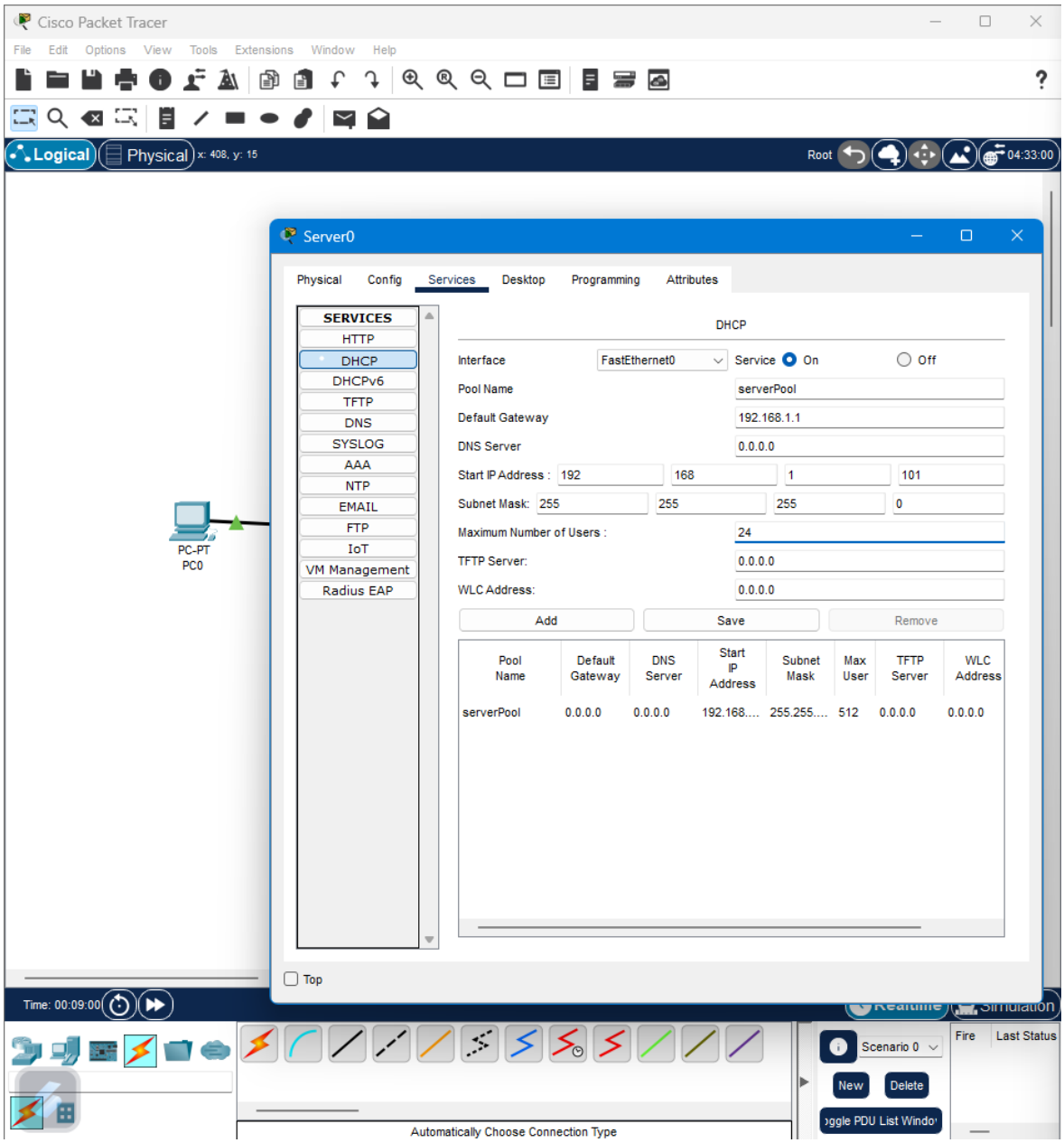
6. Then we configure the server .



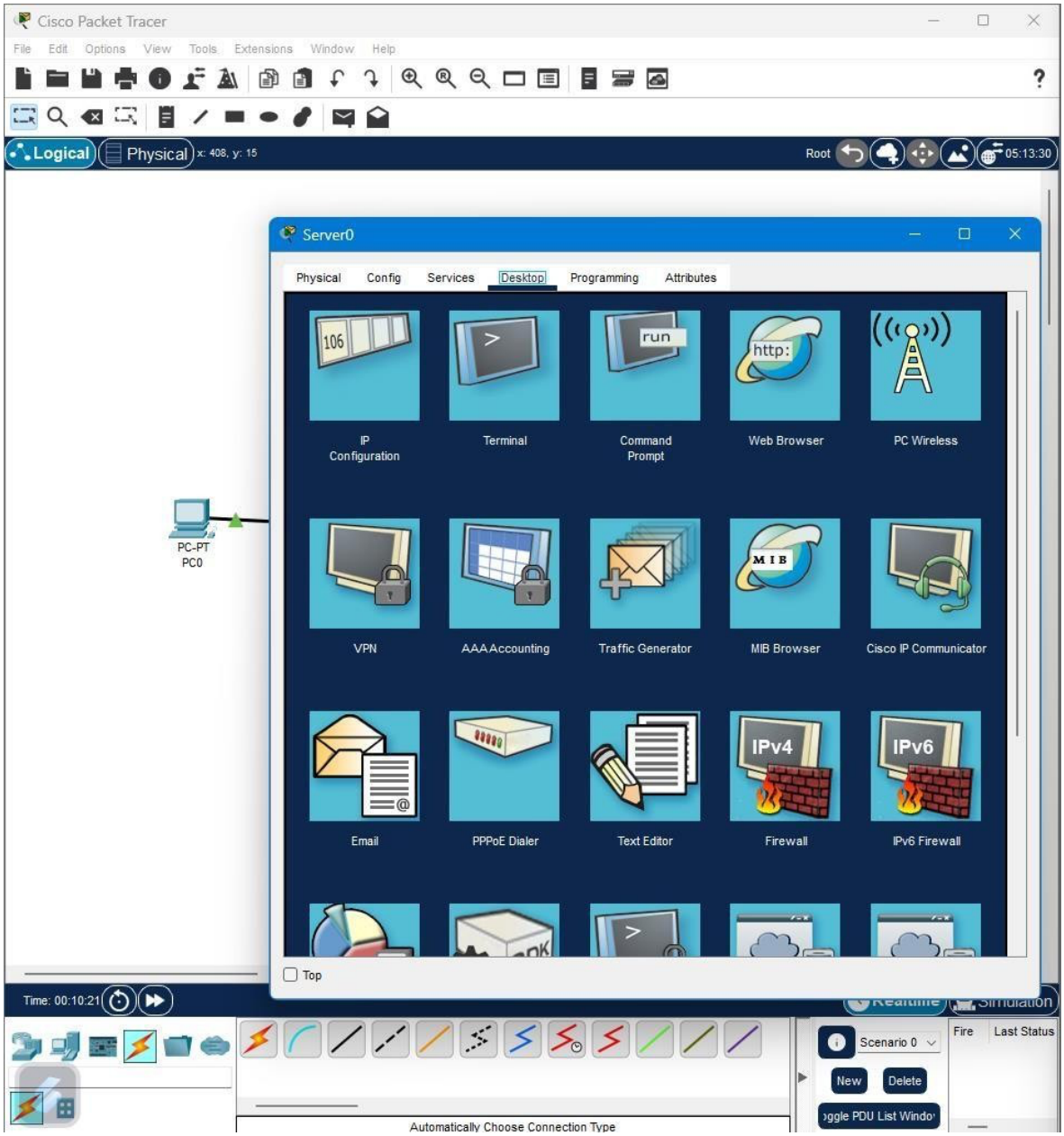
7. Then we configure the server also.



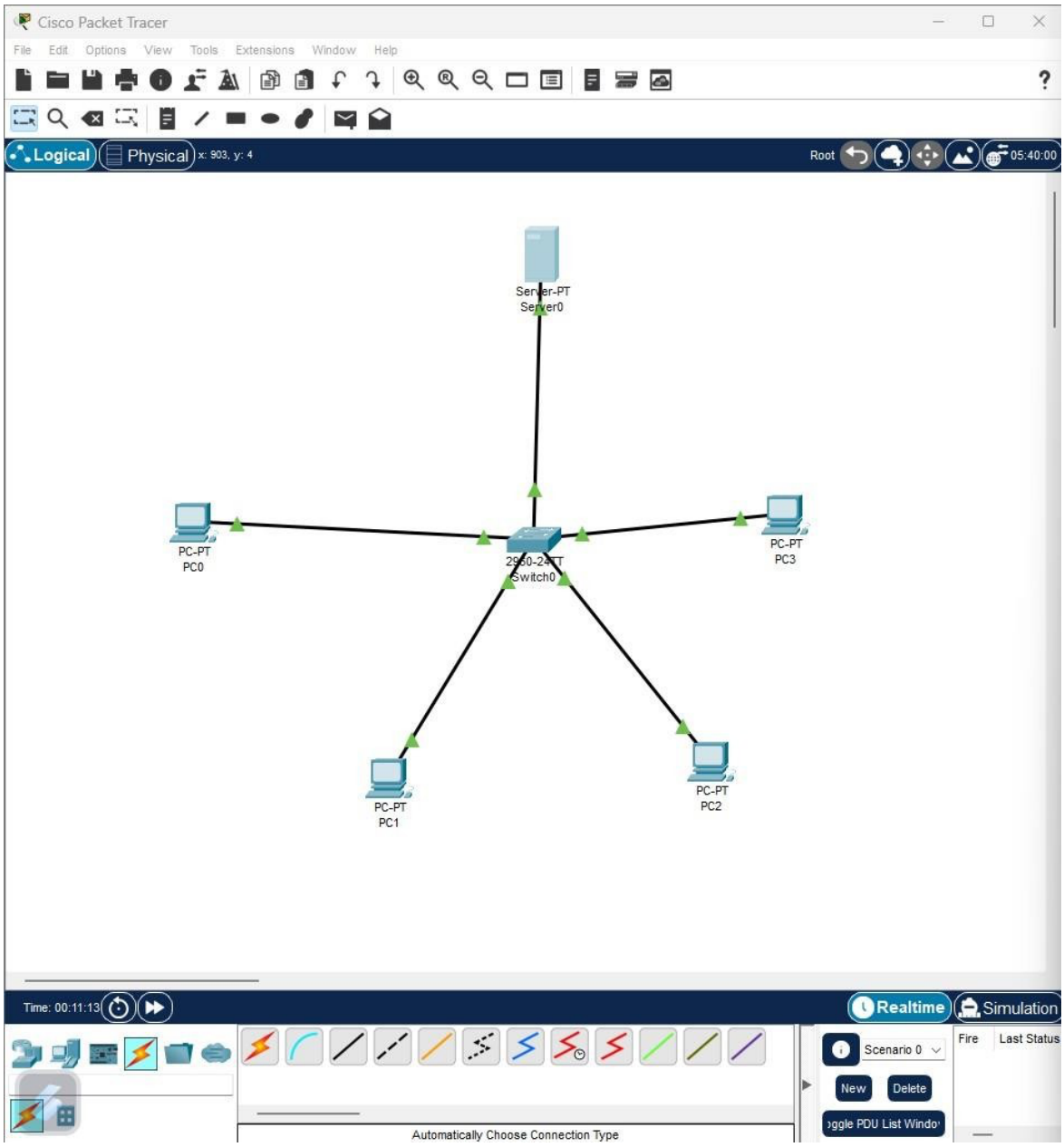
8.


Cisco Packet Tracer
File Edit Options View Tools Extensions Window Help
Logical Physical x: 408, y: 15
Root 04:33:00
Server0
Physical Config Services Desktop Programming Attributes
SERVICES
HTTP
DHCP
DHCPv6
TFTP
DNS
SYSLOG
AAA
NTP
EMAIL
FTP
IoT
VM Management
Radius EAP
DHCP
Interface FastEthernet0 Service On Off
Pool Name serverPool
Default Gateway 192.168.1.1
DNS Server 0.0.0.0
Start IP Address : 192 168 1 101
Subnet Mask: 255 255 255 0
Maximum Number of Users : 24
TFTP Server: 0.0.0.0
WLC Address: 0.0.0.0
Add Save Remove
Pool Name Default Gateway DNS Server Start IP Address Subnet Mask Max User TFTP Server WLC Address
serverPool 0.0.0.0 0.0.0.0 192.168.... 255.255.... 512 0.0.0.0 0.0.0.0
Top
PC-PT PC0
Time: 00:09:00
Scenario 0
New Delete
Fire Last Status
Automatically Choose Connection Type

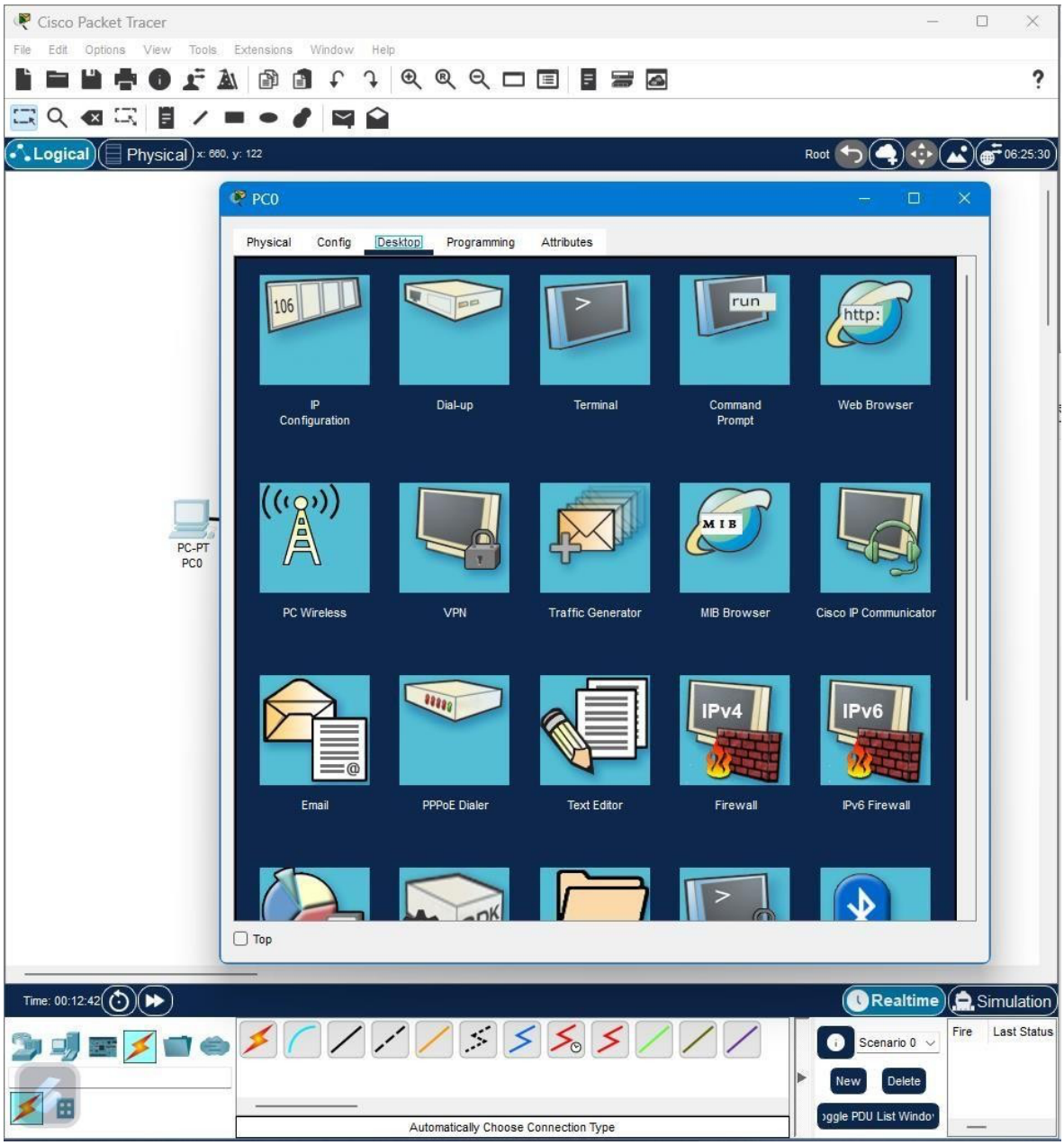
9.


Cisco Packet Tracer
File Edit Options View Tools Extensions Window Help
Logical
Physical
x: 408, y: 15
Root
05:13:30
Server0
Physical Config Services Desktop Programming Attributes
IP Configuration
Terminal
Command Prompt
Web Browser
PC Wireless
VPN
AAA Accounting
Traffic Generator
MIB Browser
Cisco IP Communicator
Email
PPPoE Dialer
Text Editor
Firewall
IPv6 Firewall
Top
PC-PT PC0
Time: 00:10:21
Realtime
Simulation
Scenario 0
New
Delete
Fire
Last Status
Automatically Choose Connection Type

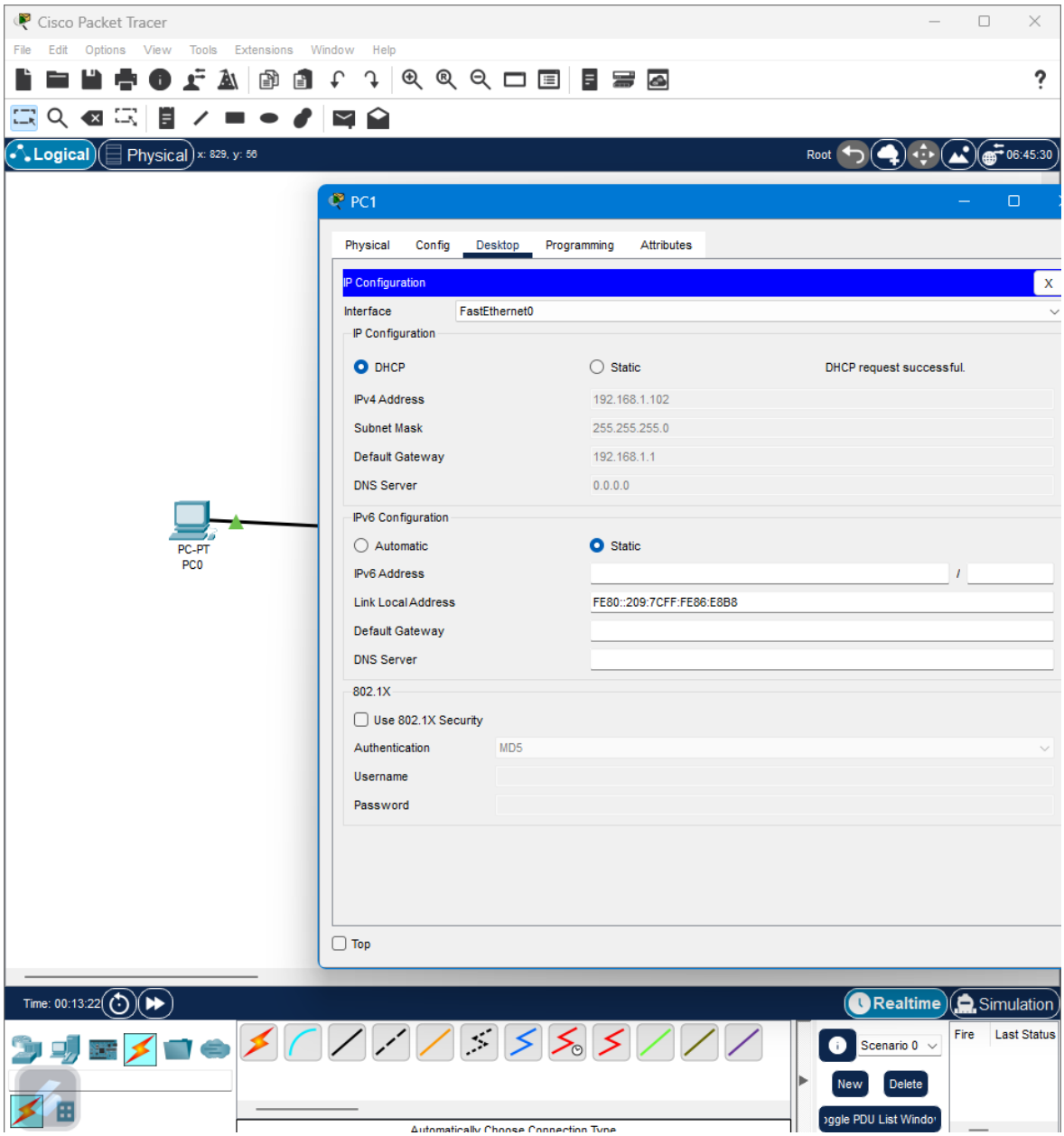
10.


Cisco Packet Tracer
File Edit Options View Tools Extensions Window Help
Logical
Physical
x: 903, y: 4
Root
05:40:00
Server-PT
Server0
PC-PT
PC0
2960-24TT
Switch0
PC-PT
PC3
PC-PT
PC1
PC-PT
PC2
Time: 00:11:13
Realtime
Simulation
Scenario 0
New
Delete
Fire
Last Status
Automatically Choose Connection Type

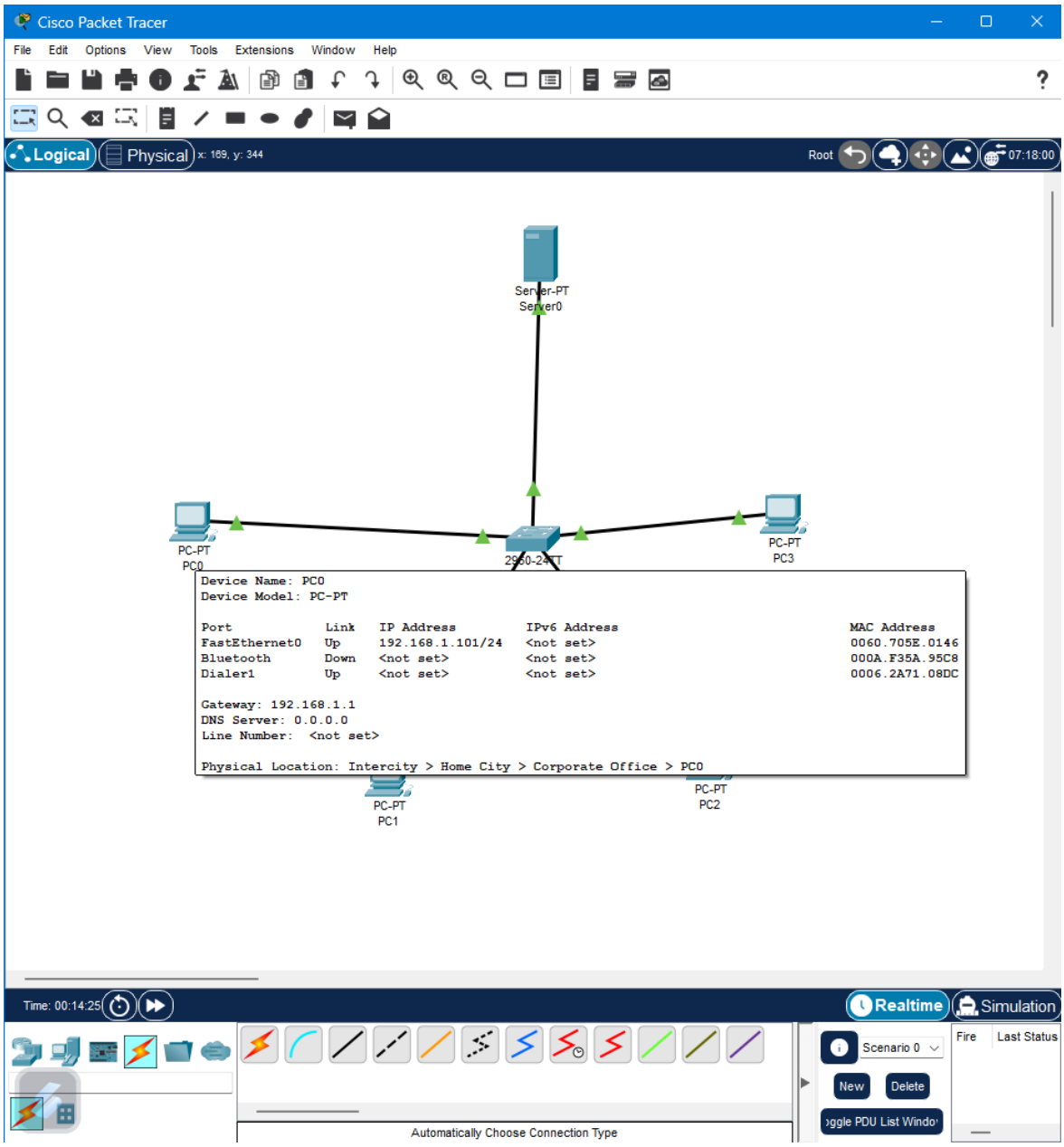
11. We go to desktop and go to IP configuration .



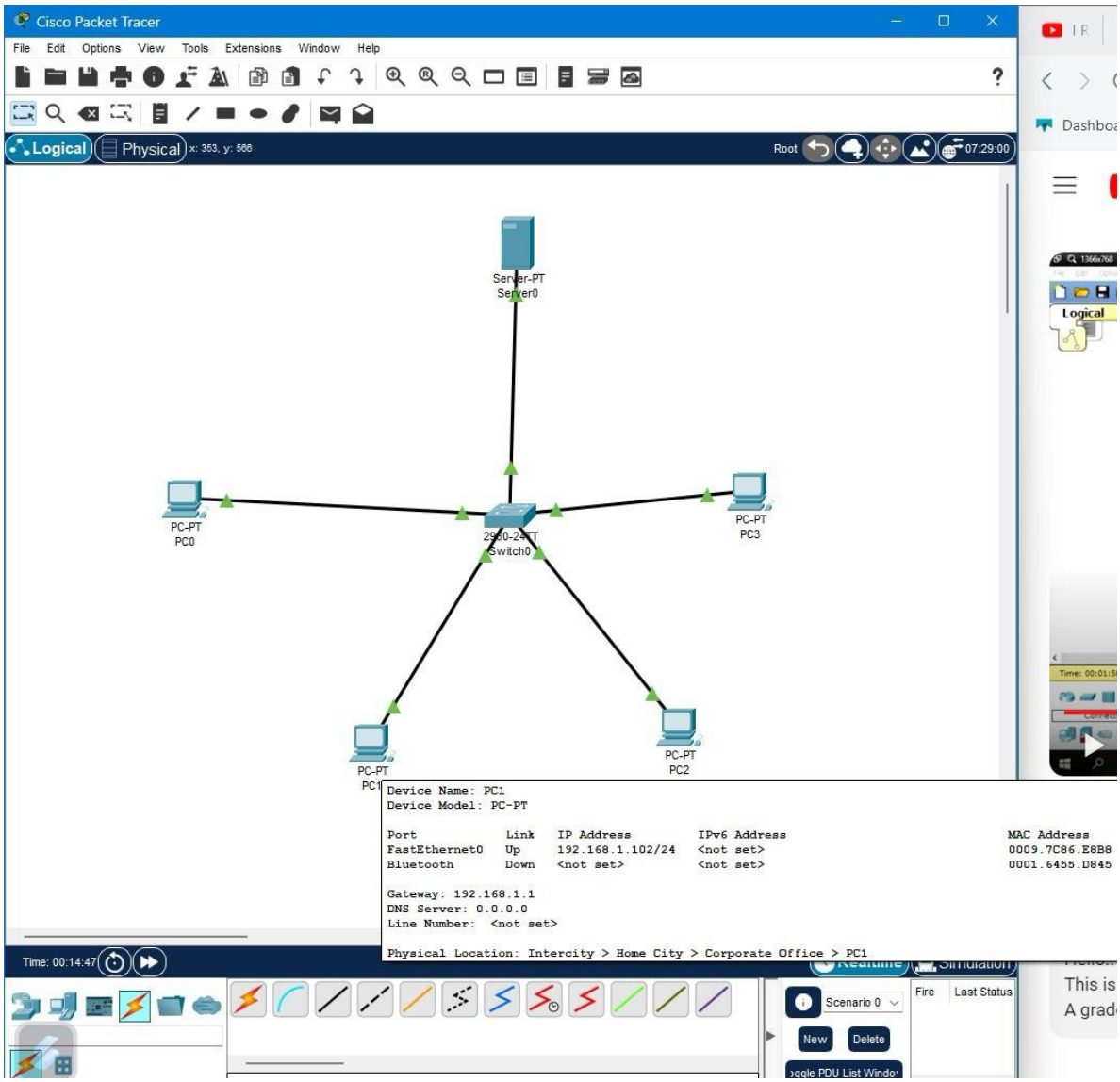
12. We go to desktop and change the IP configuration to DHCP and same for the other PC.



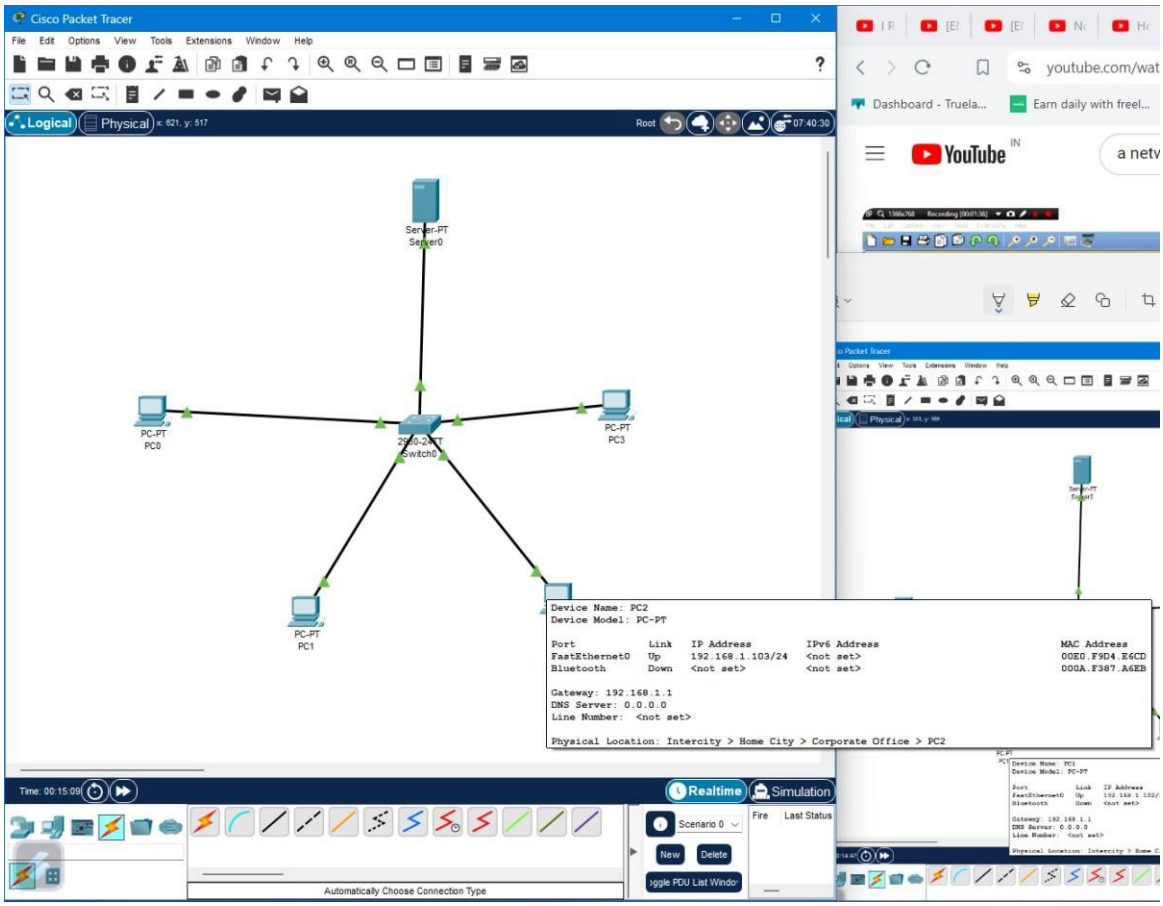
13. The information about the PC 1



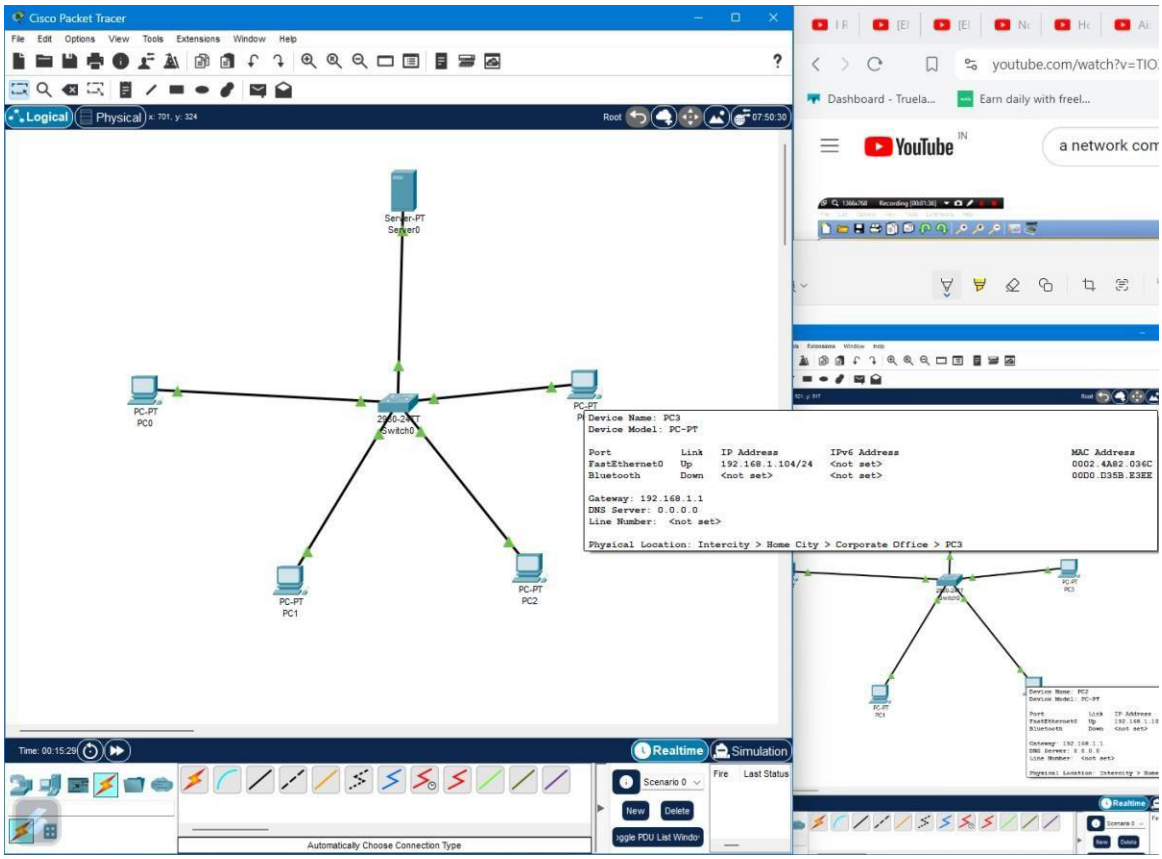
14. The information about the PC 2.



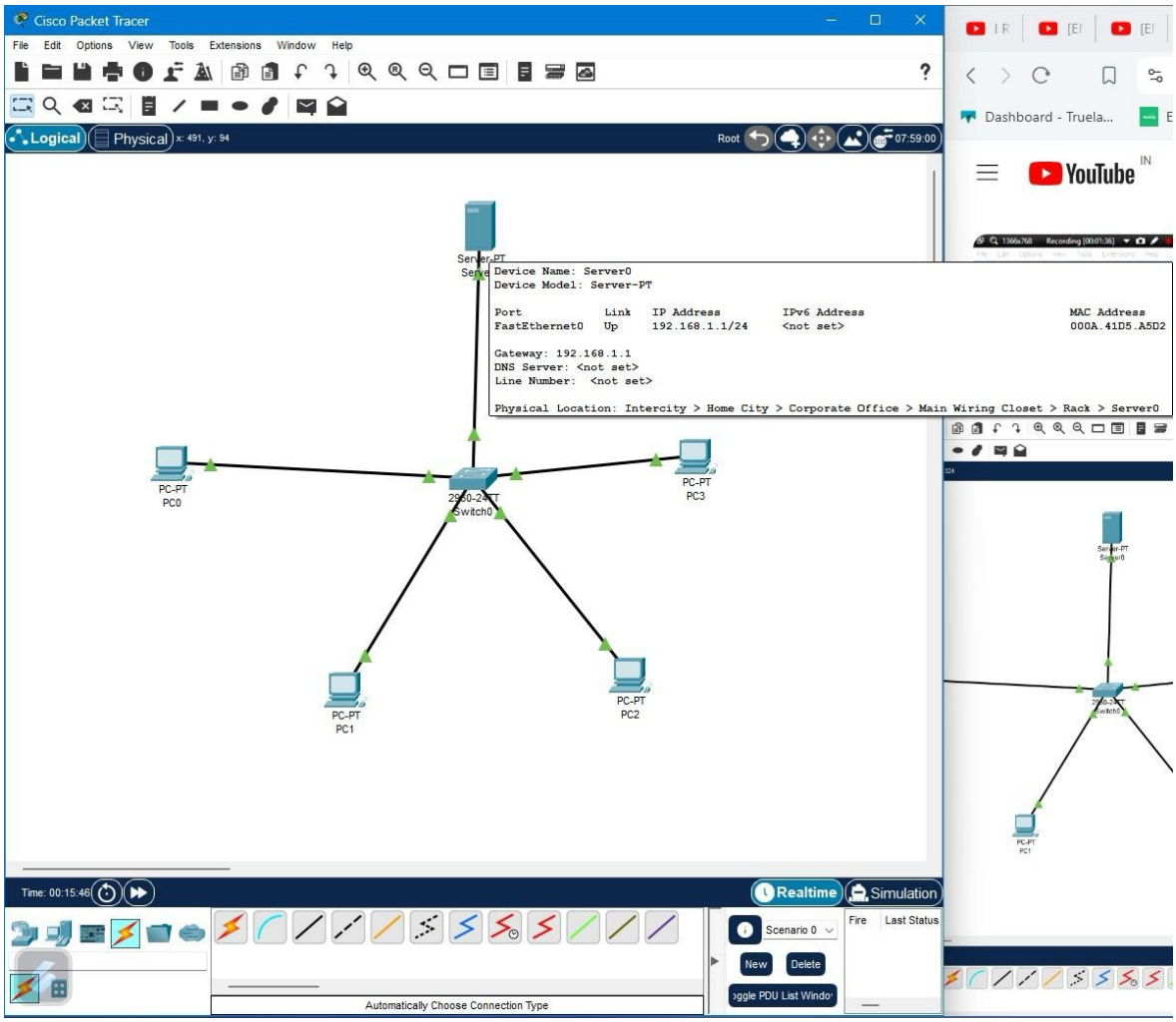
15. The information about the PC 3.



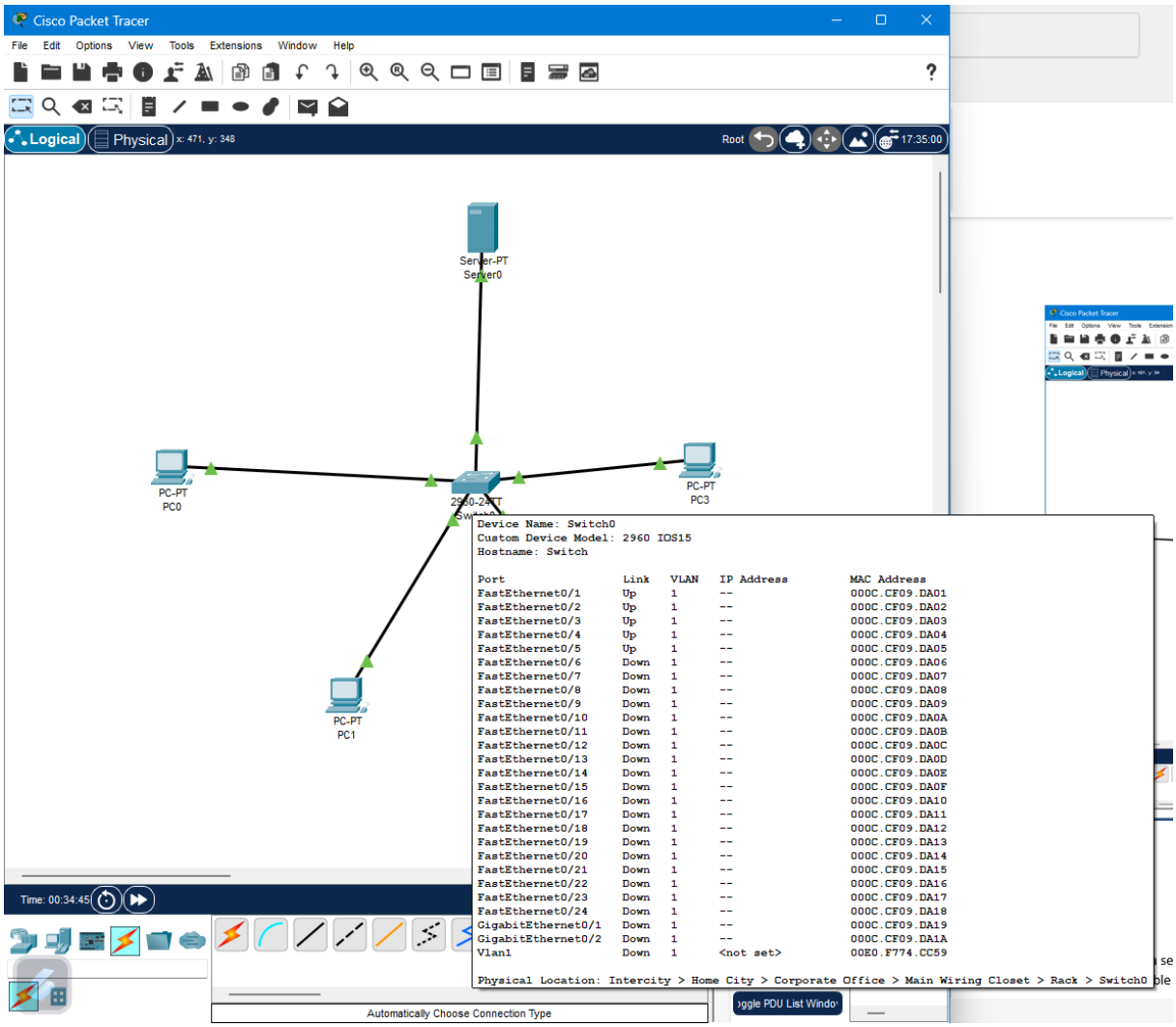
16. The information about the PC 4.



17. The information about the server.



18. The information about the switch.



Observations:-

That the 4 pc and a server and a switch are connected in a way that the data transmission from one to another is possible without an error.

Conclusion:-

A network communication model is created by establishing connection between a client and a server. The connection is also guaranteed by transferring client's IP address to the server and displays it in the server's premises. Write a program for the above situation

CODE IN PYTHON:

To create a network communication between a client and a server, where the server can display the client's IP address, you can use socket programming. Below is an example using Python:

Server Program

python

```
import socket


def start_server(host='0.0.0.0', port=12345):
    # Create a socket object
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Bind the socket to the host and port
    server_socket.bind((host, port))

    # Start listening for connections
    server_socket.listen(1)
    print(f"Server listening on {host}:{port}")

    while True:
        # Accept a connection
        client_socket, client_address = server_socket.accept()
        print(f"Connection established with {client_address}")

        # Get client's IP address
        client_ip = client_address[0]
        print(f"Client's IP address: {client_ip}")
```

```
        # Send a response back to the client
        client_socket.sendall(b"Hello from the server!")

        # Close the connection
        client_socket.close()

if __name__ == "__main__":
    start_server()
```

Client Program

```
python
import socket

def start_client(server_host='127.0.0.1', server_port=12345):
    # Create a socket object
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect to the server
    client_socket.connect((server_host, server_port))

    # Receive the response from the server
    response = client_socket.recv(1024)
    print(f"Received from server: {response.decode()}")

    # Close the connection
    client_socket.close()

if __name__ == "__main__":
    start_client()
```

Explanation

- **Server Program:**
    - The server creates a socket and binds it to the specified host and port.

- It then listens for incoming connections.
- When a client connects, the server accepts the connection and prints the client's IP address.
- The server sends a response to the client and then closes the connection.

- **Client Program:**
  - The client creates a socket and connects to the server using the server's IP address and port.
  - The client receives a response from the server, prints it, and then closes the connection.

Running the Programs

1. **Start the Server:** Run the server program first. It will start listening for incoming connections.

bash

python server.py

2. **Start the Client:** Run the client program to connect to the server.

bash

python client.py

You should see the client's IP address displayed on the server's console. The client will also receive and print a response from the server.

4o

EX. NO. 9

Aim: − Consider two processes client and server communicates across a network. The client sends a message to the server in the request and the server responds with the same message. Write a Socket program for the above mentioned scenario.:

Algorithms:

Server:

1. Create a socket.
2. Bind the socket to a host and port.
3. Listen for incoming connections.
4. Accept a connection from a client.
5. Receive a message from the client.
6. Send the same message back to the client.
7. Close the client connection.
8. Repeat steps 4-7 for each incoming connection.

Client:

1. Create a socket.
2. Connect the socket to the server's host and port.
3. Send a message to the server.
4. Receive the response from the server.
5. Print the response.
6. Close the socket.

Program:

Server code

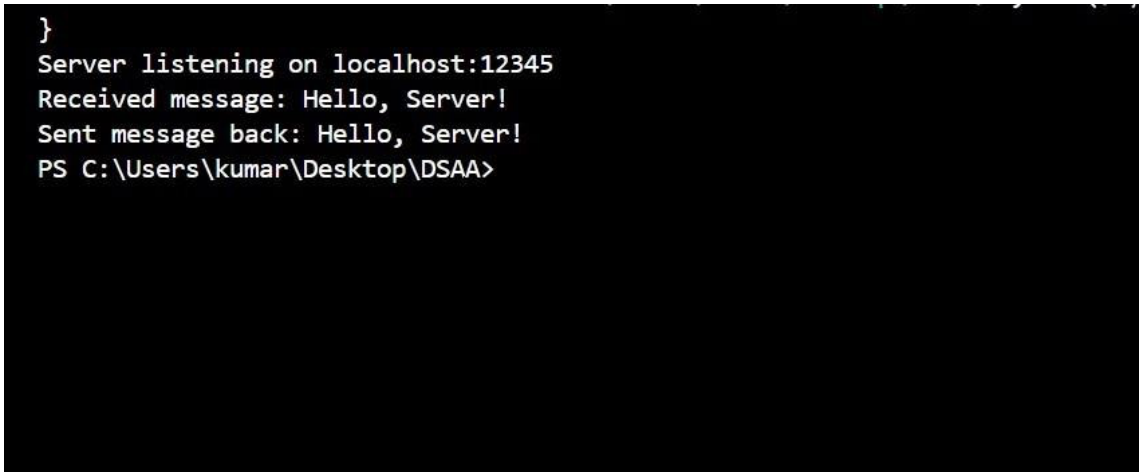
```python
import socket
def start_server(host='localhost', port=12345):

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))

    # Listen for incoming connections
    server_socket.listen(1)
    print(f"Server listening on {host}:{port}")

    while True:
        # Accept a connection
        client_socket, addr = server_socket.accept()
        print(f"Connection from {addr}")

        # Receive data from the client
        data = client_socket.recv(1024).decode('utf-8')
        print(f"Received message: {data}")

        # Send the same data back to the client
        client_socket.sendall(data.encode('utf-8'))
        print(f"Sent message back: {data}")

        # Close the client socket
        client_socket.close()

if __name__ == "__main__":
    start_server()

```

Client Side :

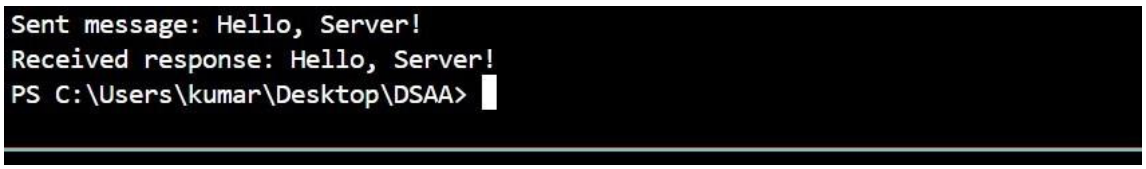
```python
import socket

def start_client(host='localhost', port=12345, message='Hello, Server!'):
    # Create a socket object
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect to the server
    client_socket.connect((host, port))

    # Send a message to the server
    client_socket.sendall(message.encode('utf-8'))
    print(f"Sent message: {message}")

    # Receive the response from the server
    response = client_socket.recv(1024).decode('utf-8')
    print(f"Received response: {response}")

    # Close the socket
    client_socket.close()

if __name__ == "__main__":
    start_client()

```

Output:

Server-side



Client- side



Result:

The program demonstrates a simple echo service where the server responds with the same message it receives from the client. Adjust the host, port, and message parameters as needed for different configurations.

EX. NO. 10

To study various TCL commands.

Aim: - To understand and execute various TCL (Tool Command Language) commands used in network simulations and configurations.

Materials Required: -

- Computer with TCL interpreter (e.g., ActiveTcl, NS2)
- Text editor or IDE (e.g., Notepad++, VS Code)
- TCL reference materials (books, online resources, official documentation)

Theory: -

TCL is a powerful scripting language used in network simulations, configuration management, and automation tasks. It is extensively used in network simulation tools like NS2/NS3 to create, configure, and manage network topologies, nodes, and links.

Procedure: -

1. Basic TCL Commands

- **Description:** This section covers basic TCL commands for variable assignment, mathematical operations, and output display, which are foundational for writing more complex network simulation scripts.
- **Implementation Steps:**
  1. **Variable Assignment:** Assign a value to a variable using the `set` command.
  2. **Mathematical Operations:** Perform basic arithmetic operations using the `expr` command.
  3. **String Operations:** Manipulate and display strings.

2. Network Node and Link Configuration

- **Description:** This section demonstrates how to configure network nodes and links using TCL commands in the context of network simulations.
- **Implementation Steps:**

  1. **Create Simulator Instance:** Initialize the network simulator.
  2. **Define Nodes:** Create network nodes.
  3. **Create Links:** Establish a link between the nodes.
  4. **Set Node Attributes:** Configure node properties such as queue type and size.

3. Traffic Generation and Simulation Control

- **Description:** This section focuses on generating network traffic and controlling the simulation.
- **Implementation Steps:**

1. **Define Traffic Sources:** Create a traffic generator (e.g., Constant Bit Rate (CBR) traffic).
2. **Attach Traffic to Nodes:** Attach the traffic generator to a node.
3. **Simulation Time Control:** Schedule simulation events and define the simulation end time.
4. **Run the Simulation:** Execute the simulation.

Observations: -

S.NO	TCL Command	Description	Output/Effect
1	set var1 10	Variable Assignment	Assigns value 10 to var1
2	$ns node	Node Creation	Creates a network node
3	$ns duplex-link	Link Creation between nodes	Establishes a duplex link with specified bandwidth and delay
4	$cbr start	Start Traffic Generation	Begins CBR traffic generation at specified time
5	$ns run	Run Simulation	Executes the simulation script

Conclusion: -

In this experiment, we explored various TCL commands used in network simulations, including basic operations, node and link configuration, traffic generation, and simulation control. Understanding these commands is crucial for effectively creating and managing network simulations using TCL in tools like NS2/NS3. This knowledge provides a foundation for more advanced network simulation tasks and automated network management.

EX. NO. 11

The message entered in the client is sent to the server and the server encodes the message and returns it to the client. Encoding is done by replacing a character by the character next to it.(i.e.) a as b, b as c…z.

Aim: - To implement a client-server communication system where the server encodes a received message by replacing each character with the next character in the alphabet and returns the encoded message to the client

Requirements

- Python programming language
- Socket library

Theory

Client-server communication involves two programs: a client that sends a message and a server that receives, processes, and responds to the message. In this experiment, the server encodes the message by shifting each character to the next character in the alphabet (e.g., 'a' becomes 'b', 'b' becomes 'c', and 'z' becomes 'a').

Equipment/Software

- A computer with Python installed
- Access to a network for communication between client and server

Procedure

Step 1: Server Program

```
import socket

 def encode_message(message):

   encoded_message = ""
```

```
    for char in message:
        if char.isalpha():
            if char == 'z':
                encoded_message += 'a'
            elif char == 'Z':
                encoded_message += 'A'
            else:
                encoded_message += chr(ord(char) + 1)
        else:
            encoded_message += char
    return encoded_message

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('localhost', 12345))
server_socket.listen(1)
print("Server is listening on port 12345")

while True:
    client_socket, addr = server_socket.accept()
    print(f"Connection from {addr} has been established.")
    message = client_socket.recv(1024).decode()
    print(f"Received message: {message}")
    encoded_message = encode_message(message)
    client_socket.send(encoded_message.encode())
    client_socket.close()
```

Step 2: Client Program

```
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

client_socket.connect(('localhost', 12345))

message = "hello world"

client_socket.send(message.encode())

encoded_message = client_socket.recv(1024).decode()

print(f"Encoded message: {encoded_message}")

client_socket.close()
```

Step 3: Running the Experiment

1. **Start the server:**
    - Run the server program. The server will start listening for incoming connections.
2. **Run the client:**
    - Run the client program. The client will connect to the server, send a message, and receive the encoded message.
3. **Observe the Output:**
    - The server console will display the received message and the encoded message.
    - The client console will display the encoded message received from the server.

Results

- The client sends a message to the server.
- The server encodes the message by replacing each character with the next character in the alphabet.
- The encoded message is sent back to the client.
- The client displays the encoded message.

Conclusion

This experiment demonstrates a basic client-server communication model where the server processes a message by encoding it and returns the encoded message to the client. The server correctly handles the encoding of characters, wrapping around from 'z' to 'a'.

EX. NO. 12

Packet Tracer: Observing Packets across the network and Performance Analysis of Routing protocols

Aim: - Packet Tracer: Observing Packets across the network and Performance Analysis of Routing protocols .

Materials Required:-

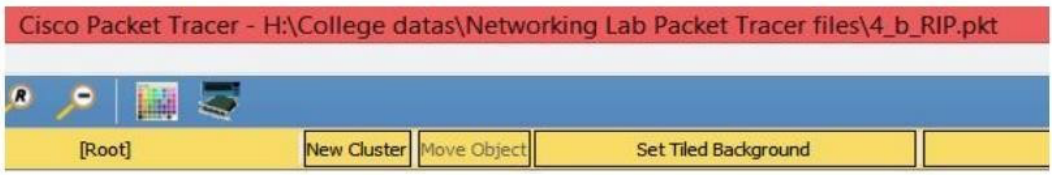
- Windows pc – 2 Nos
- CISCO Packet Tracer Software (Student Version)
- 8 port switch – 2 No
- Router – 2 Nos
- Cat-5 LAN cable

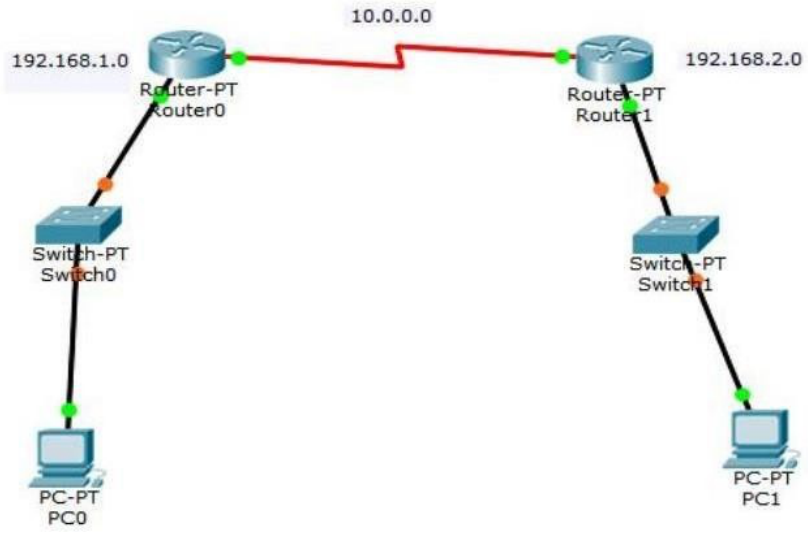
Theory:- RIP (Routing Information Protocol) is one of the oldest distance vector routing protocols. It is usually used on small networks because it is very simple to configure and maintain, but lacks some advanced features of routing protocols like OSPF or EIGRP. Two versions of the protocol exists: version 1 and version 2. Both versions use hop count as a metric and have the administrative distance of 120. RIP version 2 is capable of advertising subnet masks and uses multicast to send routing updates, while version 1 doesn't advertise subnet masks and uses broadcast for updates. Version 2 is backwards compatible with version 1. RIPv2 sends the entire routing table every 30 seconds, which can consume a lot of bandwidth. RIPv2 uses multicast address of 224.0.0.9 to send routing updates, supports authentication and triggered updates (updates that are sent when a change in the network occurs).

Procedure:- Open the CISCO Packet tracer software

- Drag and drop 5 pcs using End Device Icons on the left corner
- Select 8 port switch from switch icon list in the left bottom corner
- Select Routers and Give the IP address for serial ports of router and apply clock rate as
- per the table.
- Make the connections using Straight through Ethernet cables
- Ping between PCs and observe the transfer of data packets in real and simulation mode.

Network Topology Diagram for RIP





Input Details

PC0	PC1	Router 0	Router 1
IP Address : 192.168.1.2 Gate way : 192.168.1.1	IP Address: 192.168.2.2 Gate way : 192.168.2.1	Fast Ethernet 0/0 IP Address: 192.168.1.1 Serial 2/0 : 10.0.0.1 at 6400 clock rate	Fast Ethernet 0/0 IP Address : 192.168.2.1 Serial 2/0 : 10.0.0.2 no clock rate

OUTPUT:

RIP (PINGING FROM PC0 TO PC1):
C:\>ping 192.168.2.2
Pinging 192.168.2.2 with 32 bytes of data:
Reply from 192.168.2.2: bytes=32 time=11ms TTL=126
Reply from 192.168.2.2: bytes=32 time=12ms TTL=126
Reply from 192.168.2.2: bytes=32 time=13ms TTL=126
Reply from 192.168.2.2: bytes=32 time=11ms TTL=126
Ping statistics for 192.168.2.2:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 11ms, Maximum = 13ms, Average = 11ms