NAME: KEERTHISRI.D

ROLL NO:241901047

DATE:18.08.2025

EXERCISE-1

BASIC LINUX COMMANDS

1. *dir* - Equivalent of ls
2. *Synopsis:* dir [path] [[options]]
3. *Description:* List all files and directories.
4. *cd* - Change directory

Synopsis: cd [path]

Description: Changes the current working directory.

5. *cls* - Clear screen

Synopsis: cls

Description: Removes all text.

6. *echo* - Prints

Synopsis: echo [message]

Description: Displays messages, useful in batch scripts.

7. *dir -a* (like ls -a)

Synopsis: dir -a [path]

Description: Lists files and directories.

8. *mkdir* - make directory

Synopsis: mkdir [drive:] [path] foldername

Description: Create a new directory in a specified path.

9. *del* - delete file (remdir)

Synopsis: del file.txt

Description: Delete files or directories from the file system.

10. *tasklist* - display all currently running process

Synopsis: tasklist [options]

Description: Lists process name, PID (Process ID), session name/ID, and memory usage.

11. *find* - search for a specific string of text within files

Synopsis: find "string" [filename]

Description: Looks for the given "string" inside the specified file.

12. *systeminfo* - detailed system configuration information

Synopsis: systeminfo

Description: Often used for system audits and troubleshooting.

13. *typeperf* - display or log performance counter data

Synopsis: typeperf [counter...] [options]

Description: Can display output in the console or save it to a log file (CSV, TSV, binary).

14. *tracert* - Trace the path packets take to reach

Synopsis: tracert [options]
Description: Displays each router packets travel through until reaching the destination.

15. *ping* - network connectivity

Synopsis: ping [option]
Description: Displays packet loss, response time, and TTL (Time to Live).

16. *ipconfig* - displays and manages IP address

Synopsis: ipconfig [options]

Description: Shows current network adapter configuration.

NAME: KEERTHISRI.D

DATE: 18.08.25 ROLL NO: 241901047

EXERCISE-2

TCP CLIENT-SERVER USING SOCKET PROGRAMMING IN PYTHON

AIM:

To implement a simple client–server communication system using TCP sockets in Python, where the client sends a message to the server, and the server echoes (sends back) the same message to the client.

ALGORITHM:

SERVER:

1.Create a TCP socket and bind it to 'localhost' at port 55555.

2.Listen for incoming client connection requests (up to 3 queued).

3.Accept a client connection and receive a message from the client.

4.Print the client's message and send the same message back as a reply.

5.Ask the server operator if they want to continue; if not, close the server

CLIENT:

1.Create a TCP Socket

2. Connect the socket to the server at 'localhost' on port 55555.

3. Take input from the user as a message.

4. Send the encoded message to the server through the socket.

5.Receive and decode the server’s response, then print it.

CODE:

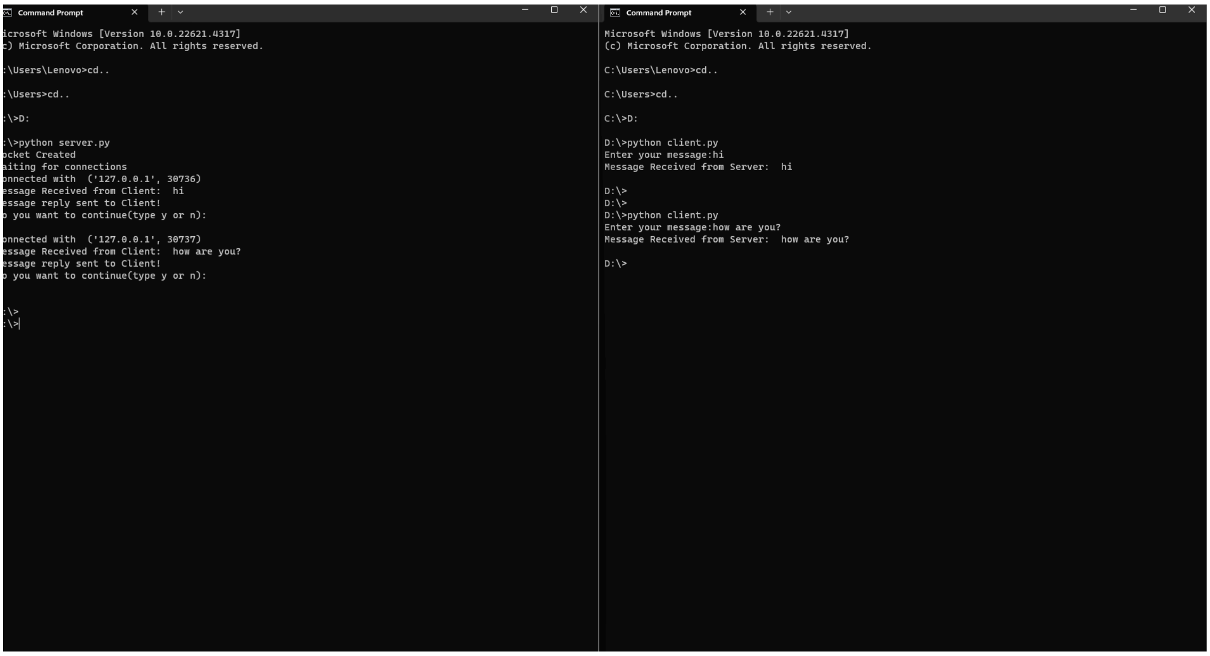
SERVER:

```
import socket
sockfd=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print('Socket Created')
sockfd.bind(('localhost',55555))
sockfd.listen(3)
print('Waiting for connections')
while True:
    clientfd,addr=sockfd.accept()
    receivedMsg=clientfd.recv(1024).decode()
    print("Connected with ",addr)
    print("Message Received from Client: ",receivedMsg)
    clientfd.send(bytes(receivedMsg,'utf-8'))
    print("Message reply sent to Client!")
    print("Do you want to continue(type y or n):")
    choice=input()
    if choice=='n':
        break
```

CLIENT:

```
import socket
clientfd=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
clientfd.connect(('localhost',55555))
name=input("Enter your message:")
clientfd.send(bytes(name,'utf-8'))
print("Message Received from Server: ",clientfd.recv(1024).decode()
```

OUTPUT:



RESULT:

The server and client successfully establish a TCP connection. The message sent by the client is received and echoed back by the server correctly.

Roll No: 241901047

Name: Keerthisri D

Exercise-3

UDP CLIENT- SERVER COMMUNICATION USING SOCKET PROGRAMMING IN PYTHON

AIM:

Server:

1. Start the program.

2.Import the socket module.

3.Create a UDP socket using socket.AF_INET and socket.SOCK_DGRAM.

4.Bind the socket to an IP address and port.

5.Receive a message from the client.

6.Send a reply back to the client.

7.Close the socket.

Client:

1. Start the program.

2.Import the socket module.

3. Create a UDP socket using socket.AF_INET and socket.SOCK_DGRAM.

4.Send a message to the server using the server’s IP and port. 5.Receive the server’s reply.

6.Display the reply message.

7.Close the socket.

CODE:

SERVER:

```
import socket
sockfd = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM) print("UDP Socket Created")
sockfd.bind(('localhost', 55555))
print("Waiting for messages on localhost:55555")
while True:
    data, addr = sockfd.recvfrom(1024)
    message = data.decode('utf-8')
    print(f"Received from {addr}: {message}")
    sockfd.sendto(data, addr)
    print(f"Echoed back to {addr}")
 choice = input("Continue receiving? (y/n): ")
 if choice.lower() == 'n':
        break
    sockfd.close()
    print("Server shutdown")
```

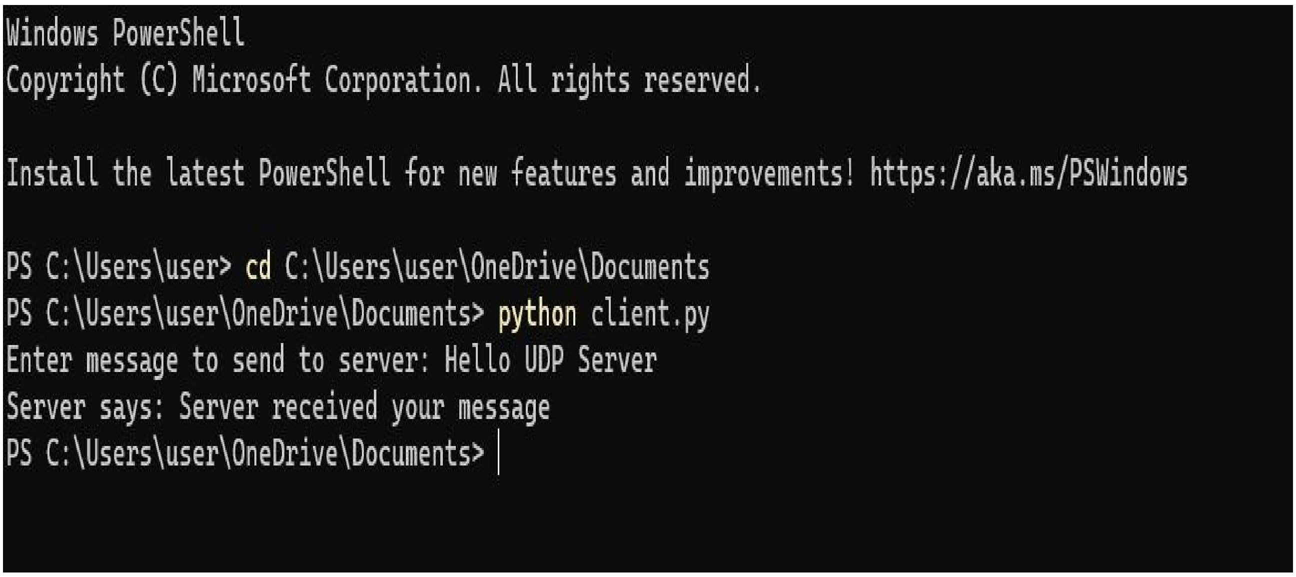
CLIENT:

```
import socket
clientfd = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM) server_addr = ('localhost',
55555)
msg = input("Enter your message: ")
clientfd.sendto(msg.encode('utf-8'), server_addr)
data, _ = clientfd.recvfrom(1024)
```

print("Echo from server:",
data.decode('utf-8')) clientfd.close()

<u>OUTPUT</u>
<u>:</u>
<u>Server:</u>

Client:



Result:

Thus, UDP server-client communication was successfully implemented using python.

`

NAME:KEERTHISRI.D

DATE:25.08.25

ROLL NO:241901047

EXERCISE-4

DEVELOP A CUSTOMIZED PING COMMAND TO TEST THE SERVER CONNECTIVITY

AIM:

To develop a customized ping command that tests the connectivity of a server by sending ICMP echo requests and receiving echo replies, measuring the response time and network availability status. This helps verify if the server is reachable and the network path is functional.

ALGORITHM:

- Set host ,port and numbers of pings.
- Create an empty list for RTT values.
- For each ping attempt:

  1.Start timer,connect to host,stop timer.

  2.Calculate RTT and store it,else print timeout

- After all attempts,display min,max and avg RTT.

CODE:

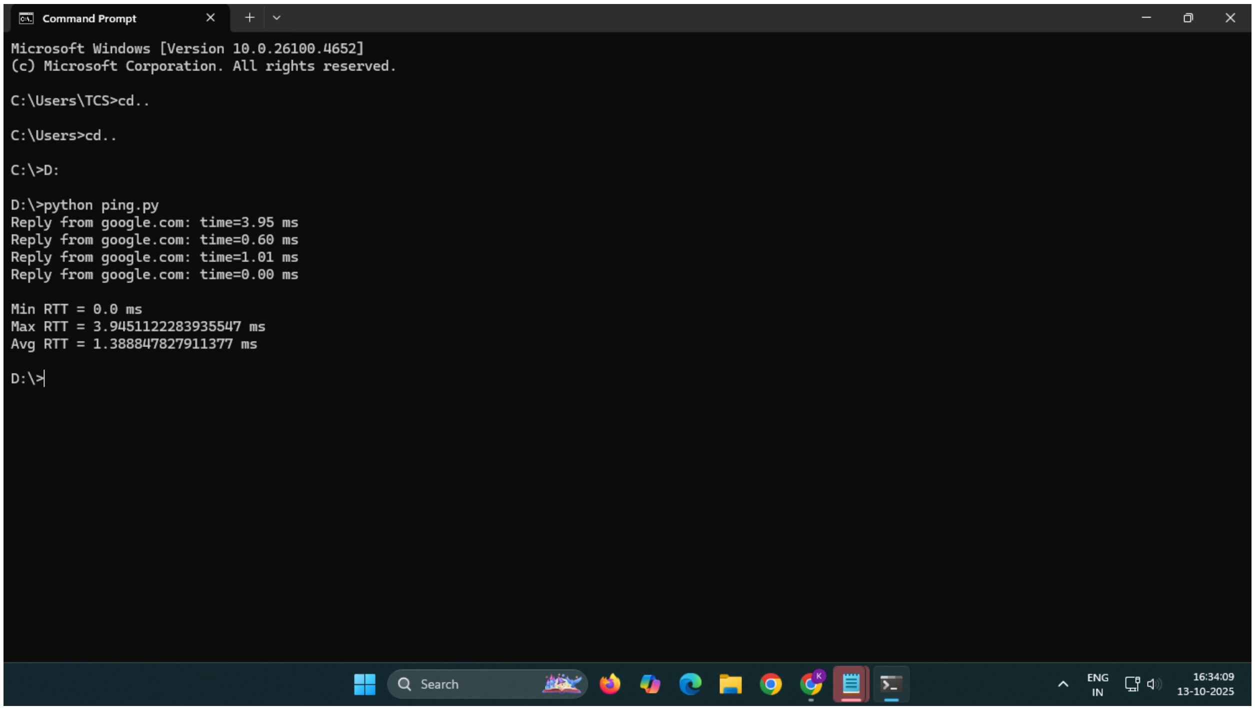
```
  import socket
import time

host = "google.com"   # you can change this
port = 80             # HTTP port
count = 4             # number of pings

for i in range(count):
    try:
        s = socket.socket()
        start = time.time()
        s.connect((host, port))
        end = time.time()
```

```
        s.close()
        print(f"Reply from {host}: time={(end-start)*1000:.2f} ms")
    except Exception:
        print("Request timed out")
```

Customized Ping Program to Measure Min, Max, and Average RTT

```
import socket, time

host = "google.com"
port = 80
count = 4
times = []

for i in range(count):
    try:
        s = socket.socket()
        start = time.time()
        s.connect((host, port))
        end = time.time()
        s.close()
        rtt = (end - start) * 1000
        times.append(rtt)
        print(f"Reply from {host}: time={rtt:.2f} ms")
    except:
        print("Request timed out")

if times:
    print("\nMin RTT =", min(times), "ms")
    print("Max RTT =", max(times), "ms")
    print("Avg RTT =", sum(times)/len(times), "ms")
```

OUTPUT:



RESULT:

The program effectively meets the objective of testing server connectivity using multiple network diagnostics methods. It enhances traditional connectivity checks and is suitable for real-world troubleshooting and monitoring tasks.

NAME:KEERTHISRI.D

DATE:26.08.25 ROLL NO:241901047

EXERCISE-5

DEVELOP A SIMPLE CALCULATOR USING XMLRPC

AIM:

To develop a simple calculator application using XML-RPC (XML Remote Procedure Call) in Python that allows a client to perform basic arithmetic operations (addition, subtraction, multiplication, division) by calling functions on a remote server.

ALGORITHM:

SERVER:

- Create functions for add, subtract, multiply, and divide.
- Set up an XML-RPC server on a specified host and port.
- Register the arithmetic functions with the server.
- Keep the server running to listen for client requests.

CLIENT:

- Connect to the XML-RPC server using its URL.
- Request an arithmetic operation by calling the corresponding remote function with two numbers.
- Receive the result from the server.
- Display the result.

CODE:

SERVER:

```
from xmlrpc.server import SimpleXMLRPCServer

def add(x, y):

   return x + y

def subtract(x, y):
```

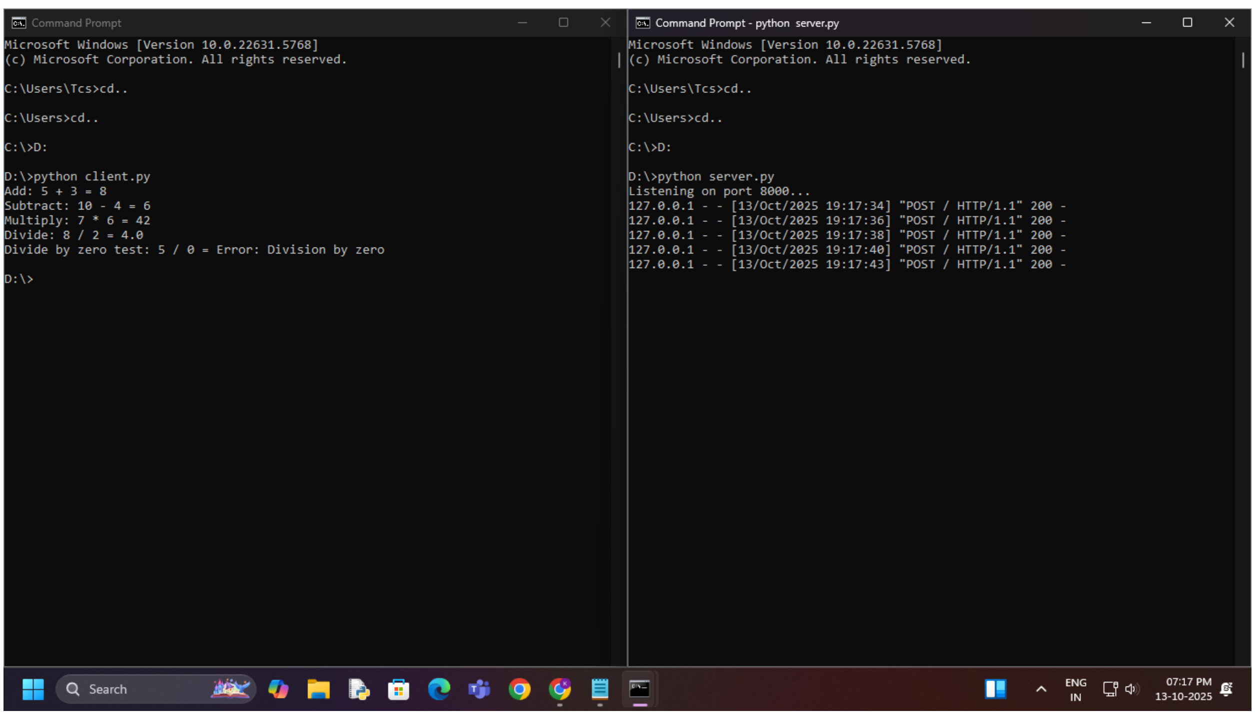
```
    return x - y
 def multiply(x, y):
    return x * y
 def divide(x, y):
    if y == 0:
        return "Error: Division by zero"
    return x / Y
server = SimpleXMLRPCServer(("localhost", 8000))
print("Listening on port 8000...")

# Register functions
server.register_function(add, 'add')
server.register_function(subtract, 'subtract')
server.register_function(multiply, 'multiply')
server.register_function(divide, 'divide')

# Run the server's main loop
server.serve_forever()
```

CLIENT:

```
import xmlrpc.client

# Connect to the server
```

```python
proxy = xmlrpc.client.ServerProxy("http://localhost:8000/")


# Example usage
print("Add: 5 + 3 =", proxy.add(5, 3))
print("Subtract: 10 - 4 =", proxy.subtract(10, 4))
print("Multiply: 7 * 6 =", proxy.multiply(7, 6))
print("Divide: 8 / 2 =", proxy.divide(8, 2))
print("Divide by zero test: 5 / 0 =", proxy.divide(5, 0))
```

OUTPUT:



RESULT:

This simple XML-RPC calculator demonstrates how a client can remotely invoke basic arithmetic operations on a server, showcasing easy inter-process communication using Python's built-in XML-RPC modules.

NAME:KEERTHISRI.D

DATE: 1.09.25

ROLL NO:241901047

EXERCISE-6

IMPLEMENT PACKET SNIFFING USING RAW SOCKETS IN PYTHON

AIM:

Build a simple Python raw-socket packet sniffer that captures Ethernet/IP packets, parses IPv4/TCP/UDP headers, and prints timestamped, human-readable summaries with short hexdumps

ALGORITHM:

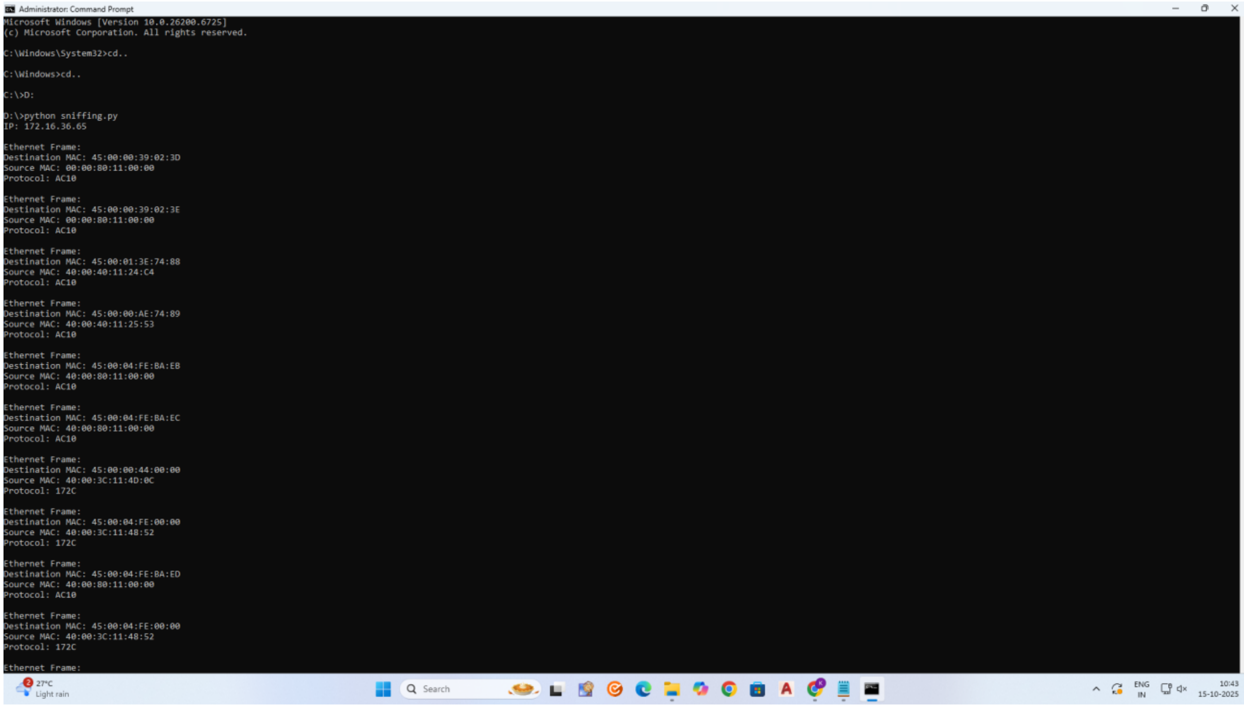
- Create a raw socket to capture all packets on the network interface (requires elevated privileges).
- Continuously receive packets from the socket in a loop.
- Parse the packet headers: Ethernet (Linux), IPv4, then TCP or UDP if applicable.
- Extract and format key information such as source/destination IPs, ports, protocol, and packet length.
- Display a timestamped summary and a short hex dump of the packet payload; repeat until stopped.

CODE:

```
import socket
import struct
import binascii
import textwrap
def main():
    # Get host
    host = socket.gethostbyname(socket.gethostname())
    print('IP: {}'.format(host))
```

```
    conn = socket.socket(socket.AF_INET, socket.SOCK_RAW,
socket.IPPROTO_IP)
    conn.bind((host, 0))


    conn.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
    # Enable promiscuous mode
    conn.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)

    while True:
        # Recive data
        raw_data, addr = conn.recvfrom(65536)

        # Unpack data
        dest_mac, src_mac, eth_proto, data = ethernet_frame(raw_data)

        print('\nEthernet Frame:')
        print("Destination MAC: {}".format(dest_mac))
        print("Source MAC: {}".format(src_mac))
        print("Protocol: {}".format(eth_proto))

# Unpack ethernet frame
def ethernet_frame(data):
    dest_mac, src_mac, proto = struct.unpack('!6s6s2s', data[:14])
    return get_mac_addr(dest_mac), get_mac_addr(src_mac),
get_protocol(proto), data[14:]

# Return formatted MAC address AA:BB:CC:DD:EE:FF
def get_mac_addr(bytes_addr):
    bytes_str = map('{:02x}'.format, bytes_addr)
    mac_address = ':'.join(bytes_str).upper()
    return mac_address

# Return formatted protocol ABCD
def get_protocol(bytes_proto):
    bytes_str = map('{:02x}'.format, bytes_proto)
    protocol = ''.join(bytes_str).upper()
    return protocol
main()
```

OUTPUT:



RESULT:

The implemented Python sniffer successfully captures live network packets, parses relevant header information, and prints readable summaries with payload hex dumps in real time.

NAME:KEERTHISRI.D

DATE:15.9.25 ROLL NO:241901047

EXERCISE-7

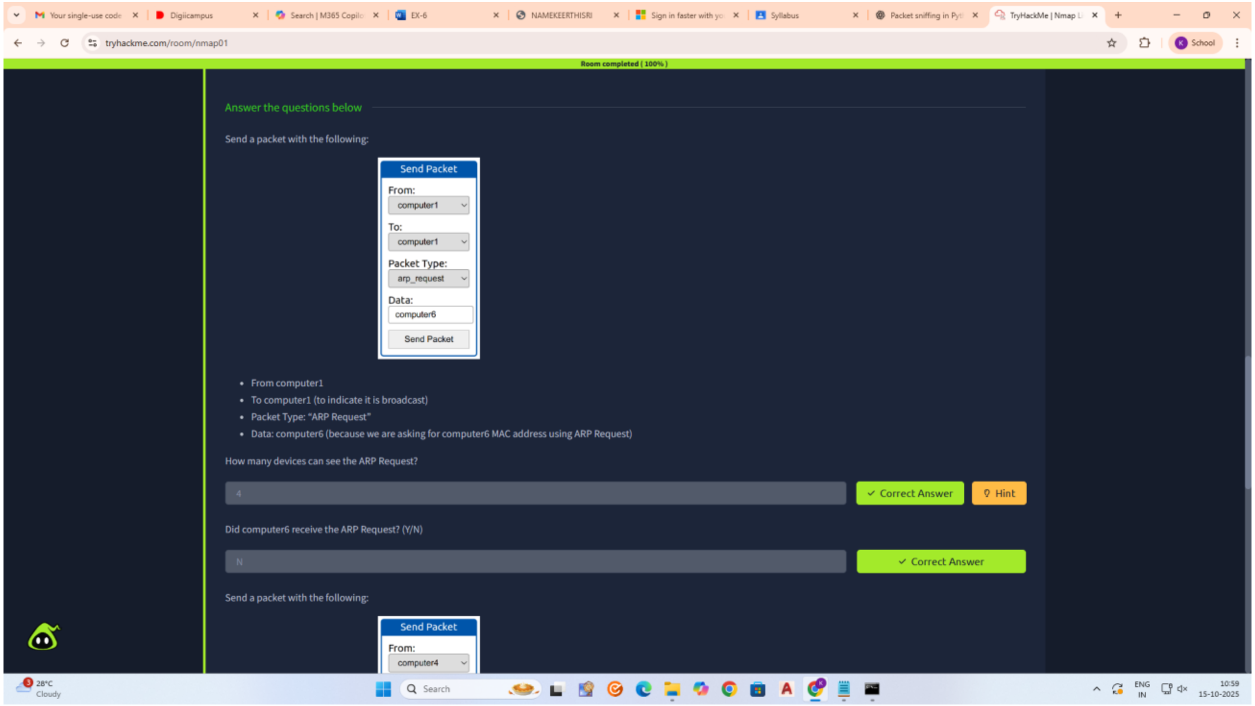
N-MAP TO DISCOVER LIVE HOSTS USING ARP REQUEST SCAN,ICMP SCAN AND TCP/UDP PING SCAN

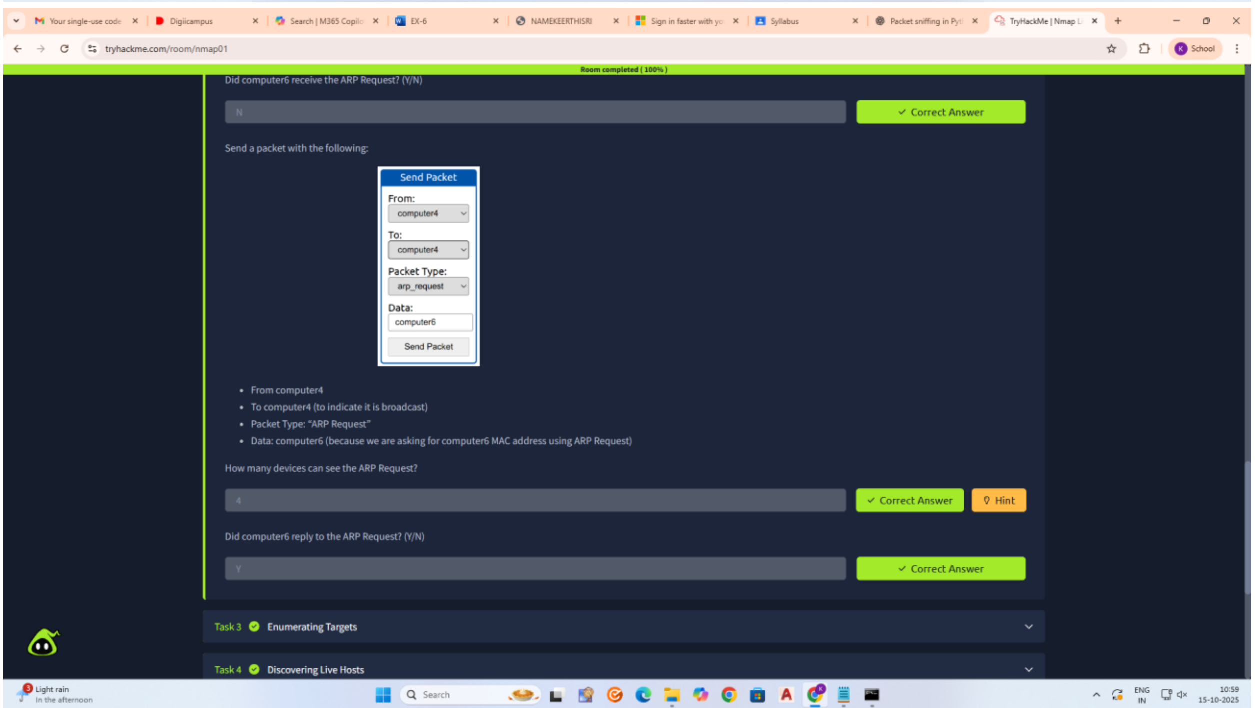
INTRODUCTION:

In this module ,we focus on discovering live host on a network using Nmap.Finding which system are online is crucial before Scanning ports or Services to save time and avoid unecesary network traffic.

TASK 2:

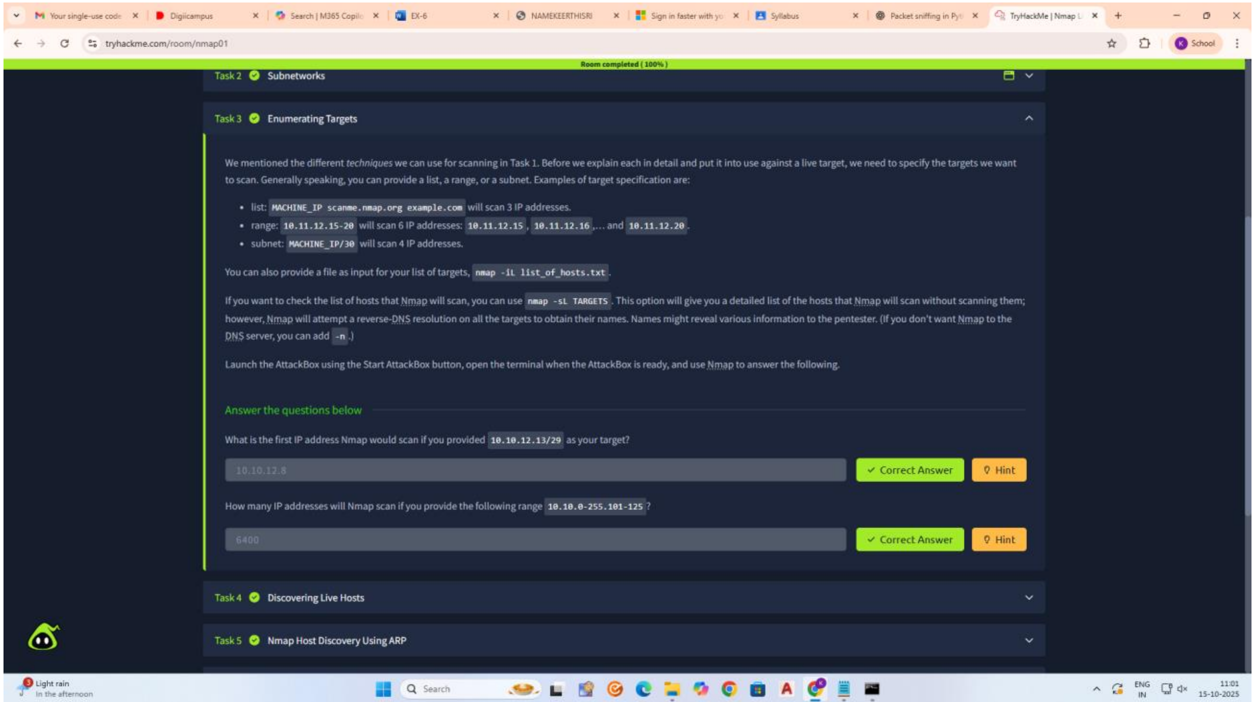
SUBNETWORKS:





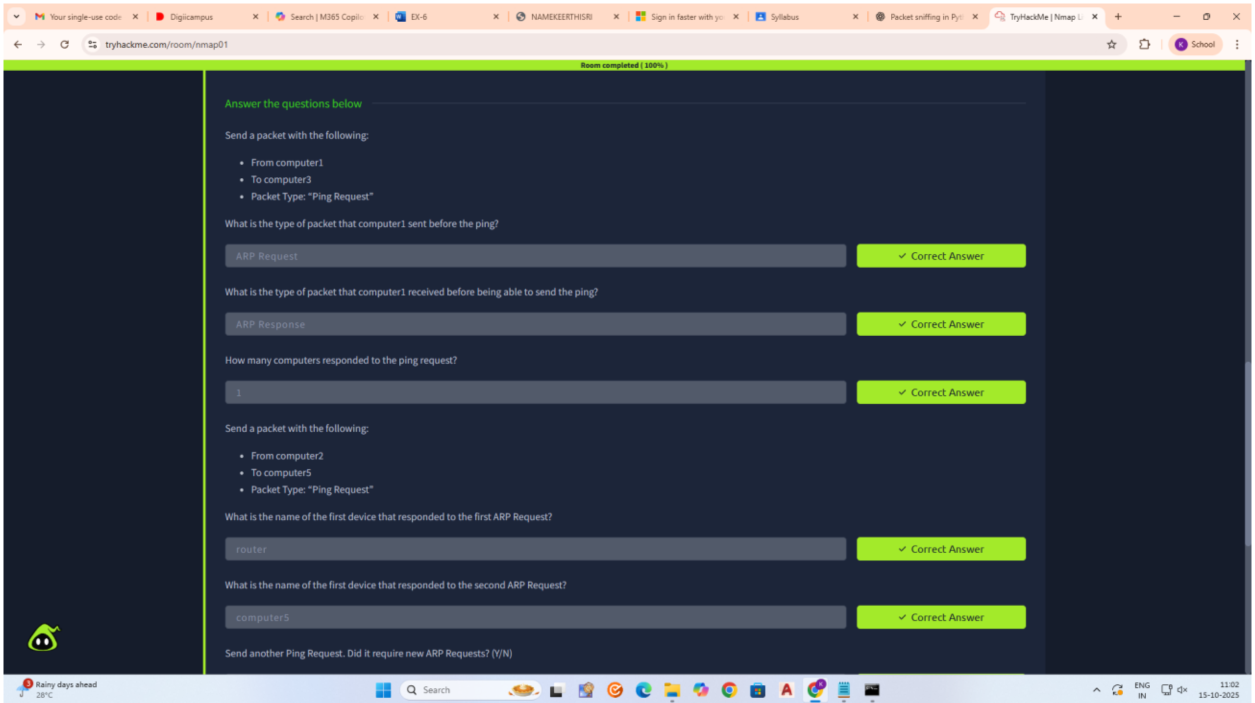
TASK 3:

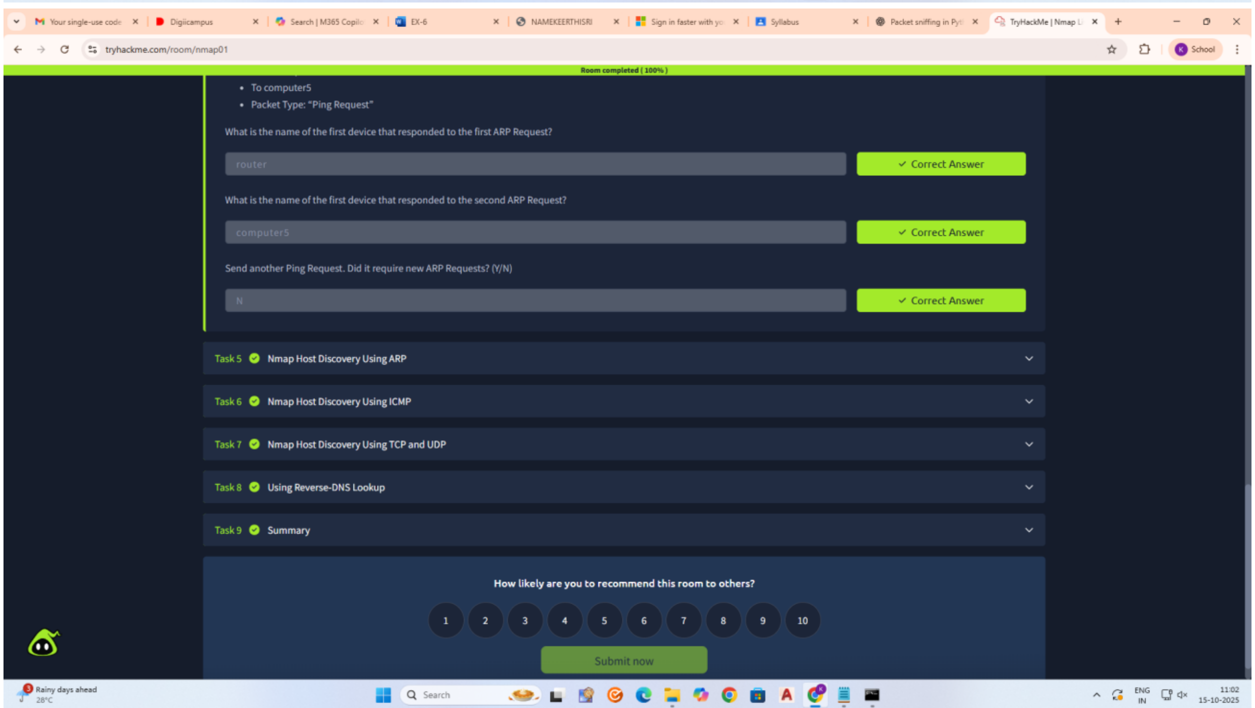
<u>ENUMERATING TARGETS:</u>



TASK 4:

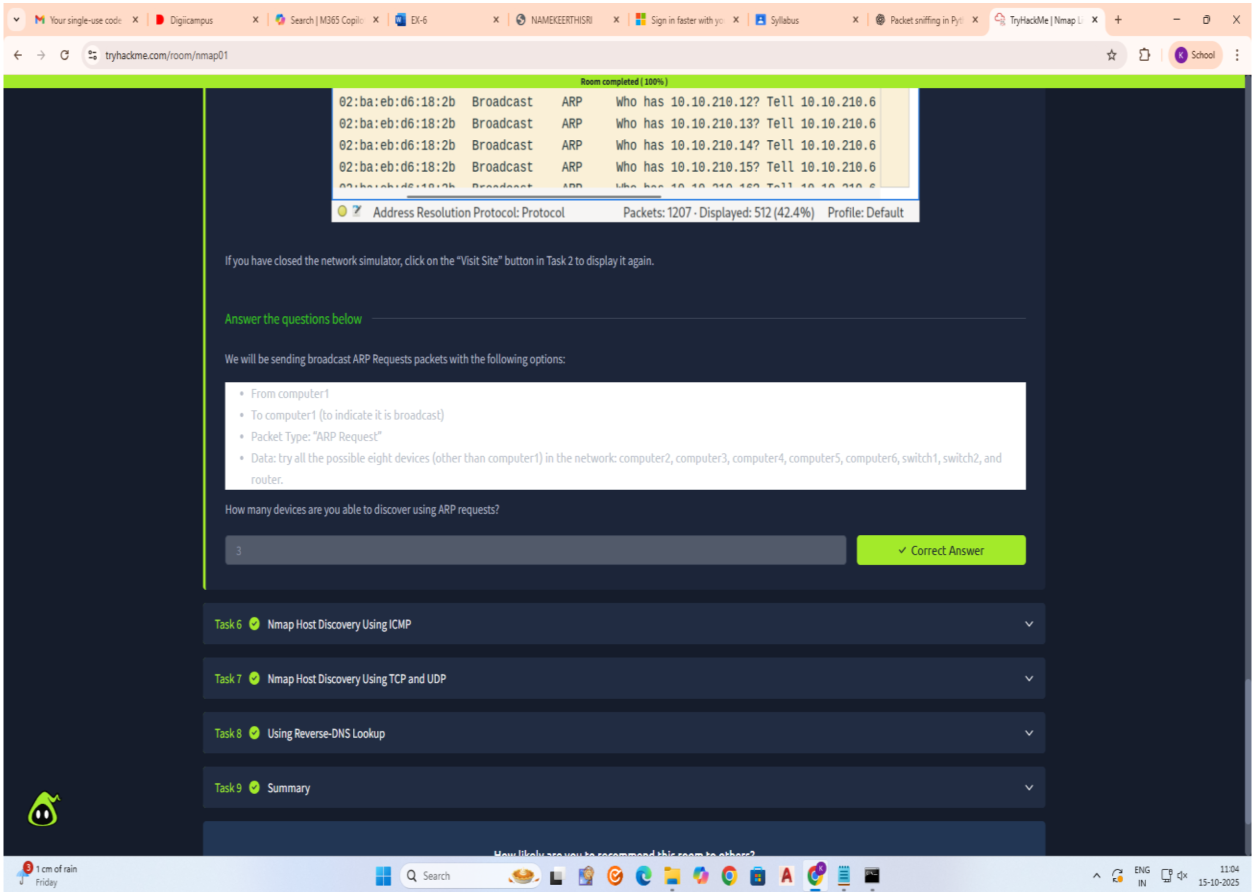
DISCOVERING LIVE HOSTS:


tryhackme.com/room/nmap01
Room completed (100%)
Answer the questions below
Send a packet with the following:
• From computer1
• To computer3
• Packet Type: "Ping Request"
What is the type of packet that computer1 sent before the ping?
ARP Request
✓ Correct Answer
What is the type of packet that computer1 received before being able to send the ping?
ARP Response
✓ Correct Answer
How many computers responded to the ping request?
1
✓ Correct Answer
Send a packet with the following:
• From computer2
• To computer5
• Packet Type: "Ping Request"
What is the name of the first device that responded to the first ARP Request?
router
✓ Correct Answer
What is the name of the first device that responded to the second ARP Request?
computer5
✓ Correct Answer
Send another Ping Request. Did it require new ARP Requests? (Y/N)
Rainy days ahead
28°C
11:02
15-10-2025


tryhackme.com/room/nmap01
Room completed (100%)
• To computer5
• Packet Type: "Ping Request"
What is the name of the first device that responded to the first ARP Request?
router
✓ Correct Answer
What is the name of the first device that responded to the second ARP Request?
computer5
✓ Correct Answer
Send another Ping Request. Did it require new ARP Requests? (Y/N)
N
✓ Correct Answer
Task 5 Nmap Host Discovery Using ARP
Task 6 Nmap Host Discovery Using ICMP
Task 7 Nmap Host Discovery Using TCP and UDP
Task 8 Using Reverse-DNS Lookup
Task 9 Summary
How likely are you to recommend this room to others?
1 2 3 4 5 6 7 8 9 10
Submit now
Rainy days ahead
28°C
11:02
15-10-2025

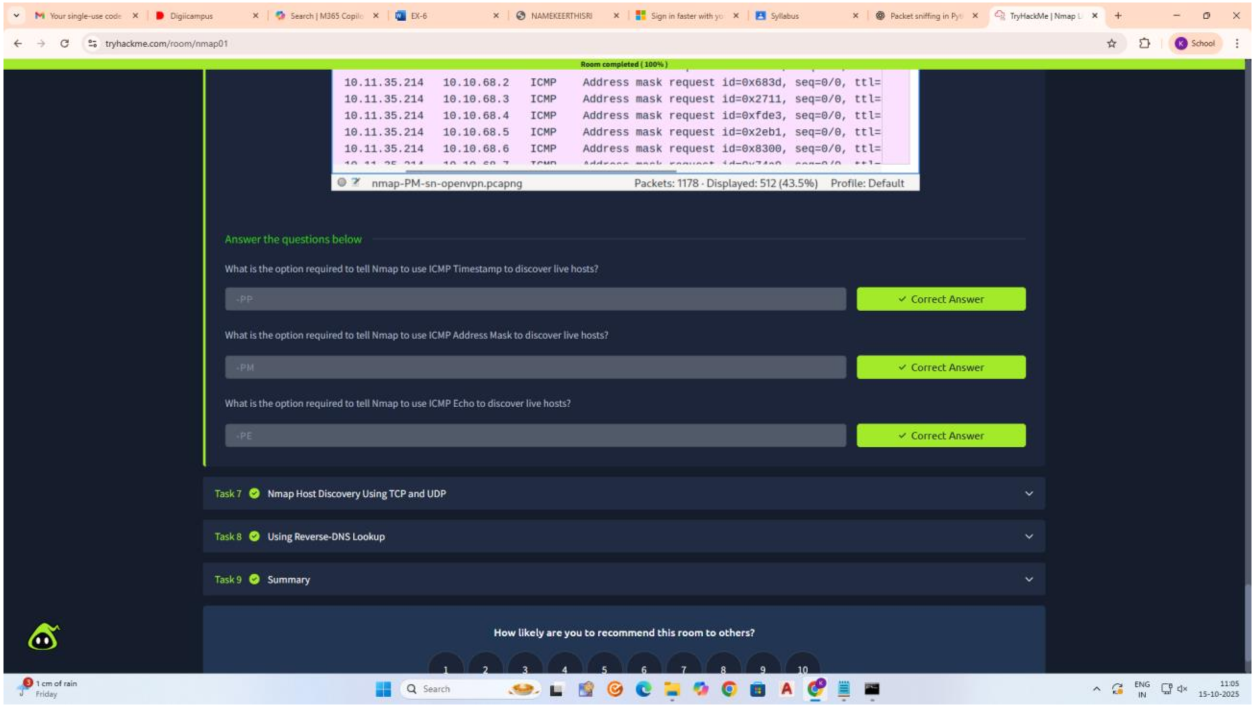
TASK 5:

NMAP HOST DISCOVERING USING ARP:



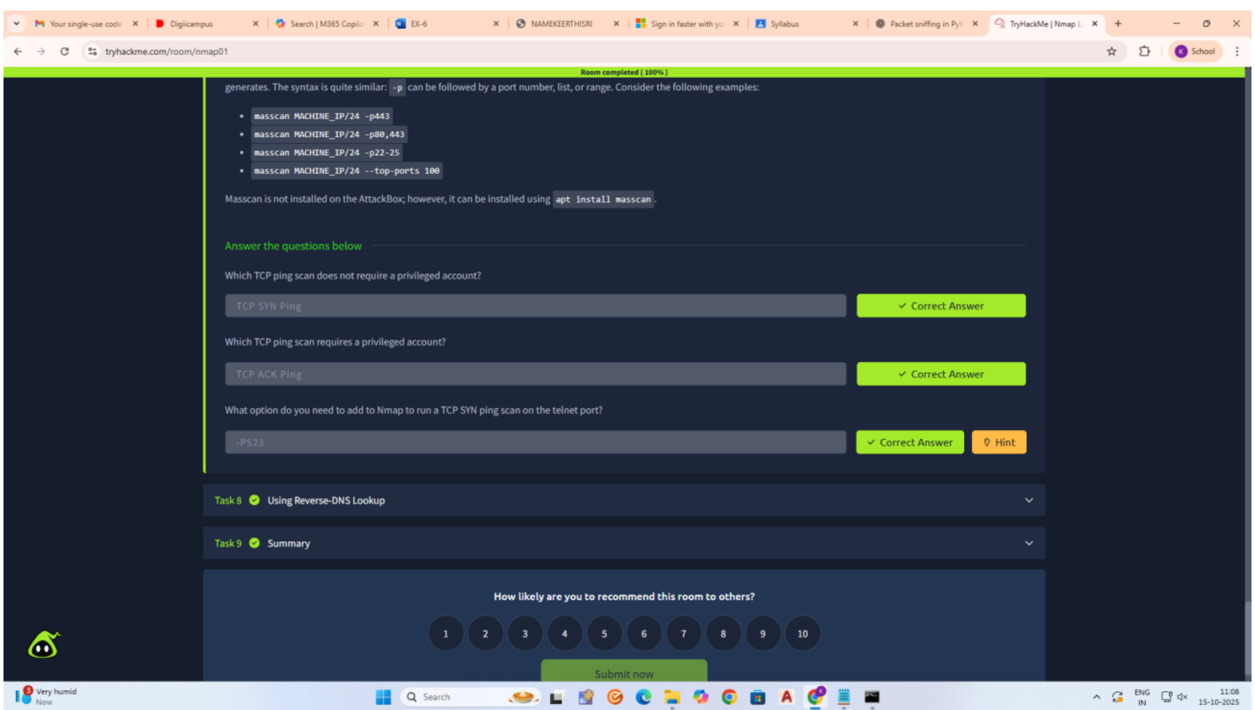
TASK 6:

NAMP HOST DISCOVERING USING ICMP:



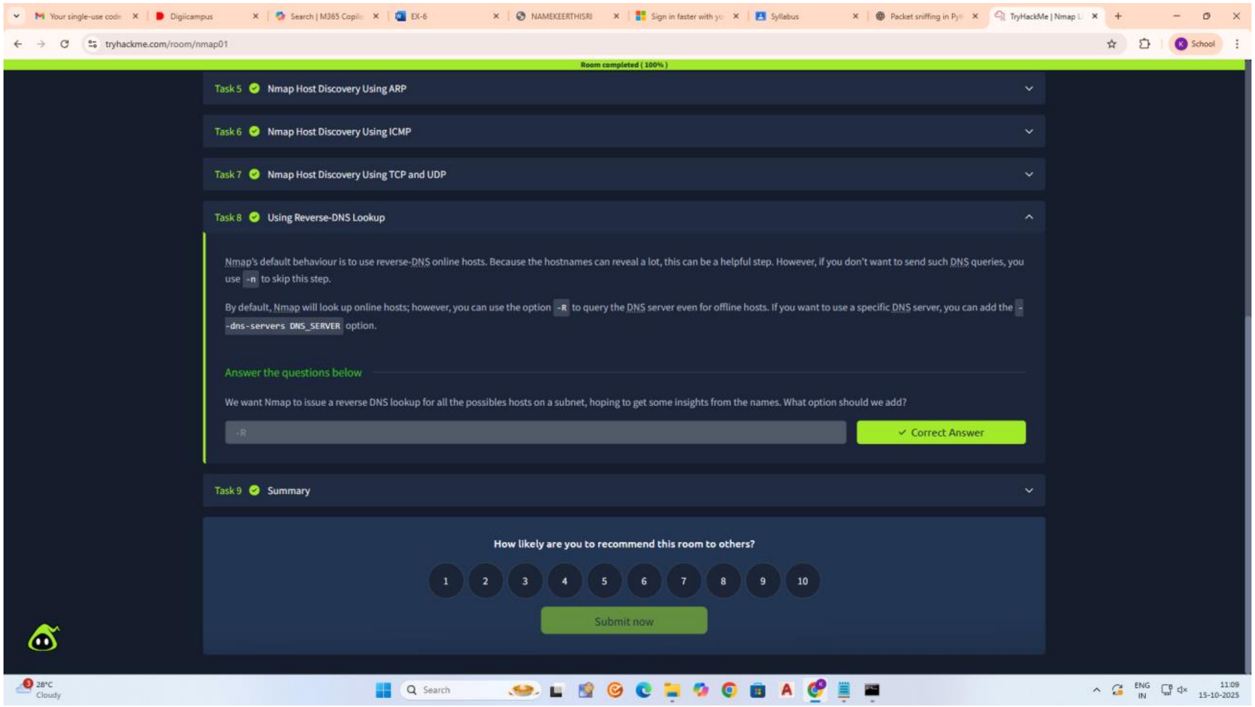
TASK 7:

NAMP HOST DISCOVERING USING TCP/UDP :



TASK 8:

USING REVERSE-DNS LOOKUP:



RESULT:

N-map detects live host using ARP ,ICMP,TCP,SYN/ACK and UDP pings DNS lookups run by default ,nut n,-R and –dns services control resolution.

NAME:KEERTHISRI.D

DATE:23.9.25 ROLL NO:241901047

EXERCISE-8

BUILDING ANONYMOUS FTP SCANNER USING FTPLIP MODULE

AIM:

Check if an FTP server allows anonymous login username “anonymous”.If successful display,the server’s welcome message and directory contents.

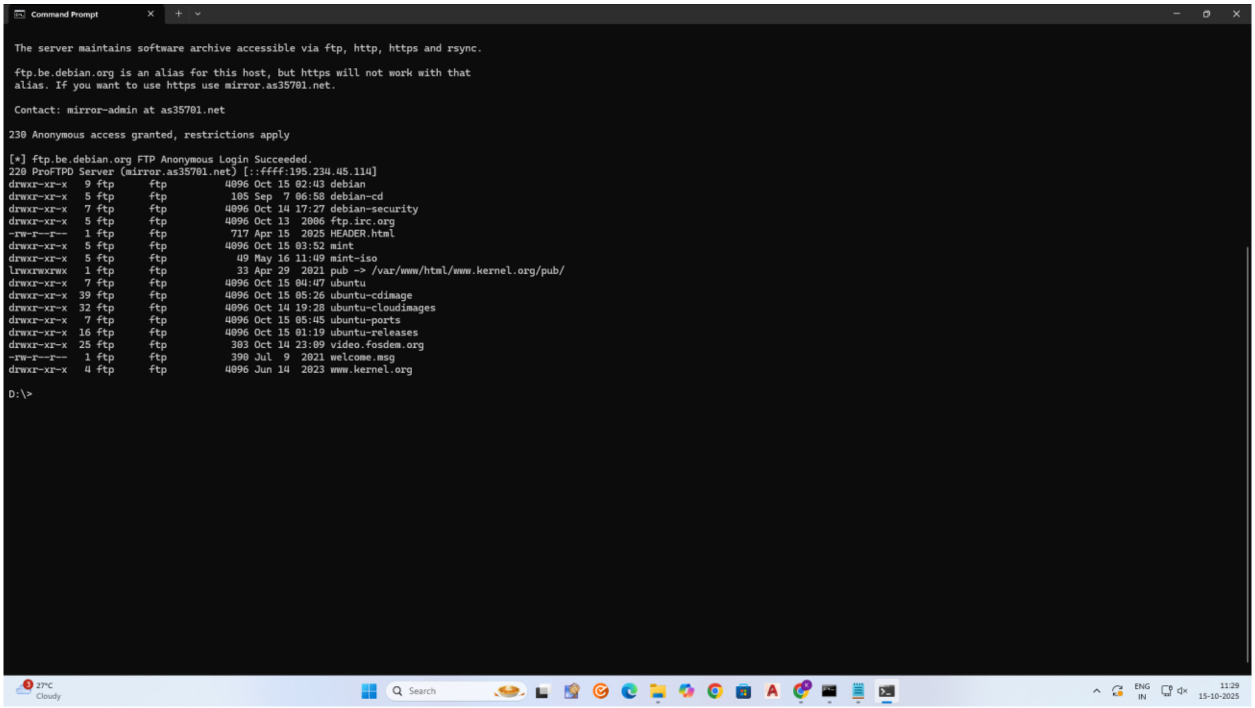
ALGORITHM:

1. Target Preparation & Safety Checks
2. Connection & Banner Fingerprinting
3. Anonymous Login Attempt
4. Non-Destructive Post-Login Probes
5. Record, Aggregate & Report Results

CODE:

```
import ftplib
def anonymousLogin(hostname):
   try:
      ftp = ftplib.FTP(hostname)
      response = ftp.login('anonymous', 'anonymous')
      print(response)
      if "230 Anonymous access granted" in response:
         print('\n[*] ' + str(hostname) +' FTP Anonymous Login Succeeded.')
         print(ftp.getwelcome())
         ftp.dir()
   except Exception as e:
      print(str(e))
      print('\n[-] ' + str(hostname) +' FTP Anonymous Login Failed.')
hostname = 'ftp.be.debian.org'
anonymousLogin(hostname)
```

OUTPUT:



RESULT:

SUCCESS: Host accepted anonymous login — an FTP session was established (may be read-only, limited, or a honeypot; login ≠ guaranteed file access).

FAILED: Connection/login failed or unreachable — covers rejected auth, no FTP service, TLS/SFTP-only, firewalls, timeouts, or transient network errors.

Roll No: 241901047

Name: Keerthisri D

Experiment: 9

DESIGN A SIMPLE TOPOLOGY AND CONFIGURE WITH ONE ROUTER, 2 SWITCHES AND PCS USING CISCO PACKET TRACER

AIM:

To design and configure a simple network topology with **one router, two switches, and multiple PCs** using **Cisco Packet Tracer** for basic communication.

ALGORITHM:

1. Start **Cisco Packet Tracer**.
2. Drag and place **1 Router**, **2 Switches**, and **4 PCs**.
3. Connect the devices using **straight-through cables**.
4. Assign **IP addresses** to all PCs and router interfaces.
5. Configure the router interfaces with the given IP addresses and enable them.
6. Save the configuration.
7. **Ping** between PCs to check network connectivity.

CODE:

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface gigabitEthernet0/0
Router(config-if)#ip address 192.168.1.1 255.255.255.0
```

Router(config-if)#no shutdown

Router(config-if)#

%LINK-5-CHANGED: Interface GigabitEthernet0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/0, changed state to up

exit

Router(config)#interface gigabitEthernet0/1

Router(config-if)#ip address 192.168.2.1 255.255.255.0

Router(config-if)#no shutdown

Router(config-if)#

%LINK-5-CHANGED: Interface GigabitEthernet0/1, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1, changed state to up

exit

Router(config)#exit

Router#

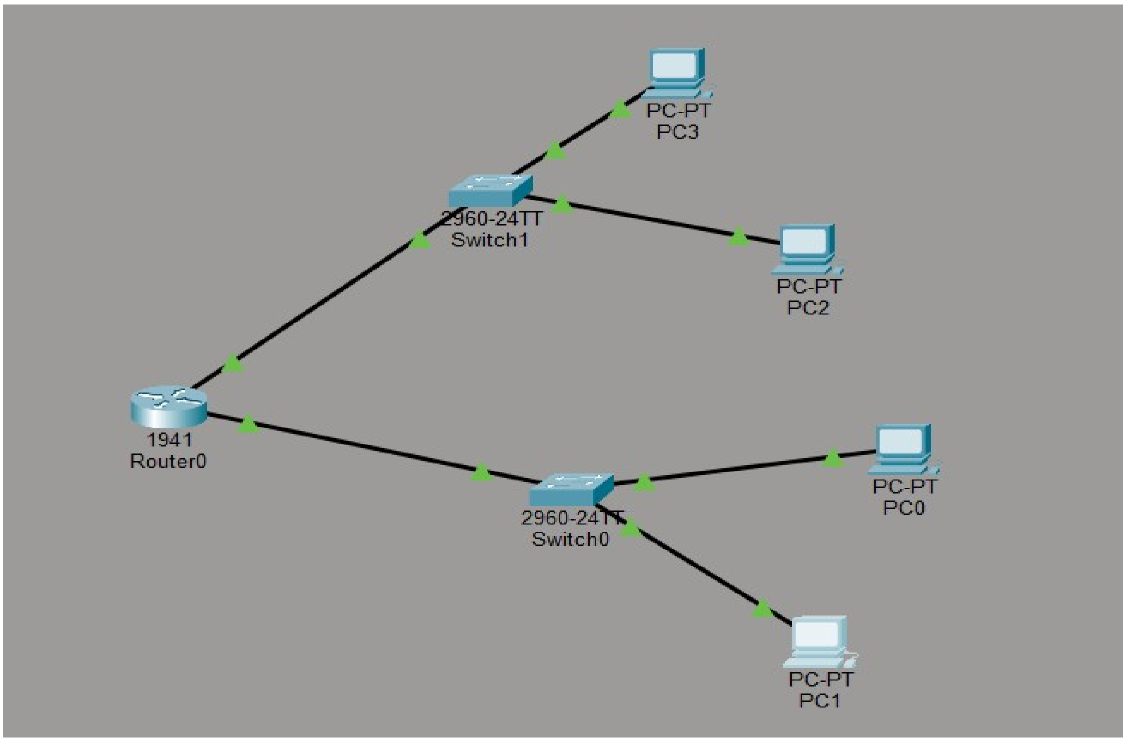
%SYS-5-CONFIG_I: Configured from console by console

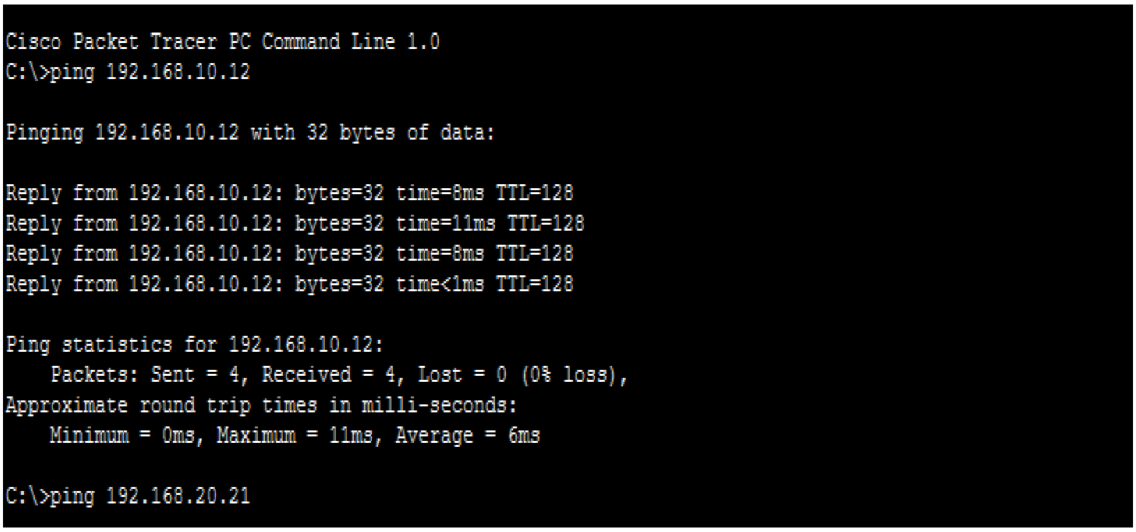
write memory

Building configuration...

[OK]

Router#





```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.10.12

Pinging 192.168.10.12 with 32 bytes of data:

Reply from 192.168.10.12: bytes=32 time=8ms TTL=128
Reply from 192.168.10.12: bytes=32 time=11ms TTL=128
Reply from 192.168.10.12: bytes=32 time=8ms TTL=128
Reply from 192.168.10.12: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.10.12:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 11ms, Average = 6ms

C:\>ping 192.168.20.21
```

RESULT:

Thus, the simple topology using router, switches and PCs are made using Cisco Packet Tracer.

NAME: KEERTHISRI D

ROLL No.: 241901047

EXERCISE 10

DESIGN A SIMPLE TOPOLOGY USING CISCO PACKET TRACER

AIM:

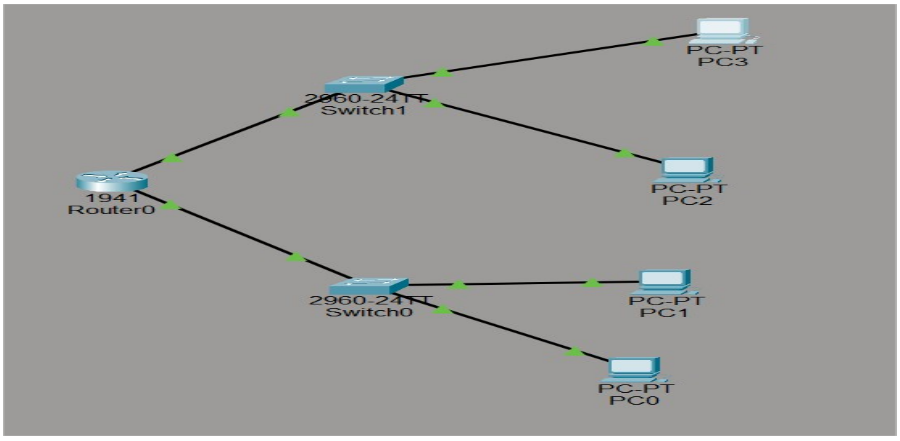
To design a simple network topology using Cisco Packet Tracer that includes one router, two switches, and multiple PCs, and to configure them for successful communication.

INTRODUCTION:

Cisco Packet Tracer is a powerful network simulation tool that allows users to design, configure, and test computer network topologies virtually. By using routers, switches, and PCs, students and professionals can build and practice real-world networking scenarios without physical hardware. In this experiment, a simple topology with one router, two switches, and multiple PCs is designed to demonstrate basic device setup and connectivity

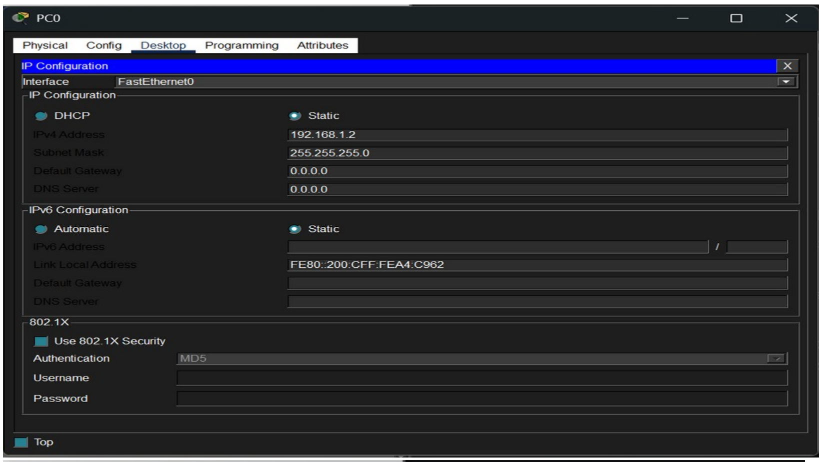
ALGORITHM:

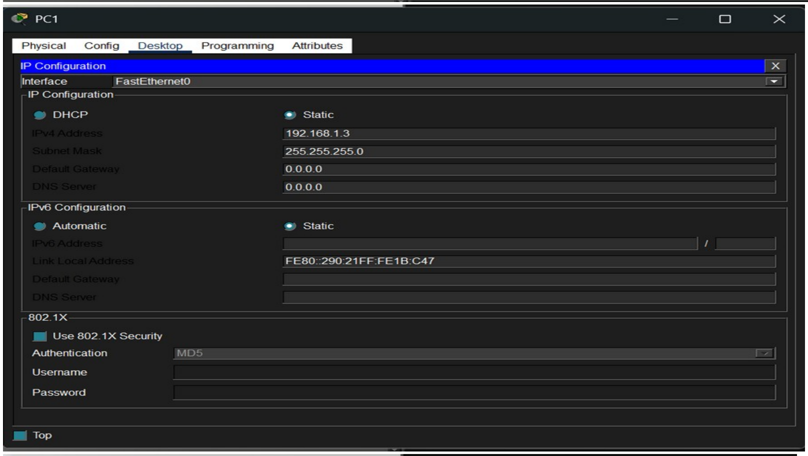
1. Open Cisco Packet Tracer and create a new project workspace.
2. Add Devices:
   - Drag one router, two switches, and multiple PCs onto the workspace.

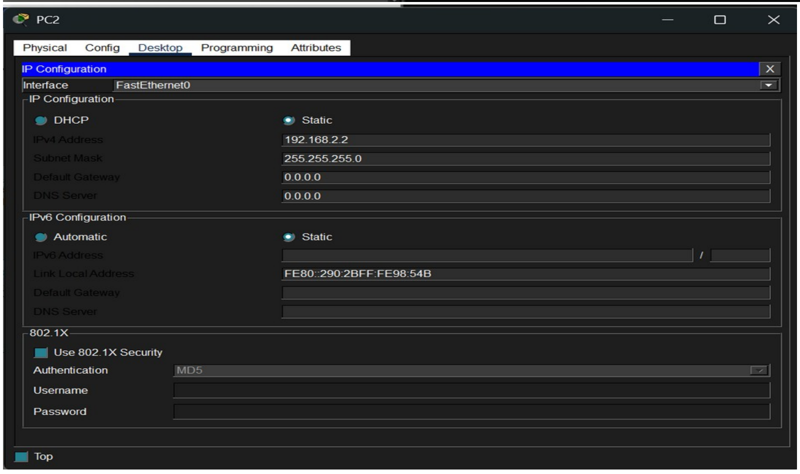


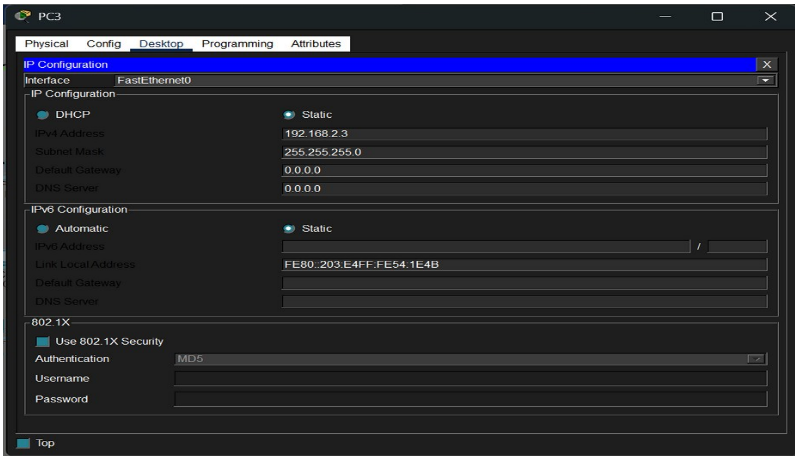
3. Connect Devices:
   - Use straight-through cables to connect each PC to a switch.
   - Connect each switch to a separate interface on the router.
4. Assign IP Addresses:

- Set appropriate IP addresses for each PC in their respective subnets.



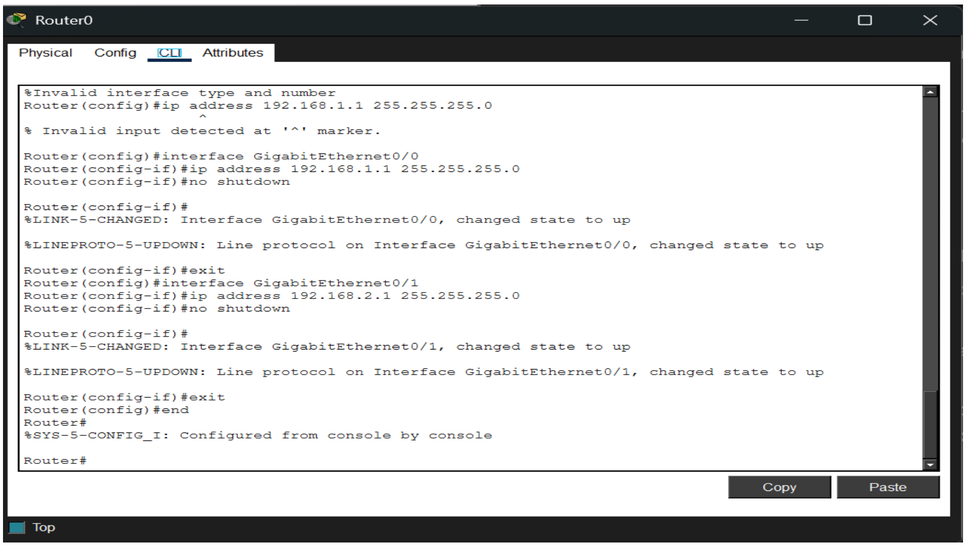






5. Configure Router Interfaces:

- Access the router CLI and assign IP addresses to each connected interface.

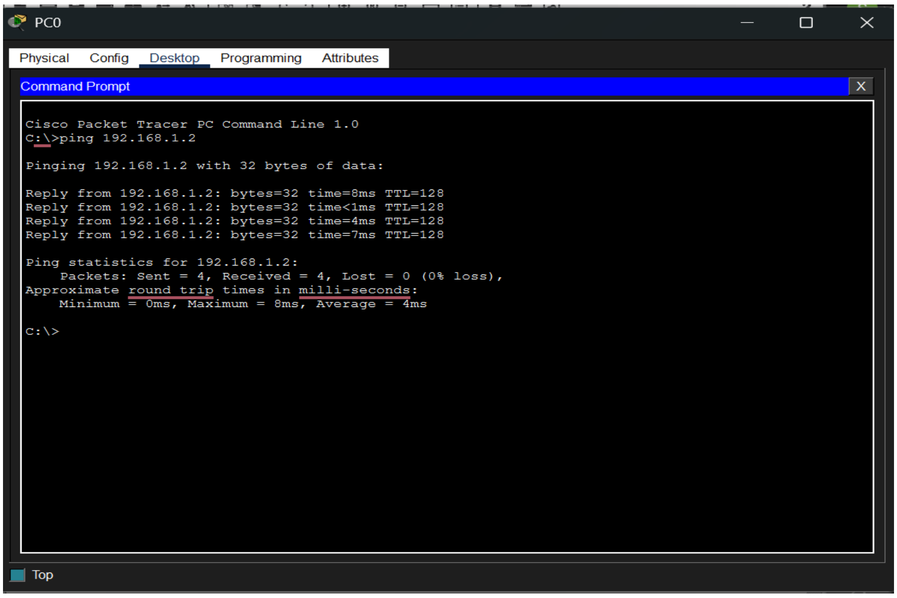


6. Set Default Gateway:

- Configure each PC’s default gateway to match the router interface IP on its subnet.

7. Verify Connectivity:

- Use the 'ping' command on PCs to check network communication across the topology.



RESULT:

The designed topology, consisting of one router, two switches, and multiple PCs, was successfully configured. All devices were able to communicate with each other across the network, demonstrating correct network setup and connectivity verification using Cisco Packet Tracer.

NAME: KEERTHISRI D

ROLL No.:241901047

EXERCISE 11

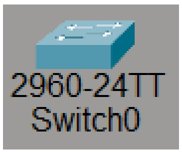
CUSTOMIZE SWITCH WITH NETWORK MODULES USING CISCO PACKET TRACER

AIM:

To customise Switch with Network Modules using Cisco Packet Tracer.

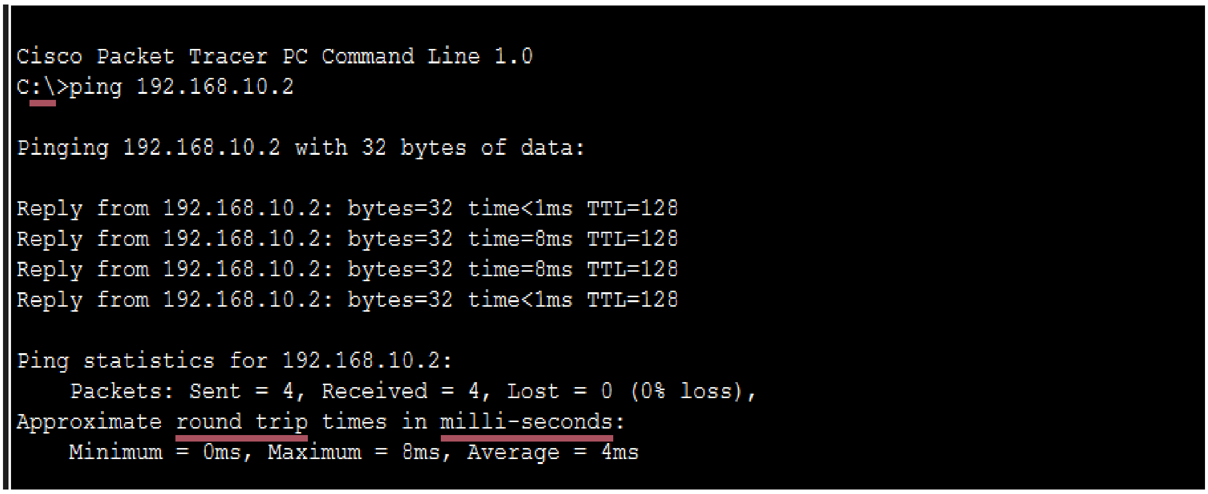
ALGORITHM:

1. Open Cisco Packet Tracer and place a switch in the workspace.



2. Click the switch to open its device window and go to the Physical tab.
3. Power off the switch by clicking the red power button.
4. Select a desired network module from the module list.
5. Drag and insert the module into an available slot on the switch chassis.
6. Power on the switch by clicking the power button again.
7. Verify the new module and interfaces are recognised in the Config or CLI tab.

```
Switch>enable
Switch#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#hostname Switch1
Switch1(config)#banner motd $ Authorized Access Only $
Switch1(config)#line console 0
Switch1(config-line)#password cisco
Switch1(config-line)#login
Switch1(config-line)#exit
Switch1(config)#enable secret class
Switch1(config)#end
Switch1#
%SYS-5-CONFIG_I: Configured from console by console
write memory
Building configuration...
[OK]
Switch1#
```



```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.10.2

Pinging 192.168.10.2 with 32 bytes of data:

Reply from 192.168.10.2: bytes=32 time<1ms TTL=128
Reply from 192.168.10.2: bytes=32 time=8ms TTL=128
Reply from 192.168.10.2: bytes=32 time=8ms TTL=128
Reply from 192.168.10.2: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.10.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 8ms, Average = 4ms
```

RESULT:

The switch was successfully customised with additional network modules, expanding its interfaces and connectivity options in Cisco Packet Tracer. The switch powered on with new modules recognised and ready for use, enabling more flexible network design and testing.

NAME: KEERTHISRI D

ROLL No.:241901047

EXERCISE 12

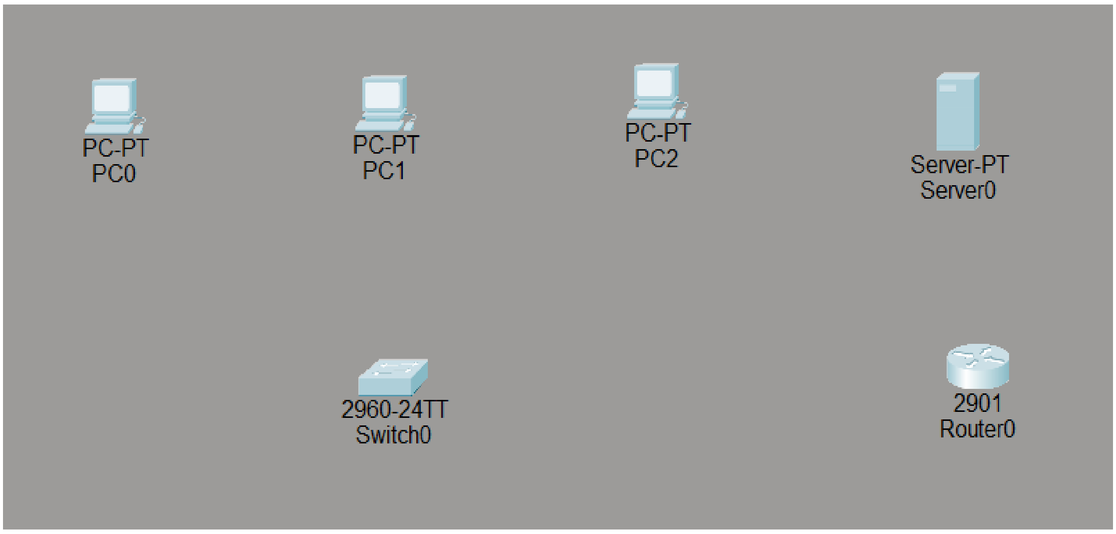
EXAMINE NETWORK ADDRESS TRANSLATION(NAT) USING CISCO PACKET TRACER

AIM:

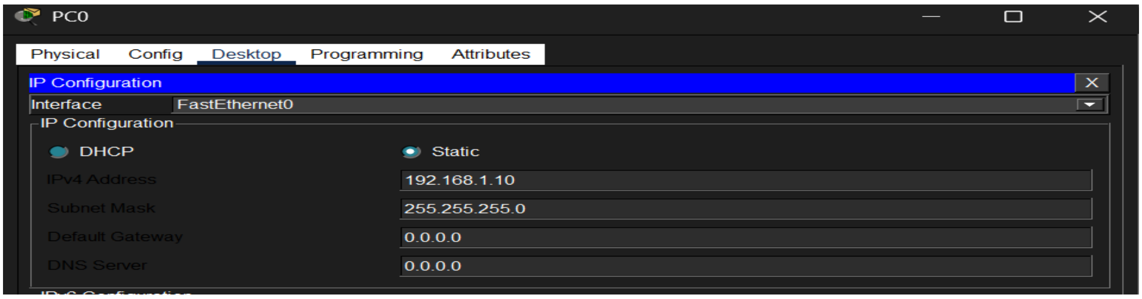
To demonstrate how NAT translates private IP addresses to public IP addresses, enabling private network devices to communicate with external networks. It also includes configuring and verifying different NAT types (Static, Dynamic, and PAT) in Cisco Packet Tracer.

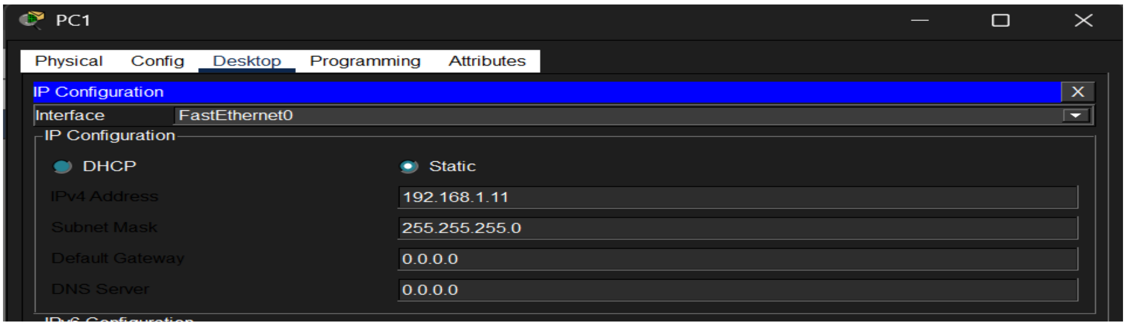
ALGORITHM:

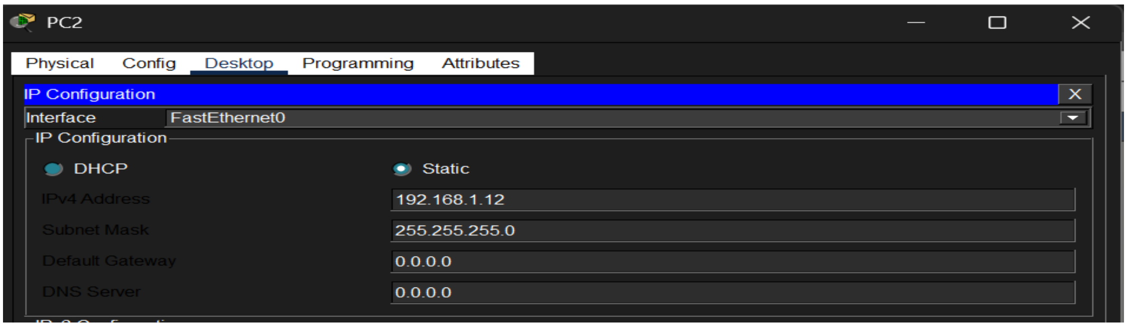
1. Build a simple topology with a router connected to a private internal network and an external network.

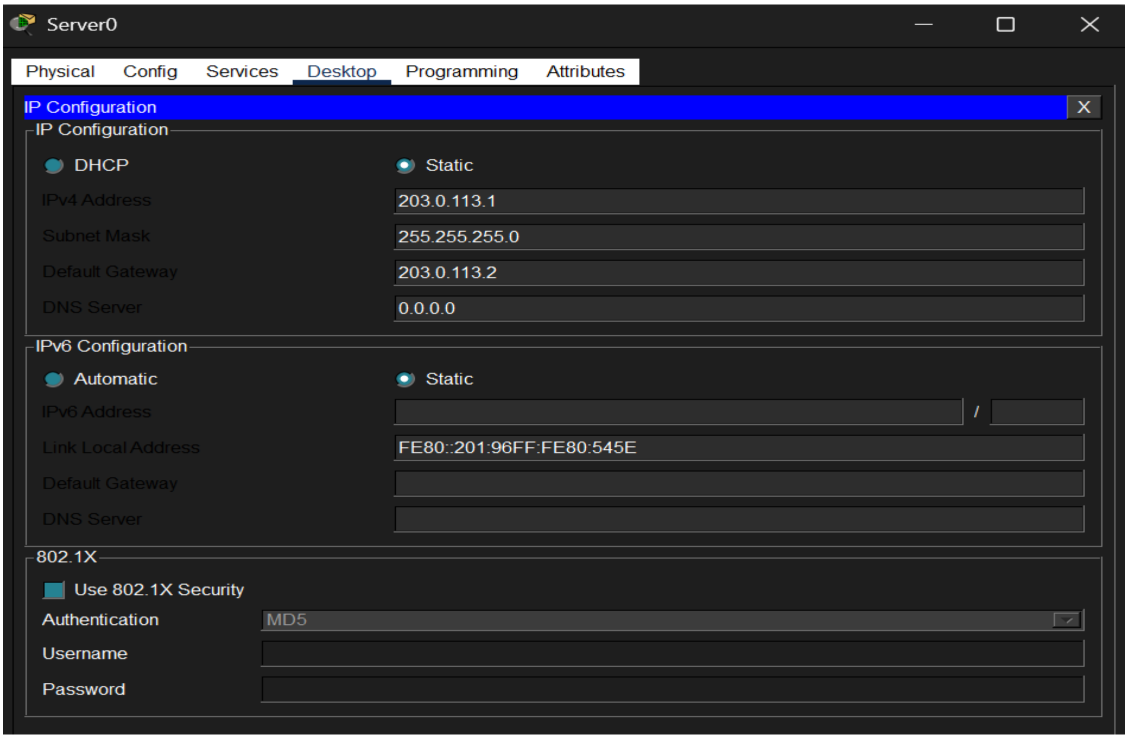


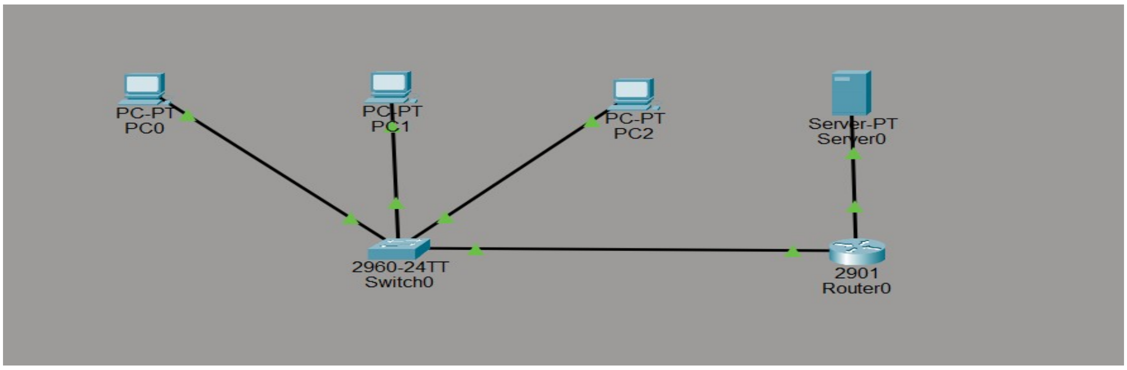
2. Assign IP addresses to all devices and router interfaces.




PC1
Physical
Config
Desktop
Programming
Attributes
IP Configuration
Interface
FastEthernet0
IP Configuration
DHCP
Static
IPv4 Address
192.168.1.11
Subnet Mask
255.255.255.0
Default Gateway
0.0.0.0
DNS Server
0.0.0.0


PC2
Physical
Config
Desktop
Programming
Attributes
IP Configuration
Interface
FastEthernet0
IP Configuration
DHCP
Static
IPv4 Address
192.168.1.12
Subnet Mask
255.255.255.0
Default Gateway
0.0.0.0
DNS Server
0.0.0.0


Server0
Physical
Config
Services
Desktop
Programming
Attributes
IP Configuration
IP Configuration
DHCP
Static
IPv4 Address
203.0.113.1
Subnet Mask
255.255.255.0
Default Gateway
203.0.113.2
DNS Server
0.0.0.0
IPv6 Configuration
Automatic
Static
IPv6 Address
/
Link Local Address
FE80::201:96FF:FE80:545E
Default Gateway
DNS Server
802.1X
Use 802.1X Security
Authentication
MD5
Username
Password


PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
Server-PT
Server0
2960-24TT
Switch0
2901
Router0

3. Configure router interfaces as NAT inside (connected to private network) and NAT outside (connected to external network).
4. Define NAT translation rules:
   - Static NAT: Map a specific internal IP to a specific external IP.
   - Dynamic NAT: Use a pool of public IP addresses for translation.
   - PAT (Port Address Translation): Use a single public IP with different port numbers.
5. Create access control lists (ACLs) to match private IP addresses allowed to be translated.
6. Enable the NAT on the router with the appropriate commands.
7. Test connectivity by pinging from inside devices to external devices or servers.
8. Verify NAT translation tables using commands like show ip nat translations.

```
Router(config-if)#interface gigabitethernet0/0
Router(config-if)#ip address 192.168.1.1 255.255.255.0
% 192.168.1.0 overlaps with GigabitEthernet0/1
Router(config-if)#no shutdown
% 192.168.1.0 overlaps with GigabitEthernet0/1
GigabitEthernet0/0: incorrect IP address assignment
Router(config-if)#exit
Router(config)#interface gigabitethernet0/0
Router(config-if)#
Router(config-if)#
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#access-list 1 permit 192.168.1.0.0.0.0.255
                                    ^
% Invalid input detected at '^' marker.

Router(config)#access-list 1 permit 192.168.1.0.0.0.0.255
                                    ^
% Invalid input detected at '^' marker.

Router(config)#access-list 1 permit 192.168.1.0.0.0.0.255
                                    ^
% Invalid input detected at '^' marker.

Router(config)#access-list 1 permit192.168.1.0.0.0.0.255
                                   ^
% Invalid input detected at '^' marker.

Router(config)#access-list 1 permit 192.168.1.0 0.0.0.255
Router(config)#ip nat inside source list 1 interface
% Incomplete command.
Router(config)#ip nat inside source list 1 interface gigabitethernet0/1 overload
Router(config)#show ip nat translations
                ^
% Invalid input detected at '^' marker.

Router(config)#end
Router#
%SYS-5-CONFIG_I: Configured from console by console

Router#show ip nat translations
Router#
```

```
Router#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0     192.168.1.1     YES manual administratively down down
GigabitEthernet0/1     192.168.1.1     YES manual up                    up
Vlan1                  unassigned      YES unset  administratively down down
Router#
```

```
Router#
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface gigabitethernet0/1
Router(config-if)#ip address 203.0.113.2 255.255.255.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface gigabitethernet0/0
Router(config-if)#ip address 192.168.1.1 255.255.255.0
Router(config-if)#no shutdown

Router(config-if)#
%LINK-5-CHANGED: Interface GigabitEthernet0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/0, changed state to up
```

RESULT:

NAT translates private IPs to public IPs for internet access. Routers keep translation tables. NAT enables private devices to communicate externally, verified through Packet Tracer simulations and commands.

Roll No: 241901047

Name: Keerthisri D

Experiment: 13

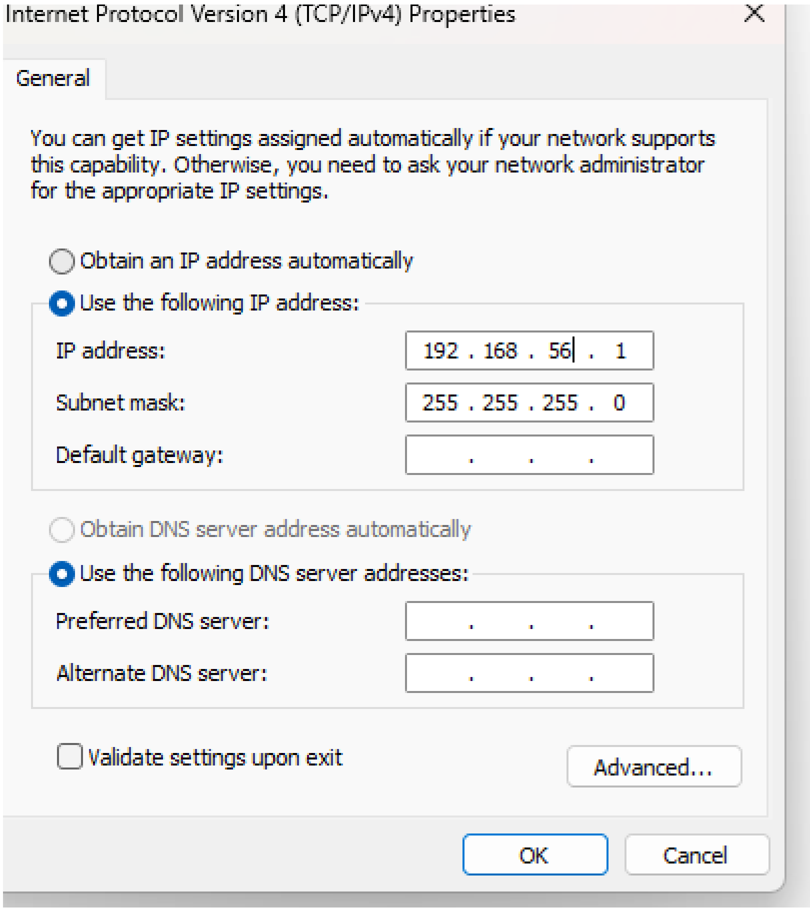
LEARNING AND ASSIGNMENT OF IP ADDRESS MANUALLY TO COMPUTERS

AIM:

To learn how to **assign IP addresses manually** to computers in a network using **Cisco Packet Tracer**.

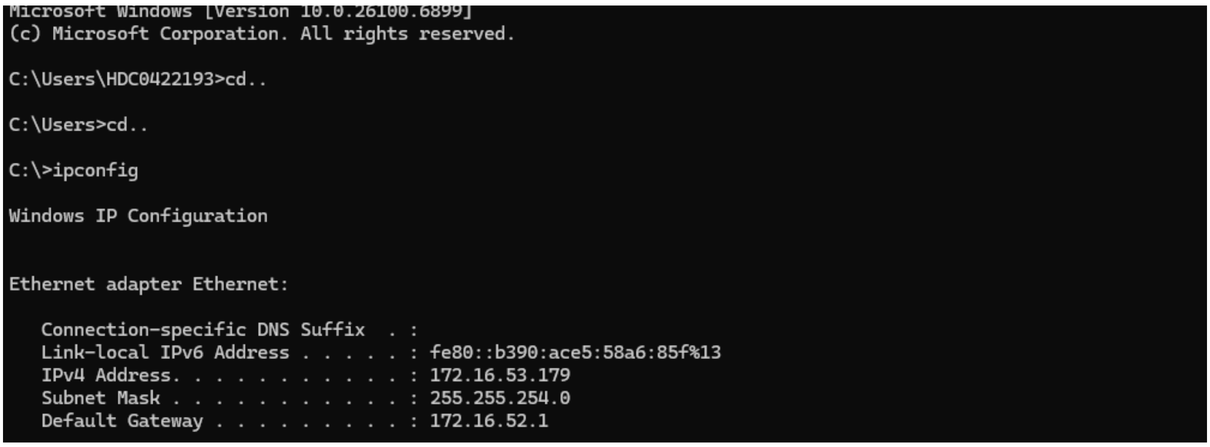
ALGORITHM:

1. Open **Cisco Packet Tracer**.
2. **Create a simple network** by connecting two or more PCs with a switch.
3. **Select a PC**, click on it, and open the **Desktop tab**.
4. Click on **IP Configuration**.
5. **Manually enter**:
    - IP Address (e.g., 192.168.1.2)
    - Subnet Mask (e.g., 255.255.255.0)
    - Default Gateway (if required, e.g., 192.168.1.1)
6. **Repeat** the process for all other PCs (assign different IPs in the same network).
7. **Test connectivity** using the **ping command** from one PC to another.
8. **Observe the results** and save the configuration.



OUTPUT:

BEFORE CHANGING



```
Microsoft Windows [Version 10.0.26100.6899]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HDC0422193>cd..

C:\Users>cd..

C:\>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::b390:ace5:58a6:85f%13
   IPv4 Address. . . . . . . . . . . : 172.16.53.179
   Subnet Mask . . . . . . . . . . . : 255.255.254.0
   Default Gateway . . . . . . . . . : 172.16.52.1
```

AFTER CHANGING

```
C:\>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::b390:ace5:58a6:85f%13
   IPv4 Address. . . . . . . . . . . : 172.16.53.179
   Subnet Mask . . . . . . . . . . . : 255.255.254.0
   Default Gateway . . . . . . . . . : 172.16.52.1

Ethernet adapter Ethernet 2:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::2739:d7ac:d7d9:4c2f%5
   IPv4 Address. . . . . . . . . . . : 192.168.56.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
```

RESULT:

Thus, the system was configured and assigned an IP address manually to the system.