THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO. 1

Aim :- Use of Crimping Tool for RJ45.

Theory :-

Cables can transmit information along their length. To actually get that information where it needs to go, you need to make right connections to a RJ45 connector. Your cable run needs to terminate into a connector, and that connector needs a jack to plug into.

Registered Jack 45 (RJ45) is a standard type of physical connector for network cables. RJ45 connectors are commonly seen with Ethernet cables and networks.

Modern Ethernet cables feature a small plastic plug on each end of the cable. That plug is inserted into RJ45 jacks of Ethernet devices. The term "plug" refers to the cable or "male" end of connection while the term "jack" refers to the port or "female" end.

T568A and T568B are the two colour codes used for writing eight-position modular plugs. Both are allowed under the ANSI/TIA/EIA wiring standards. The only difference between the two color codes is that orange and green pairs are interchanged. There is no transmission differences between T568A and T568B cabling schemes. North America's preference is for T568B. Both ends must use the same standard. It makes no difference to transmission characteristics of data.

STEP 1 :

Using a Crimping Tool, trim the end of the cable you're terminating, to ensure that ends of conducting wires are even.

STEP 2 :

Being careful not to damage inner conducting wires, strip off approximately 1 inch of cable's jacket, using a modular crimping tool or a UTP cable stripper.

STEP 3 :

Seperate 4 twisted wire pairs from each other, and then unwind each pair, so that you end up with 8 individual wires. Flatten wires out as mush as possible, since they'll need to be very straight for proper insertion into connector.

STEP 4 :

Holding cable with the wire ends facing away from you. Moving from left to right, arrange the wires in a flat, side-by-side ribbon formation, placing them in following order : white / orange, solid orange, white / green, solid blue, white / blue, solid green, white / brown, solid brown.

STEP 5 :

Holding RJ45 connector so that its pins are facing away from you and the plug-clip side is facing down, carefully insert the flattened, arranged wires into the connector, pushing through until wire ends emerge from the pins. For strength of connection, also push as much of the cable jacket as possible into the connector.

STEP 6 :

Check to make sure that wire ends coming out of connector's pin side are in the correct order; if not, remove them from connector, rearrange into proper formation, and re-insert. Remember, once the connector is crimped onto the cable, it's permanent. If you realize that a mistake has been made in wire order after termination, you'll have to cut the connector off and start all over again!

STEP 7 :

Insert prepared connector/cable assembly into RJ45 slot in your crimping tool. Firmly squeeze the crimper's handles together until you can't go any further. Release the handles and repeat this step to ensure a proper crimp.

STEP 8 :

If your crimper doesn't automatically trim the wire ends upon termination, carefully cut wire ends to make them as flush with connector's surface as possible. The closer wire ends are trimmed, better your final plug-in connection will be.

STEP 9 :

After first termination is complete, repeat process on the opposite end of your cable.

Conclusion :-

Hence, we have implemented RJ45 and CAT cabling and connection using crimping tool.

AT
CPS
8/8/23

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO. 2

Aim :- Implementation of Hamming code for Error Detection and correction.

Theory :-

Hamming code is a set of error-correction codes that can be used to detect and correct the errors that can occur when the data is moved or stored from the sender to the receiver. It is a technique developed by R.W. Hamming for error correction. Redundant bits are extra binary bits that are generated and added to information-carrying bits of data transfer to ensure that no bits were lost during the data transfer. The number of redundant bits can be calculated using the following formula:

$$2^r \geq m+r+1$$

where, r = redundant bits, m = data bits

Suppose number of data bits is 7, then number of redundant bits will be 4 ($2^4 \geq 7+4+1$)

Parity bits are used for error detection. There are two types of parity bits :

Even parity bit : In this, number of 1's are counted. If count is odd, parity bit value is set to 1, else set to 0.

Odd parity bit : In this, number of 1's are counted. if count is odd, parity bit value is set to 0. else set to 1.

General Algorithm of Hamming code :

1. Write the bit positions starting from 1 in binary form (1, 10, 11, etc).
2. All bit positions that are a power of 2 are marked as parity bits (1, 2, 4, 8, etc)
3. All the other bit positions are marked as data bits
4. Each data bit is included in a unique set of parity bits, as determined its bit position in binary form
   a. Parity bit 1 covers all bits positions whose binary representation includes a 1 in the least significant position (1, 3, 5, 7, 9, 11, etc).
   b. Parity bit 2 covers all bits positions whose binary representation includes a 1 in second position from least significant bit (2, 3, 6, 7, 10, 11) etc
   c. Parity bit 4 covers all bits positions whose binary representation includes a 1 in the third position from least significant bit (4-7, 12-15, 20-23, etc).
   d. Parity bit 8 covers all bits positions whose binary representation includes a 1 in fourth bit position from the least significant bit bits (8-15, 24-31, 40-47, etc).

e. In general, each parity bit covers all bits where the bitwise AND of the parity position and bit position is non-zero.

5. Since we check for even parity set a parity bit to 1 if total number of ones in the positions it checks is odd.

Hamming code can thereby detect and correct any single-bit error. If two data bits were flipped, it could detect it but not correct the error. Because the parity bits themselves do not have any parity data stored, if a data bit and a parity bit were flipped, it would be indistinguishable from a single-bit flip. Therefore, an additional overall parity bit is often added to reliably detect errors with 2 bits.

Conclusion :-

Hamming code is a widely used error-correction capable of correcting a single-bit error, but can only correct a limited number of multiple errors.

A+
21/8/23

```
op = int(input("Enter 1 for Hamming code generation\nEnter 2 for
error detection\n"))

if op == 1:
    m = list(map(int, input("Enter the data bits in binary:\n")))
    r = 0
    while (len(m) + r + 1) > (2 ** r):
        r += 1
    print("Total number of data bits m =", len(m))
    print("Total number of parity bits required r =", r)
    print("Total number of bits in the encoded data =", len(m) + r)
    print("The redundant bits are placed in the position", [2 ** x
for x in range(r)])

    m.reverse()
    c, ch, j, hamming = 0, 0, 0, []

    for i in range(0, (r + len(m))):
        p = (2 ** c)

        if p == (i + 1):
            hamming.append(0)
            c = c + 1
        else:
            hamming.append(int(m[j]))
            j = j + 1

    for parity in range(0, len(hamming)):
        ph = (2 ** ch)
        if ph == (parity + 1):
            startIndex = ph - 1
            i = startIndex
            y = []

            while i < len(hamming):
                block = hamming[i:i + ph]
                y.extend(block)
                i += 2 * ph

            for z in range(1, len(y)):
                hamming[startIndex] = hamming[startIndex] ^ y[z]
            ch += 1

    hamming.reverse()
    print('Hamming code generated would be:', end="")
    print(int(''.join(map(str, hamming))))

elif op == 2:
    print('Enter the received Hamming code')
    d = input()
    data = list(d)
    data.reverse()
    c, ch, h, h_copy = 0, 0, [], []
```

```
    for k in range(0, len(data)):
        p = (2 ** c)
        h.append(int(data[k]))
        h_copy.append(data[k])
        if p == (k + 1):
            c = c + 1

    parity_list = []

    for parity in range(0, len(h)):
        ph = (2 ** ch)
        if ph == (parity + 1):
            startIndex = ph - 1
            i = startIndex
            y = []

            while i < len(h):
                block = h[i:i + ph]
                y.extend(block)
                i += 2 * ph

            for z in range(1, len(y)):
                h[startIndex] = h[startIndex] ^ y[z]
            parity_list.append(h[parity])
            ch += 1
    parity_list.reverse()
    error = sum(int(parity_list) * (2 ** i) for i, parity_list in
enumerate(parity_list[::-1]))

    if error == 0:
        print('There is no error in the received Hamming code')
    elif error >= len(h_copy):
        print('Error cannot be detected')
    else:
        print('Error is in', error, 'bit')

        if h_copy[error - 1] == '0':
            h_copy[error - 1] = '1'
        elif h_copy[error - 1] == '1':
            h_copy[error - 1] = '0'
        print('After correction, Hamming code is:')
        h_copy.reverse()
        print(int(''.join(map(str, h_copy))))
else:
    print('Option entered does not exist')

'''
python -u "C:/Users/Rishab/OneDrive/Desktop/CN Experiments/import
hamm.py"
Enter 1 for Hamming code generation
Enter 2 for error detection
1
Enter the data bits in binary:
1101
Total number of data bits m = 4
Total number of parity bits required r = 3
Total number of bits in the encoded data = 7
The redundant bits are placed in the position [1, 2, 4]
Hamming code generated would be:1100110
```

```
python -u "C:/Users/Rishab/OneDrive/Desktop/CN Experiments/import
hamm.py"
Enter 1 for Hamming code generation
Enter 2 for error detection
2
Enter the received Hamming code
1100110
There is no error in the received Hamming code
python -u "C:/Users/Rishab/OneDrive/Desktop/CN Experiments/import
hamm.py"
Enter 1 for Hamming code generation
Enter 2 for error detection
2
Enter the received Hamming code
1100111
Error is in 1 bit
After correction, Hamming code is:
1100110
'''
```

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO. 3

Aim :- Implementation of CRC (Cyclic Redundancy Code) for Error Detection.

Theory :-

CRC or Cyclic Redundancy Check is a method of detecting accidental changes / errors in communication channel.

CRC uses generator polynomial is of the form like x^3+x+1 which is available on both sender and receiver side. Another example is x^2+1 that represents key 101.

Let n be number of bits in data to be sent from sender side.
Let k be number of bits in the key obtained from generator polynomial.

Sender Side :

1. The binary data is first augmented by adding k-1 zeroes in the end of the data.
2. Use modulo-2 binary division to divide the key and store remainder of division.
3. Append the remainder at the end of the data to form encoded data and send the same.

Receiver Side :

Perform modulo-2 division again and if the remainder is 0, then there are no errors.
For modulo-2 binary division, instead of subtraction, we use XOR here.
In each step, a copy of divisor is XORed with k bits of dividend. The result of the XOR operation (remainder) is (n-1) bits, which is used for next step after 1 extra bit is pulled down to make it n bits long. When there are no bits left to pull down, we have a result. The (n-1) bit remainder which is appended at sender side.

For example, data word to be sent - 100100
Key - 1101

Sender Side,
Adding n-1 = 4-1 = 3 zeroes at the end of data and taking modulo-2 division,

```
          111101
      ------------
1101 ) 100100000
       - 1101
       -------
         1000
         - 1101
         -------
           1010
           - 1101
           -------
           011010
```

```
  1101
- 0110
  0000
- 1100
  1101
-  001
```

Therefore, remainder is 001 and hence the encoded data sent is 100100001.

Now, for the Receiver Side,
Code word received at receiver side : 1001000

```
          111101
      ____________
1101 ) 100100001
       - 1101
         1000
       - 1101
          1010
        - 1101
           1110
         - 1101
            1101
          - 1101
            0000
```

Therefore, the remainder is all zeroes. Hence the data received has no error.

Conclusion :-

Thus, the cyclic redundancy check (CRC) is a technique which is used to detect the errors in the digital data.

22/8/23

```
#CRC code generation

def CRC(data, poly):


    data.extend([0] * (len(poly) - 1))
    for i in range(len(data) - len(poly) + 1):
        if data[i] == 1:
            for j in range(len(poly)):
                data[i + j] = data[i + j] ^ poly[j]

    crc = data[-len(poly) + 1:]

    return crc



data = list(map(int, input("Enter the data bits:\n")))
dup_data=data.copy()
poly = list(map(int, input("Enter the generator polynomial in binary
form:\n")))
crc_code = CRC(data,poly)
dup_data.extend(crc_code)
print("CRC code:{}".format(''.join(map(str, dup_data)) ))
# error checking
def check():
    data1 = list(map(int,input("Enter the data word received by you:
\n")))
    data1_dup=data1.copy()

    poly1 = list(map(int,input("Enter your generator polynomial in
binary form:\n")))

    for i in range(len(data1) - len(poly1) + 1):
        if data1[i] == 1:
            for j in range(len(poly1)):
                data1[i + j] = data1[i + j] ^ poly1[j]
    crc = data1[-len(poly) + 1:]

    count = 0
    for i in range(len(poly1)-1):
        if crc[i]!=0:
            count+=1
    if count==0:
        print("The codeword received has no error")
        print("The correct data bits are =",(data1_dup[0:-
len(poly1)+1]))
    else:
        print("The received code word is wrong\n")

check()
'''
python -u "C:/Users/Rishab/OneDrive/Desktop/CN Experiments/import
crc.py"
Enter the data bits:
110101
Enter the generator polynomial in binary form:
101
CRC code:11010111
```

```
Enter the data word received by you:
11010110
Enter your generator polynomial in binary form:
101
The received code word is wrong
python -u "C:/Users/Rishab/OneDrive/Desktop/CN Experiments/import
crc.py"
Enter the data bits:
1101011
Enter the generator polynomial in binary form:
101
CRC code:110101110
Enter the data word received by you:
110101110
Enter your generator polynomial in binary form:
101
The codeword received has no error
The correct data bits are = [1, 1, 0, 1, 0, 1, 1]
'''
```

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO. 4

Aim :- To simulate the Go Back N flow control algorithm.

Theory :-

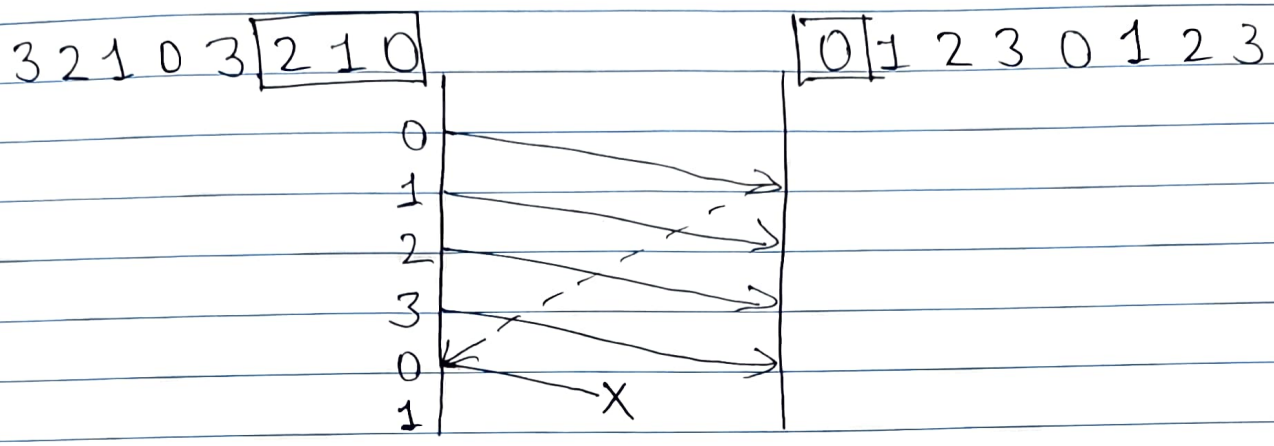
Go Back N is a type of a sliding window protocol. In this protocol, the sender can send the frames in window without receiving the acknowledgement of previously sent frame.

In the Go Back N, the window size of sender is N bits, and it means N bits can be sent without receiving acknowledgement of first frame. The receiver's window size is of 1 bit.

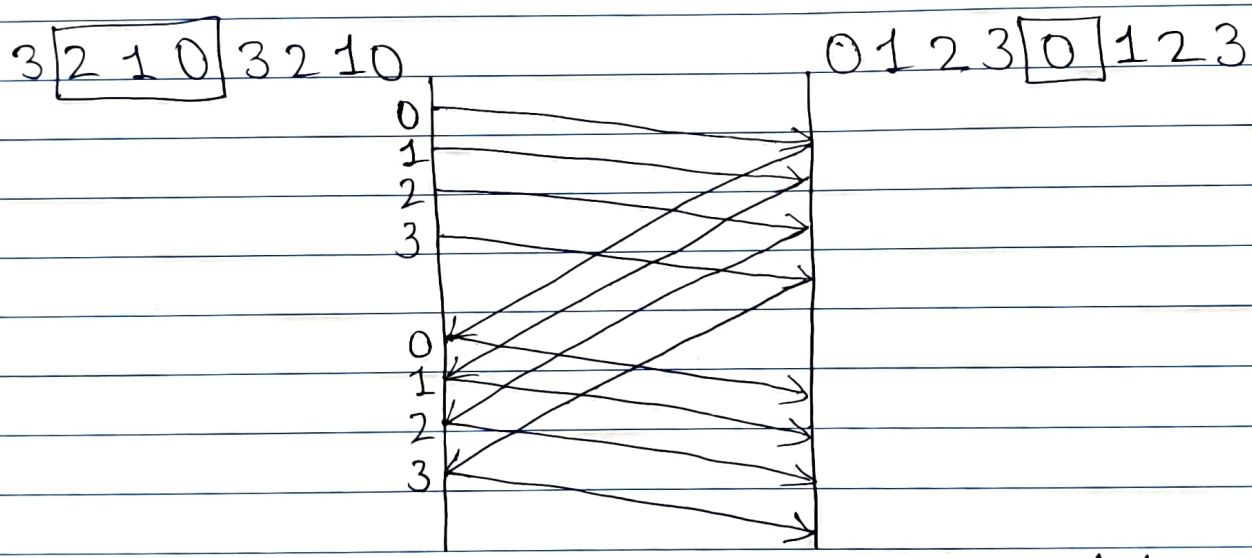
For Go Back N protocol, if there are k bits in sequence numbers, the sender's window size is of $2^k - 1$ bits. If sender does not receive the acknowledgement, it leads to retransmission of all the current window frames.

If the window size of sender is set to 2^k bits, then data loss can occur if the first bit of sequence gets lost, then the sequence on the receiver's end gets ahead of the data from sender's side.

Example :- k=2, data = 01230123
Window size of sender = $2^k - 1 = 3$



Frame 0 was lost, so the entire window consisting of 0, 1 and 2 will be retransmitted.



The window was resent and the complete data got transmitted successfully.

Conclusion :- Implemented the Go Back N ARQ flow control algorithm.

```
import random
import time

total_frames = int(input("Enter the total number of frames: "))
window_size = int(input("Enter the Window Size: "))
total_transmissions = 0
random.seed(time.time())

current_frame = 1
while current_frame <= total_frames:
    transmitted_frames = 0
    for frame in range(current_frame, min(current_frame +
window_size, total_frames + 1)):
        print("Frames sent", frame)
        total_transmissions += 1

    for frame in range(current_frame, min(current_frame +
window_size, total_frames + 1)):
        flag = random.randint(0, 1)
        if not flag:
            print("Acknowledgment for Frame", frame)
            transmitted_frames += 1
        else:
            print("Frame", frame, "Not Received")
            print("Retransmitting Window")
            break

    print()
    current_frame += transmitted_frames

print("Total number of transmissions:", total_transmissions)

'''
python -u "C:/Users/Rishab/OneDrive/Desktop/CN Experiments/import
random.py"
Enter the total number of frames: 4
Enter the Window Size: 3
Frames sent 1
Frames sent 2
Frames sent 3
Acknowledgment for Frame 1
Frame 2 Not Received
Retransmitting Window
Frames sent 2
Frames sent 3
Frames sent 4
Frame 2 Not Received
Retransmitting Window
Frames sent 2
Frames sent 3
Frames sent 4
Acknowledgment for Frame 2
Frame 3 Not Received
Retransmitting Window
Frames sent 3
Frames sent 4
Frame 3 Not Received
Retransmitting Window
```

```
Frames sent 3
Frames sent 4
Frame 3 Not Received
Retransmitting Window
Frames sent 3
Frames sent 4
Acknowledgment for Frame 3
Acknowledgment for Frame 4
Total number of transmissions: 15
'''
```

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO. 5

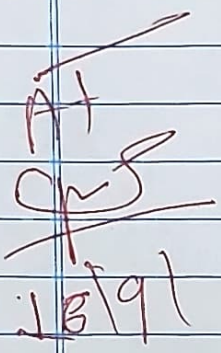
Aim :- To build a simple network topology and configure it for static routing protocol using packet tracer. Setup a network and configure IP addressing, subnetting, masking.

Theory :-

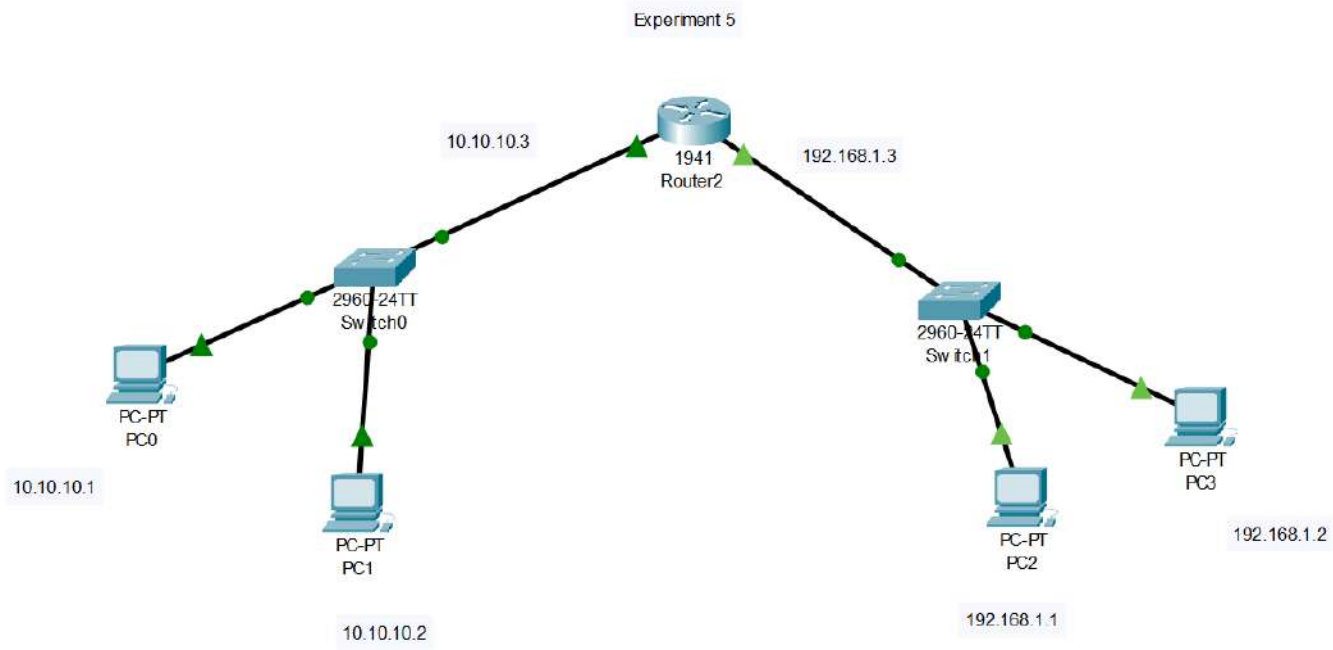
Cisco Packet Tracer is a cross-platform visual simulation tool designed by Cisco Systems that allows user to create network topologies and imitate modern computer networks. The software allows users to simulate the configuration of Cisco routers and switches using a simulated command line interface. CPT uses a drag and drop user interface, allowing users to add and remove simulated network devices as they see fit. The software is used to allow the users to create simulated network topologies by dragging and dropping routers, switches and various other types of network devices.

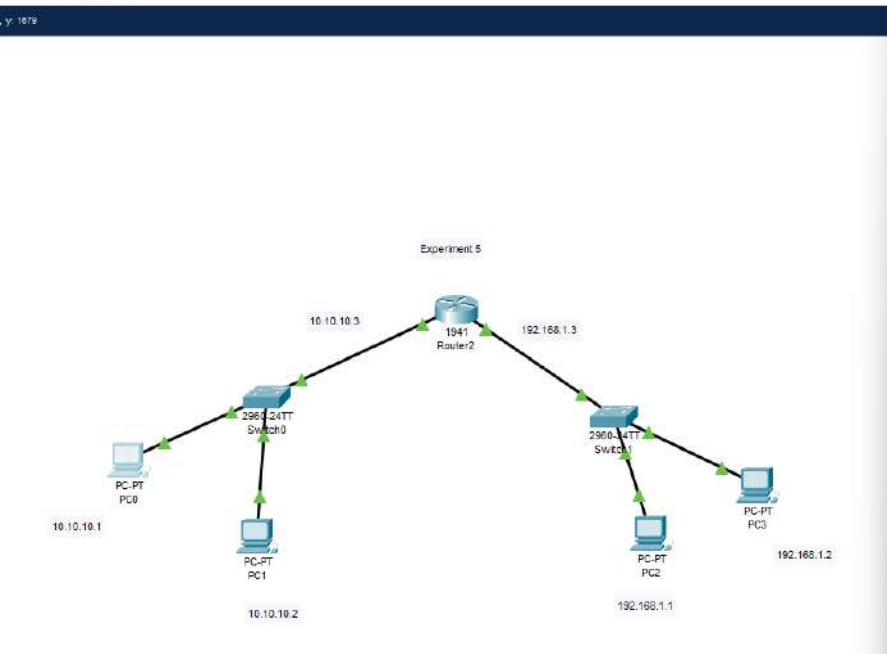
Cisco Packet Tracer allows users to simulate and experiment with computer networks without the need for physical hardware.
It provides a virtual environment where users can create and configure networks, devices, and protocols.

The tool includes a wide range of Cisco networking devices such as routers, switches and access points. Cisco Packet Tracer supports various networking protocols and services, including TCP/IP, DHCP, DNS, FTP, HTTP and more.
Users can configure and test these protocols in a controlled environment.
Users can design complex network topologies by connecting devices and creating virtual LANs (VLANs). It is a valuable tool to plan and visualize network designs.



16/9/23


Experiment 5
10.10.10.3
1941
Router2
192.168.1.3
2960-24TT
Switch0
2960-24TT
Switch1
PC-PT
PC0
10.10.10.1
PC-PT
PC1
10.10.10.2
PC-PT
PC2
192.168.1.1
PC-PT
PC3
192.168.1.2



PC0

Physical | Config | Desktop | Programming | Attributes

Command Prompt

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

FastEthernet0 Connection:(default port)

   Connection-specific DNS Suffix..:
   Link-local IPv6 Address.....: FE80::2E0:A3FF:FEE4:E9CB
   IPv6 Address.....: ::
   IPv4 Address.....: 10.10.10.1
   Subnet Mask.....: 255.0.0.0
   Default Gateway.....: ::
                                 10.10.10.3

Bluetooth Connection:

   Connection-specific DNS Suffix..:
   Link-local IPv6 Address.....: ::
   IPv6 Address.....: ::
   IPv4 Address.....: 0.0.0.0
   Subnet Mask.....: 0.0.0.0
   Default Gateway.....: ::
                                 0.0.0.0

C:\>ping 192.168.1.1

Pinging 192.168.1.1 with 32 bytes of data:

Request timed out.
Reply from 192.168.1.1: bytes=32 time<1ms TTL=127
Reply from 192.168.1.1: bytes=32 time<1ms TTL=127
Reply from 192.168.1.1: bytes=32 time<1ms TTL=127

Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO. 6

Aim :- Design VPN and configure the RIP using Cisco Packet tracer.

Theory :-

Routing Information Protocol (RIP) :-

RIP is a dynamic routing protocol that uses hop count as a routing metric to find the best path between the source and the destination network. It is a distance-vector routing protocol that has been working on Network layer of the OSI model. It is majorly used for small to medium-sized networks.

Steps to configure RIP routing in Cisco Packet Tracer :

1. Create a Network Topology :
   Launch Cisco Packet Tracer and create a network topology. You can add routers and switches to workspace and connect them using appropriate cables.

2. Configure device interfaces :
   Setup IP addresses, networks and gateway addresses for the devices present in the workspace.

3. Configure Router Interfaces :

Access the router's CLI and configure the interfaces of the router. For example, if you have two routers connected through their Fast Ethernet interfaces, configure them with IP addresses.

4. Enable RIP Routing:
Enable RIP Routing after entering the configuration mode and repeat the same on Router 2.

5. Test Connectivity:
After configuring RIP on both routers, you should be able to ping devices on remote networks. Test connectivity between devices connected to different routers:

6. Save the configuration and test:
Test the networks in the workspace (both the realtime and simulation mode).

VLAN :-

Configuring Virtual LANs (VLANs) in Cisco Packet Tracer and create a network topology that includes switches and devices. You can use the physical workspace to drag and drop switches and connect them using appropriate cables.

Steps to configure VLANs in CPT :

1. Create a Network Topology :
Launch Cisco Packet Tracer and create a network topology that includes switches and devices, and connect them using appropriate cables.

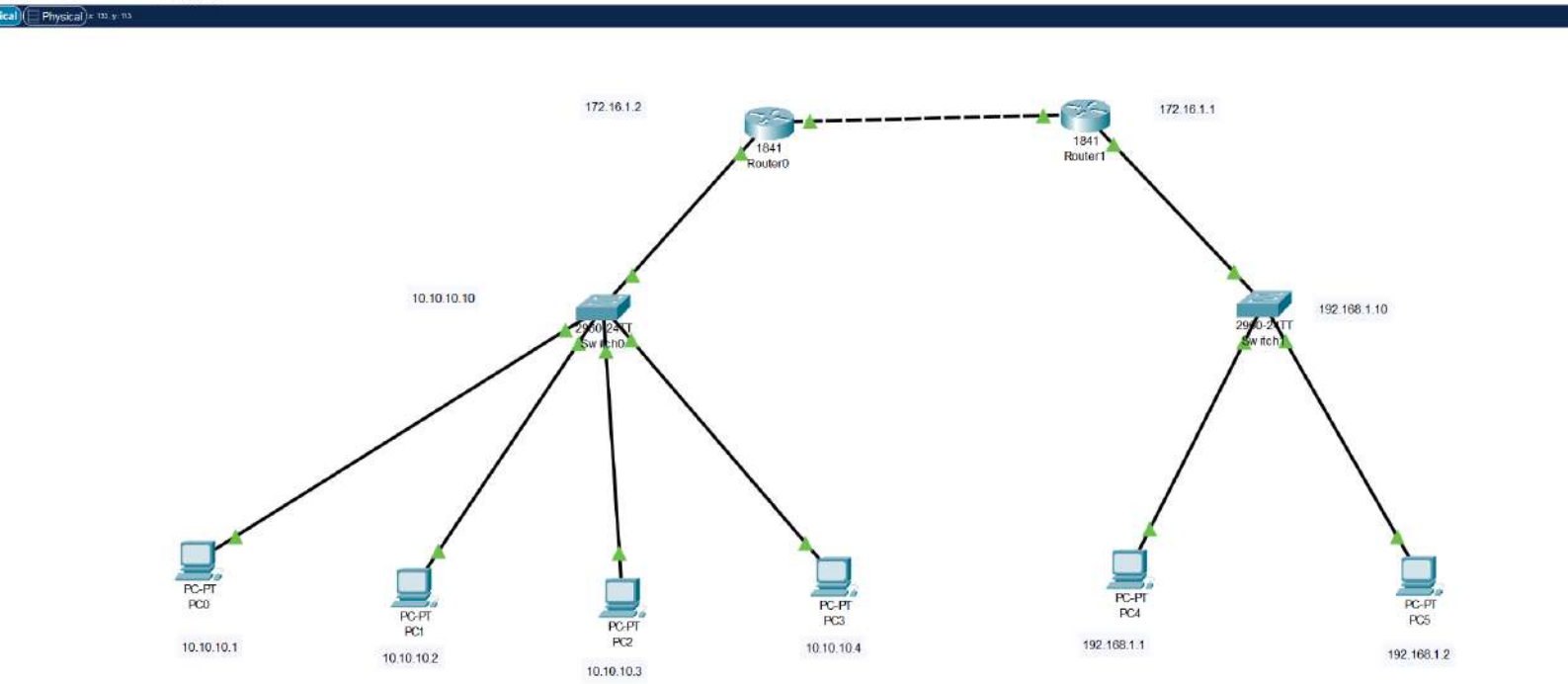
2. Configure Switch Interfaces and IP addresses :
After configuring IP, access CLI of each switch in your topology by clicking on it and selecting CLI. Then use the following commands,
vlan 10 : Creates VLAN 10
switchport access vlan 10 : Assigns the interface to the VLAN 10.
exit : Used to exit.
Repeat the above steps for all switches and VLANs as needed.

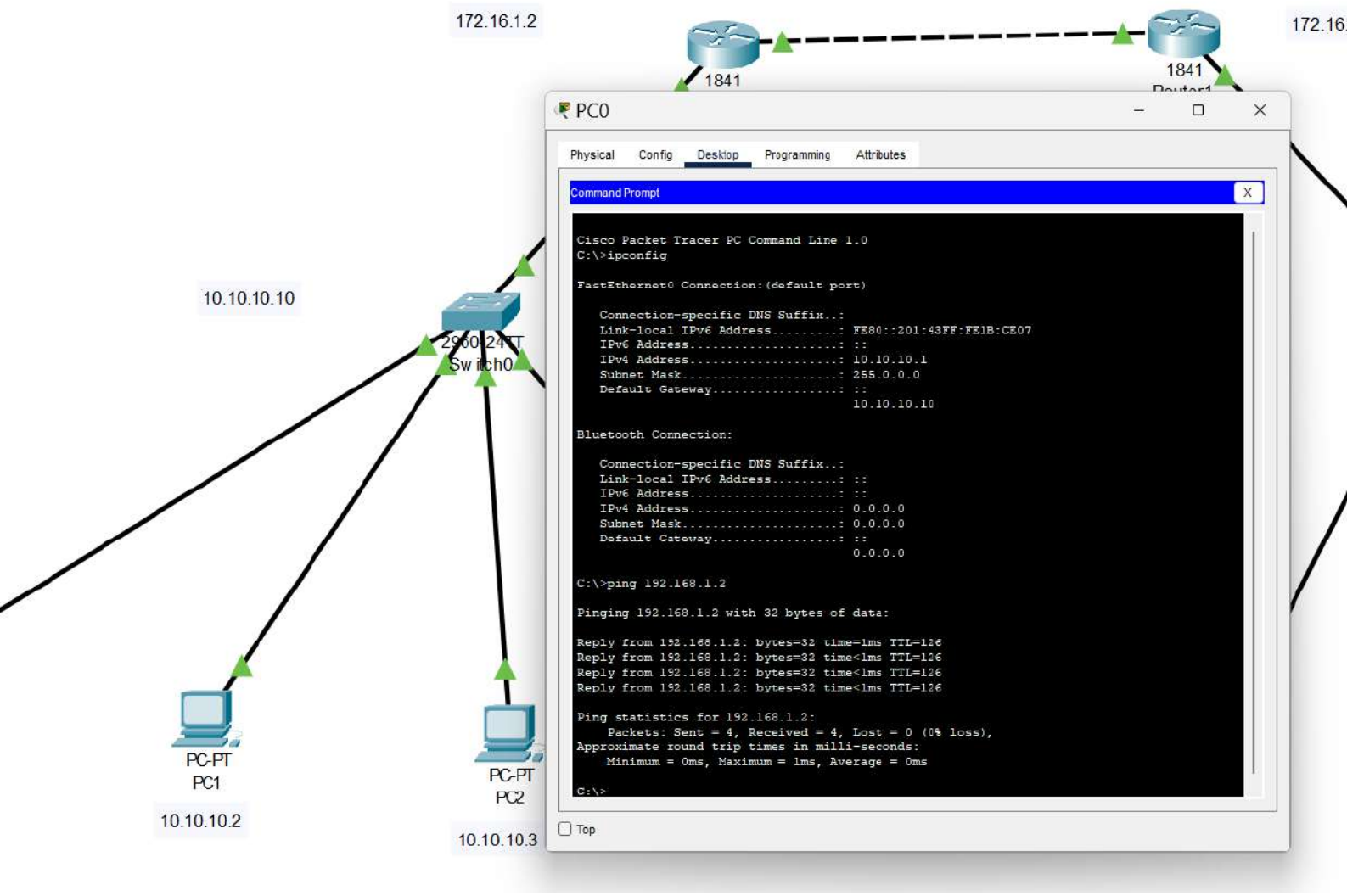
3. Test VLAN configuration :
Connect devices to the switch ports and verify that devices in the same VLAN can communicate, while devices in the different VLANs cannot communicate directly.

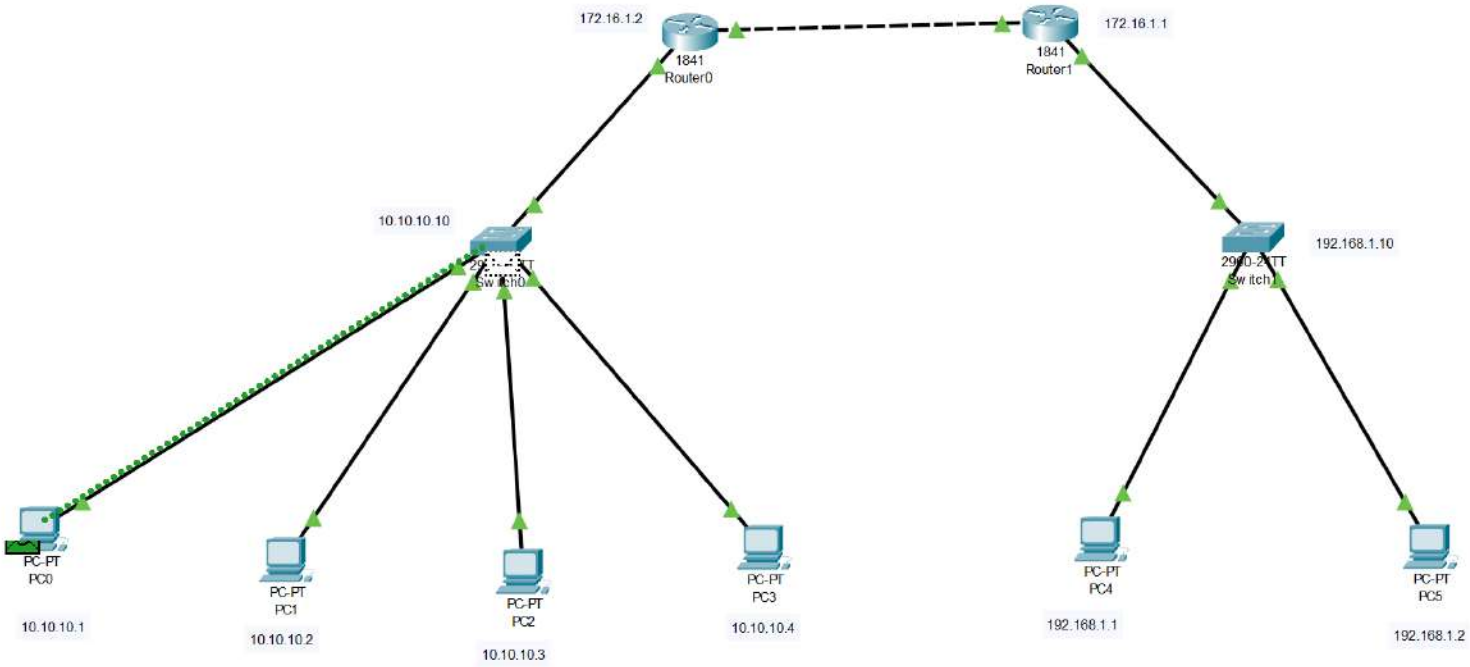
4. Save configuration (Optional) :
Save your workspace configurations to ensure they persist after a reboot.

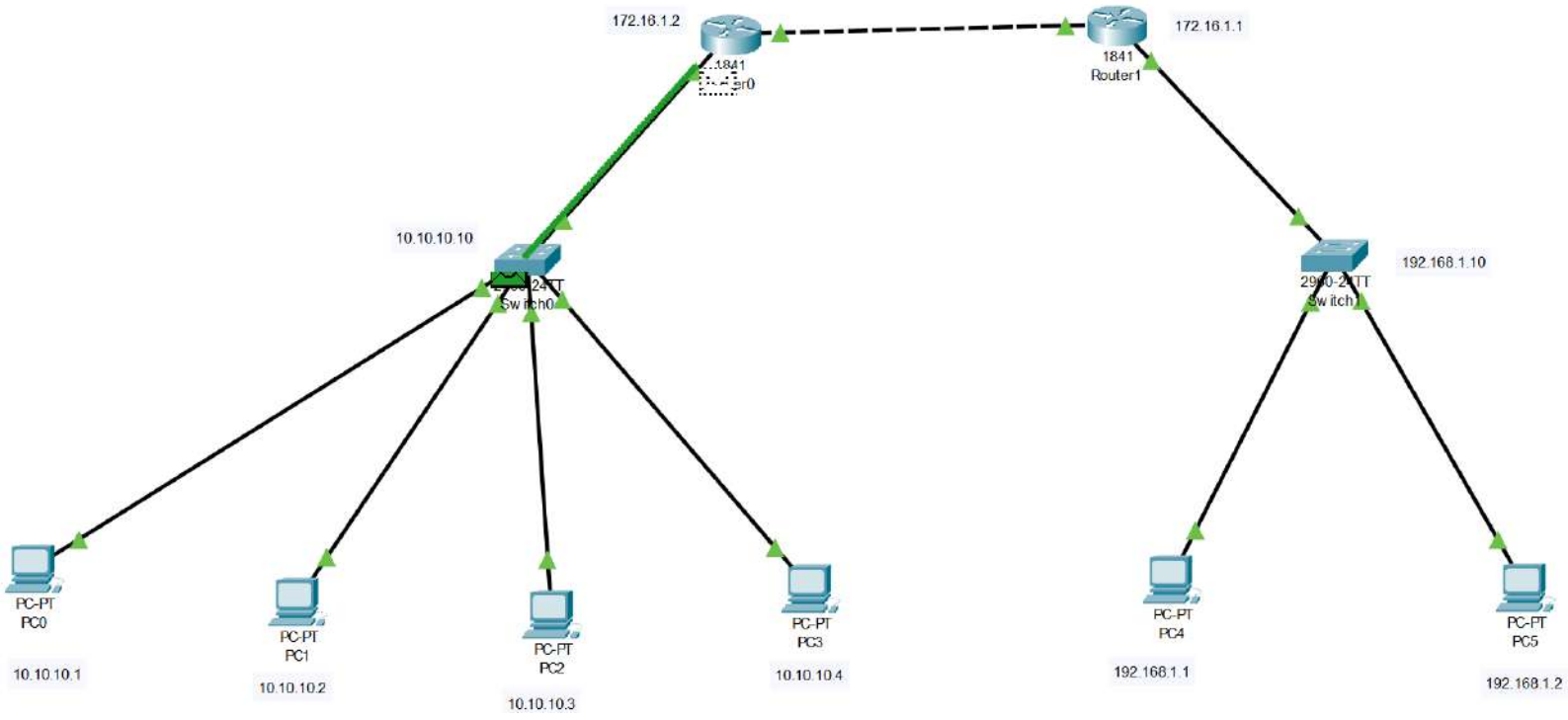
Conclusion :- Thus, we have designed and implemented RIP and VLANs using the Cisco Packet Tracer.

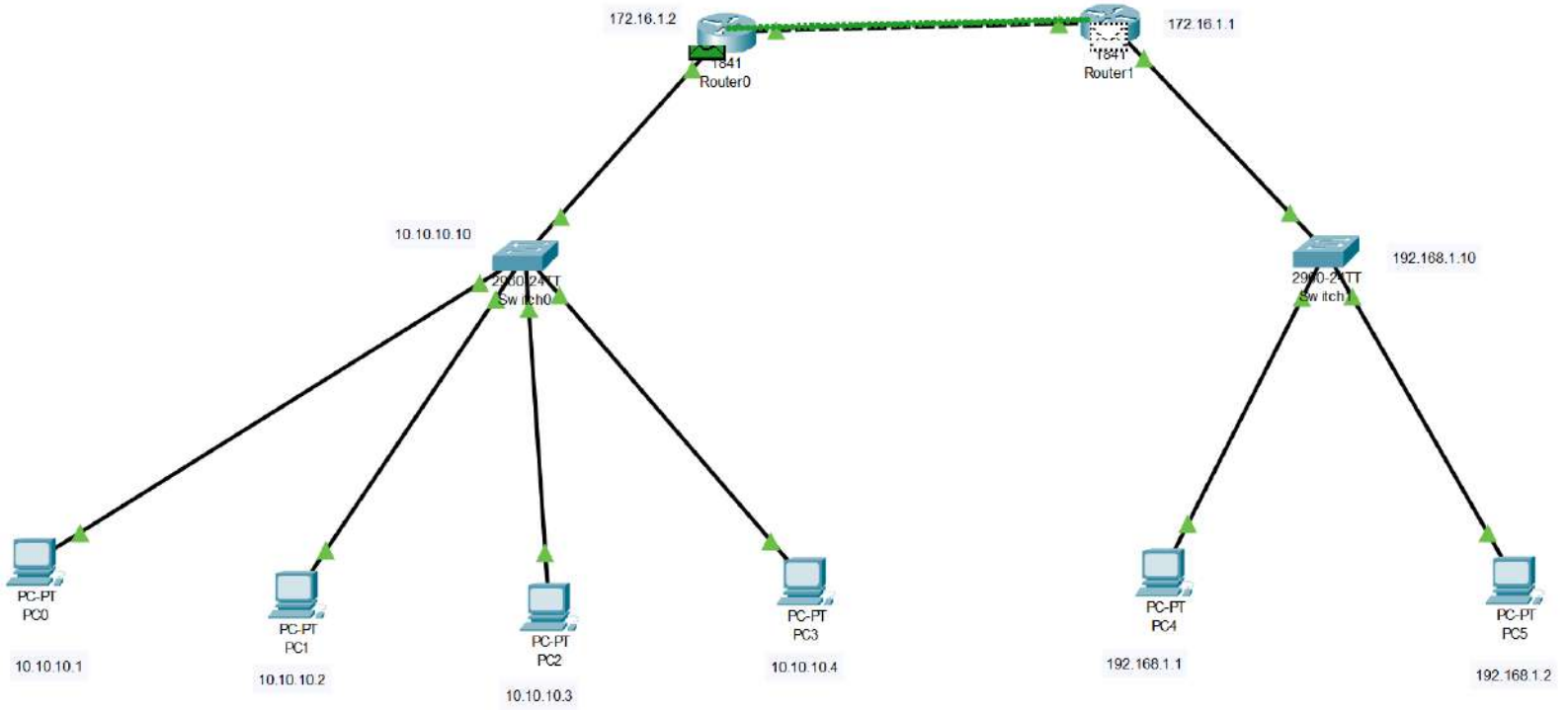
29/9/23

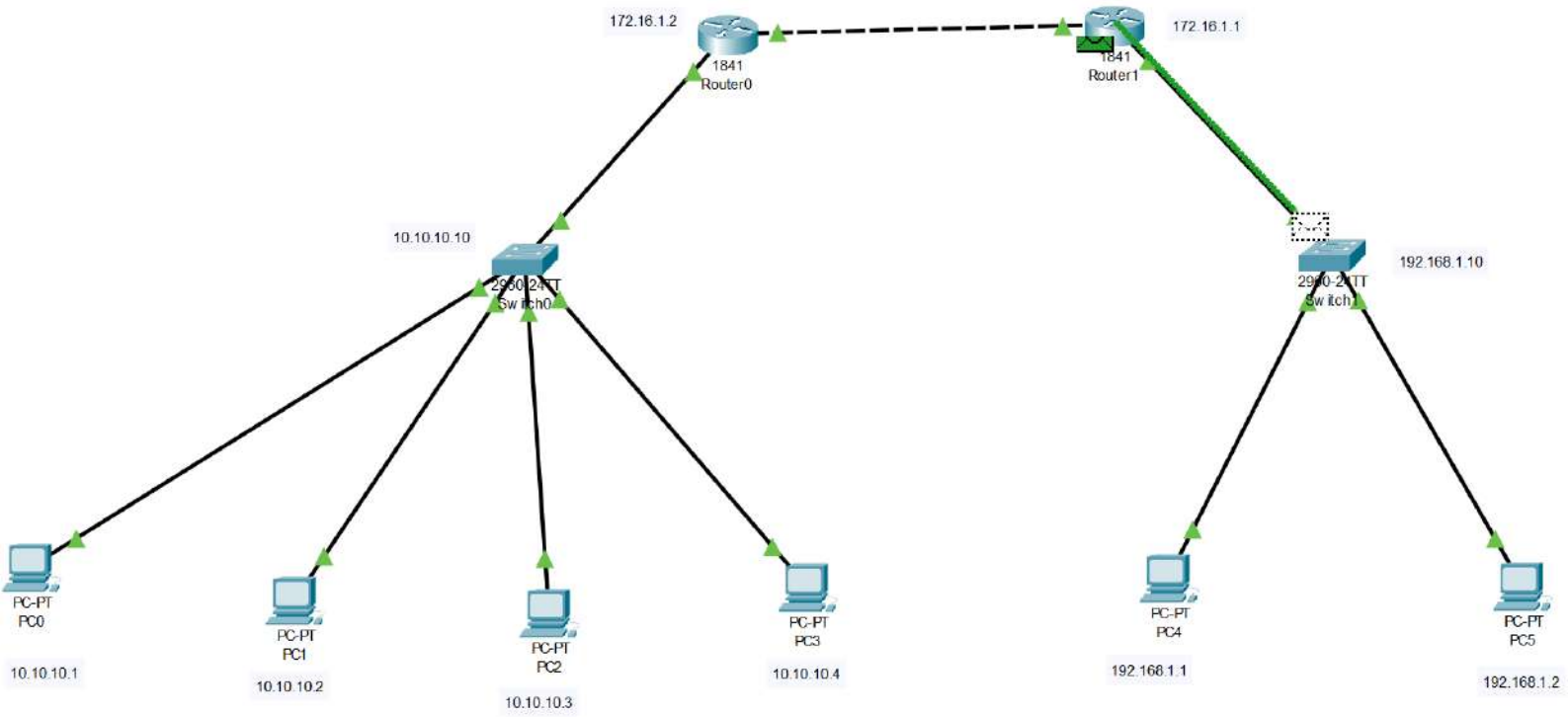

Physical
172.16.1.2
1841
Router0
172.16.1.1
1841
Router1
10.10.10.10
192.168.1.10
PC-PT
PC0
10.10.10.1
PC-PT
PC1
10.10.10.2
PC-PT
PC2
10.10.10.3
PC-PT
PC3
10.10.10.4
PC-PT
PC4
192.168.1.1
PC-PT
PC5
192.168.1.2

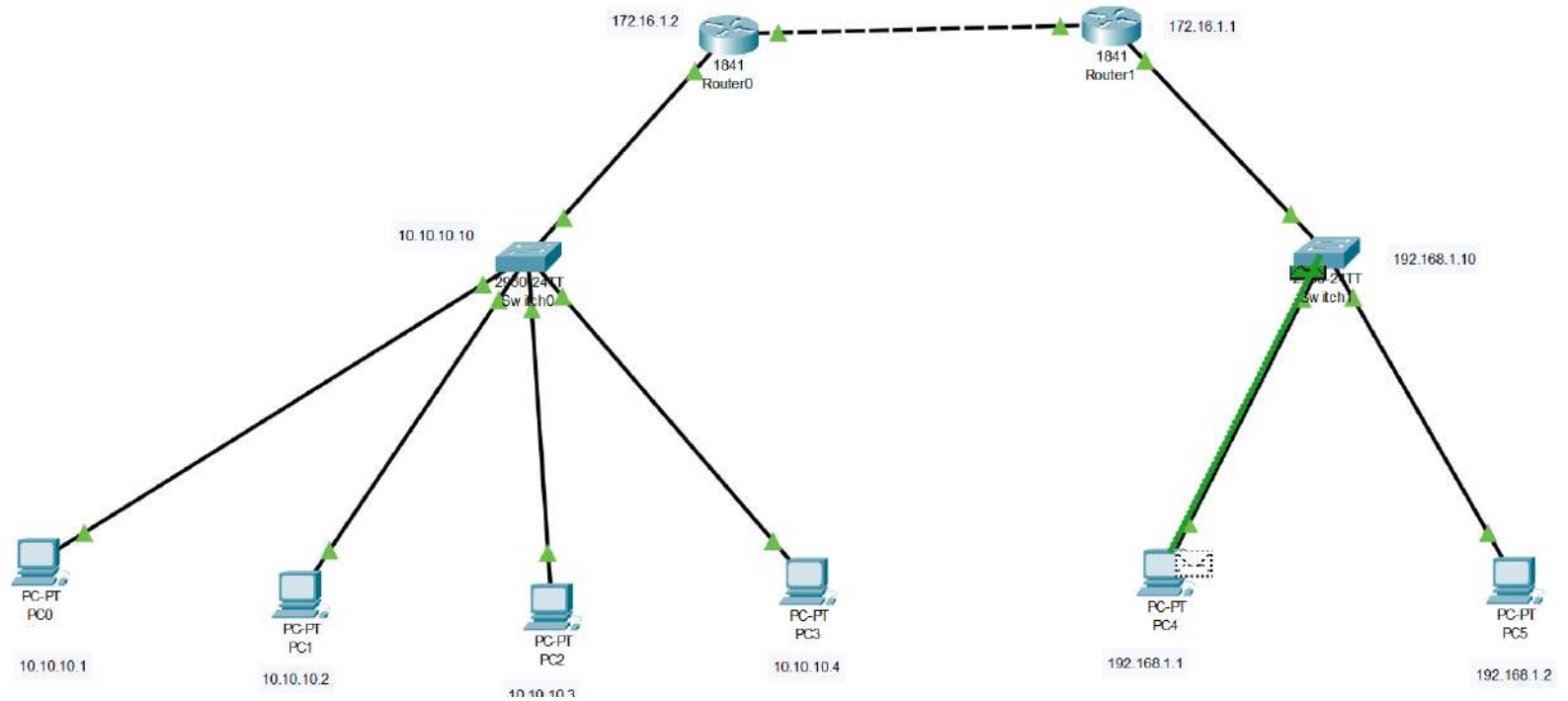

172.16.1.2
1841
1841
PC0
Physical
Config
Desktop
Programming
Attributes
Command Prompt
X
Cisco Packet Tracer PC Command Line 1.0
C:\>ipconfig
FastEthernet0 Connection:(default port)
Connection-specific DNS Suffix..:
Link-local IPv6 Address.....: FE80::201:43FF:FE1B:CE07
IPv6 Address.....: ::
IPv4 Address.....: 10.10.10.1
Subnet Mask.....: 255.0.0.0
Default Gateway.....: ::
10.10.10.10
Bluetooth Connection:
Connection-specific DNS Suffix..:
Link-local IPv6 Address.....: ::
IPv6 Address.....: ::
IPv4 Address.....: 0.0.0.0
Subnet Mask.....: 0.0.0.0
Default Gateway.....: ::
0.0.0.0
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=126
Reply from 192.168.1.2: bytes=32 time<1ms TTL=126
Reply from 192.168.1.2: bytes=32 time<1ms TTL=126
Reply from 192.168.1.2: bytes=32 time<1ms TTL=126
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 = 1ms, Average = 0ms
C:\>
Top
10.10.10.10
2960-24TT
Switch0
PC-PT
PC1
10.10.10.2
PC-PT
PC2
10.10.10.3

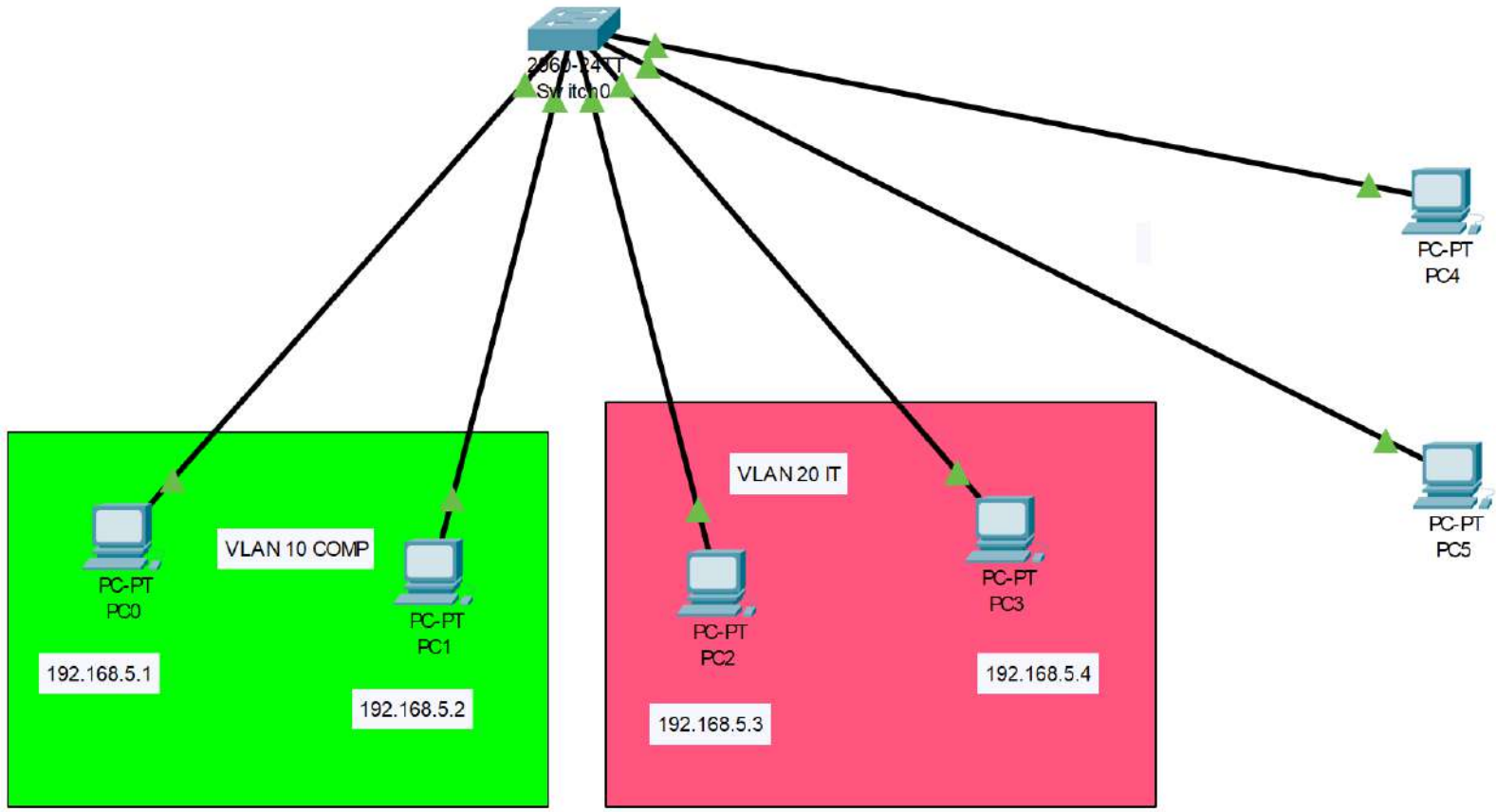

172.16.1.2
1841
Router0
172.16.1.1
1841
Router1
10.10.10.10
192.168.1.10
PC-PT
PC0
10.10.10.1
PC-PT
PC1
10.10.10.2
PC-PT
PC2
10.10.10.3
PC-PT
PC3
10.10.10.4
PC-PT
PC4
192.168.1.1
PC-PT
PC5
192.168.1.2

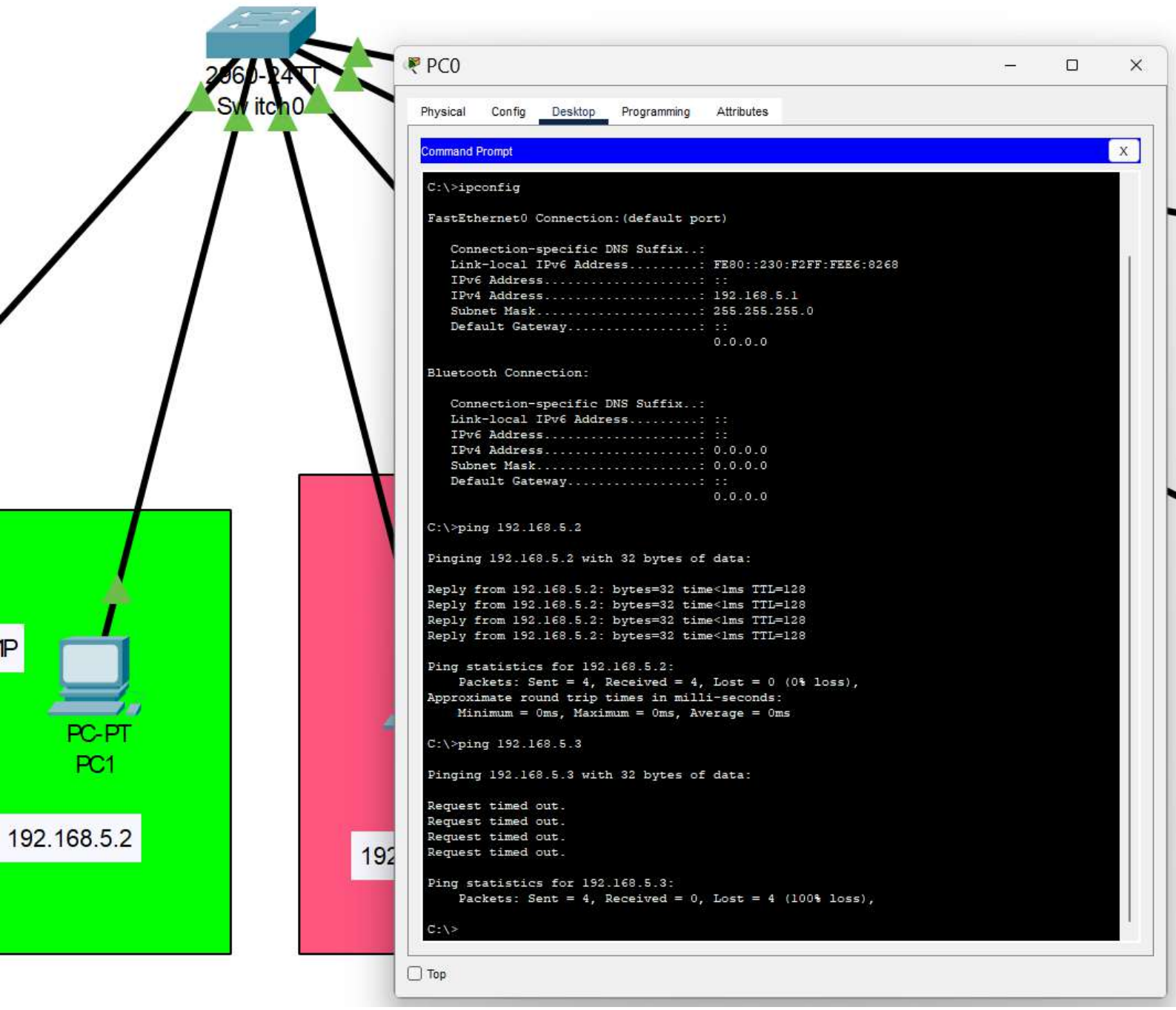

172.16.1.2
1841
Router0
172.16.1.1
1841
Router1
10.10.10.10
Switch0
192.168.1.10
Switch1
PC-PT
PC0
10.10.10.1
PC-PT
PC1
10.10.10.2
PC-PT
PC2
10.10.10.3
PC-PT
PC3
10.10.10.4
PC-PT
PC4
192.168.1.1
PC-PT
PC5
192.168.1.2

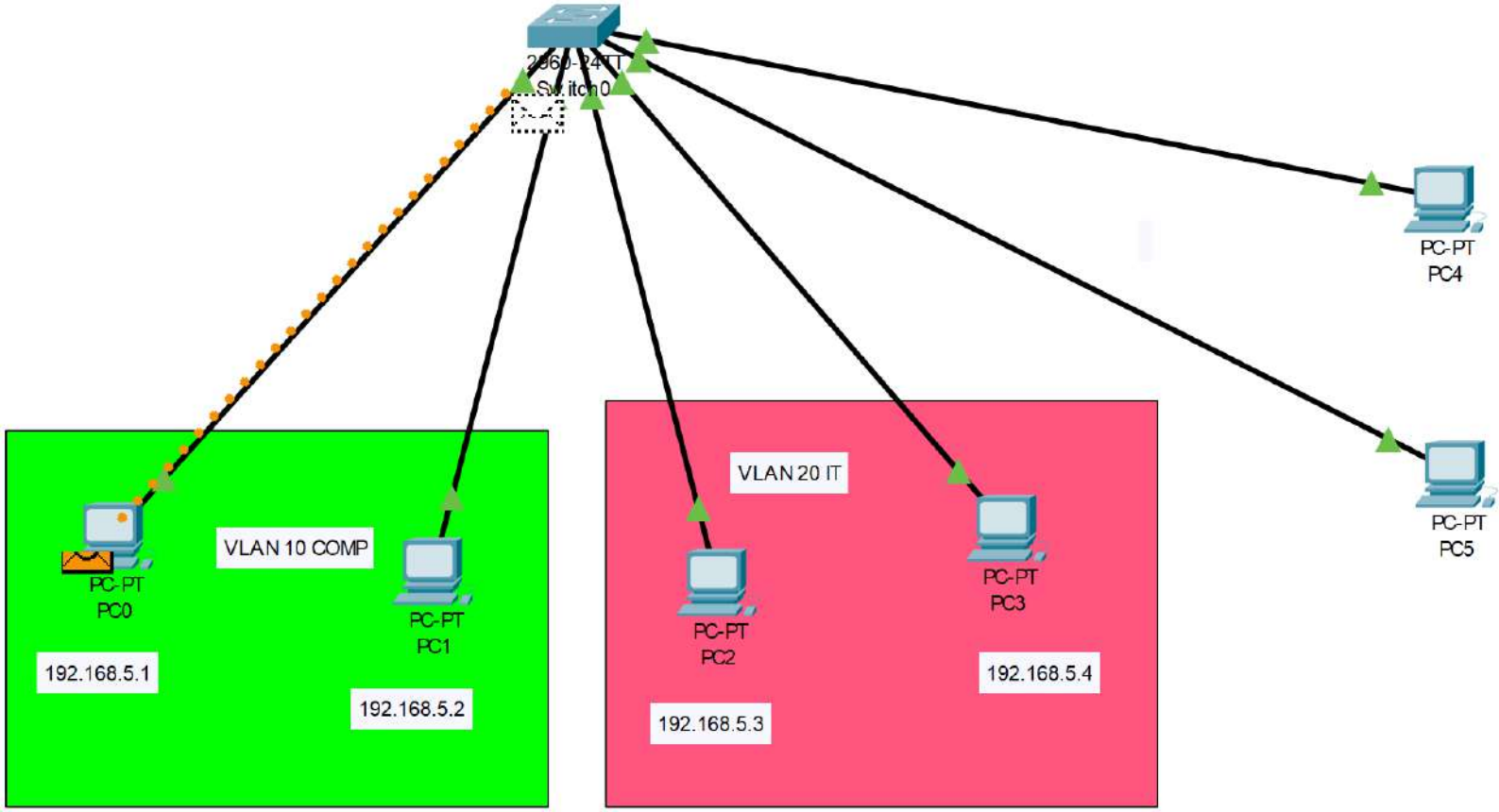

172.16.1.2
1841
Router0
172.16.1.1
1841
Router1
10.10.10.10
2960-24TT
Switch0
192.168.1.10
2960-24TT
Switch1
PC-PT
PC0
10.10.10.1
PC-PT
PC1
10.10.10.2
PC-PT
PC2
10.10.10.3
PC-PT
PC3
10.10.10.4
PC-PT
PC4
192.168.1.1
PC-PT
PC5
192.168.1.2

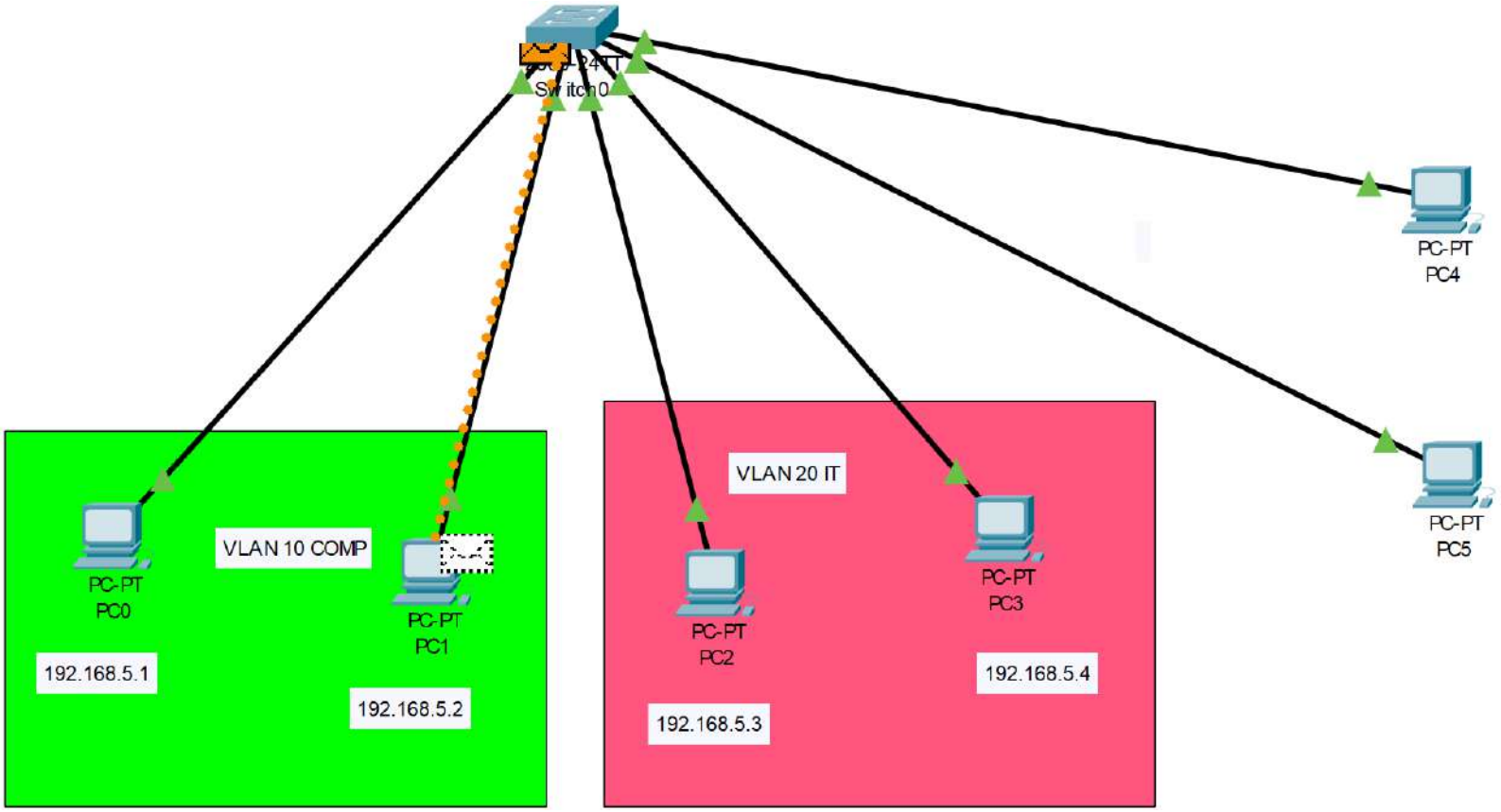

172.16.1.2
1841
Router0
172.16.1.1
1841
Router1
10.10.10.10
2950-24TT
Switch0
192.168.1.10
2950-24TT
Switch1
PC-PT
PC0
10.10.10.1
PC-PT
PC1
10.10.10.2
PC-PT
PC2
10.10.10.3
PC-PT
PC3
10.10.10.4
PC-PT
PC4
192.168.1.1
PC-PT
PC5
192.168.1.2

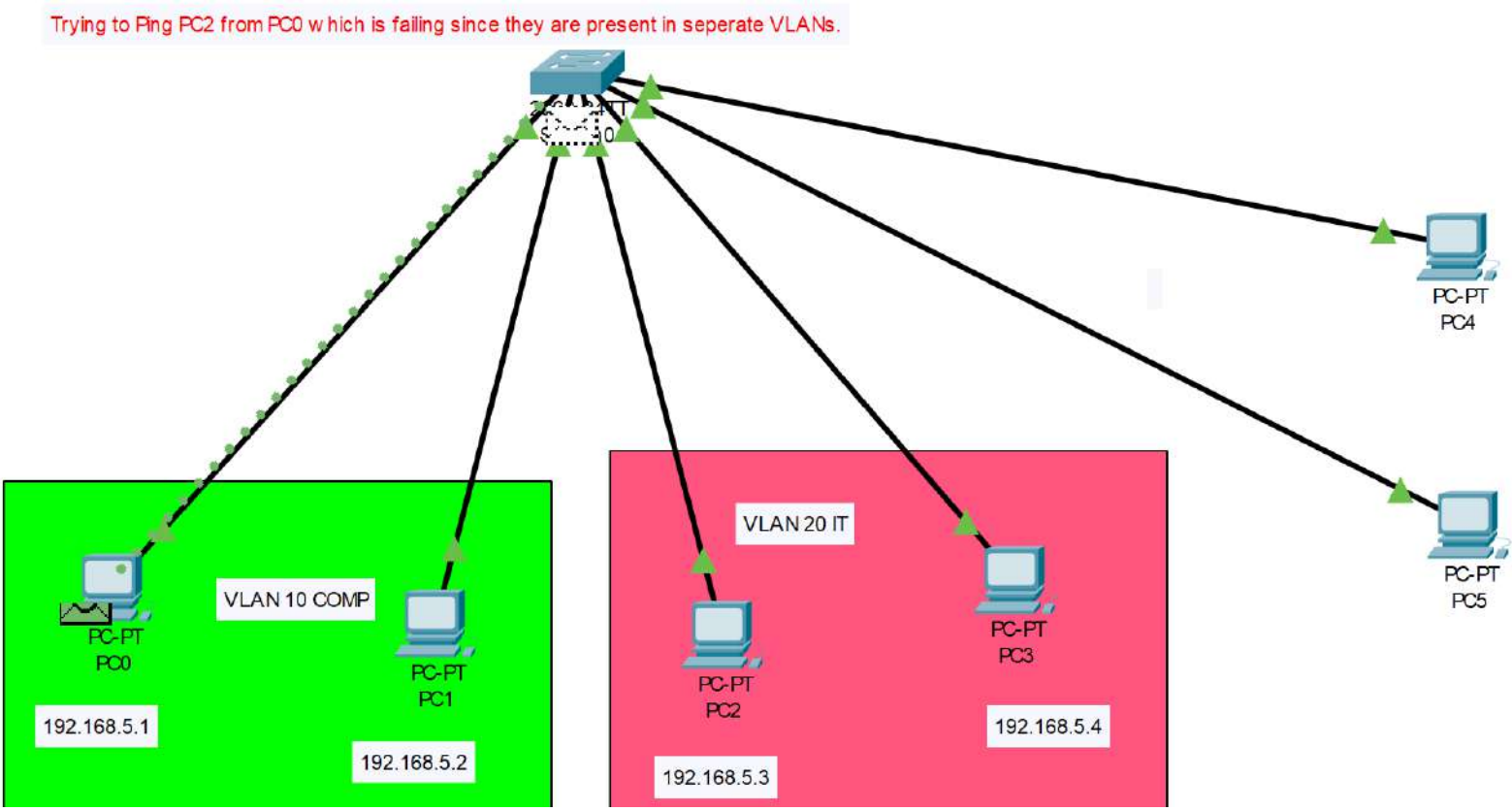

172.16.1.2
1841
Router0
172.16.1.1
1841
Router1
10.10.10.10
192.168.1.10
PC-PT
PC0
10.10.10.1
PC-PT
PC1
10.10.10.2
PC-PT
PC2
PC-PT
PC3
10.10.10.4
PC-PT
PC4
192.168.1.1
PC-PT
PC5
192.168.1.2

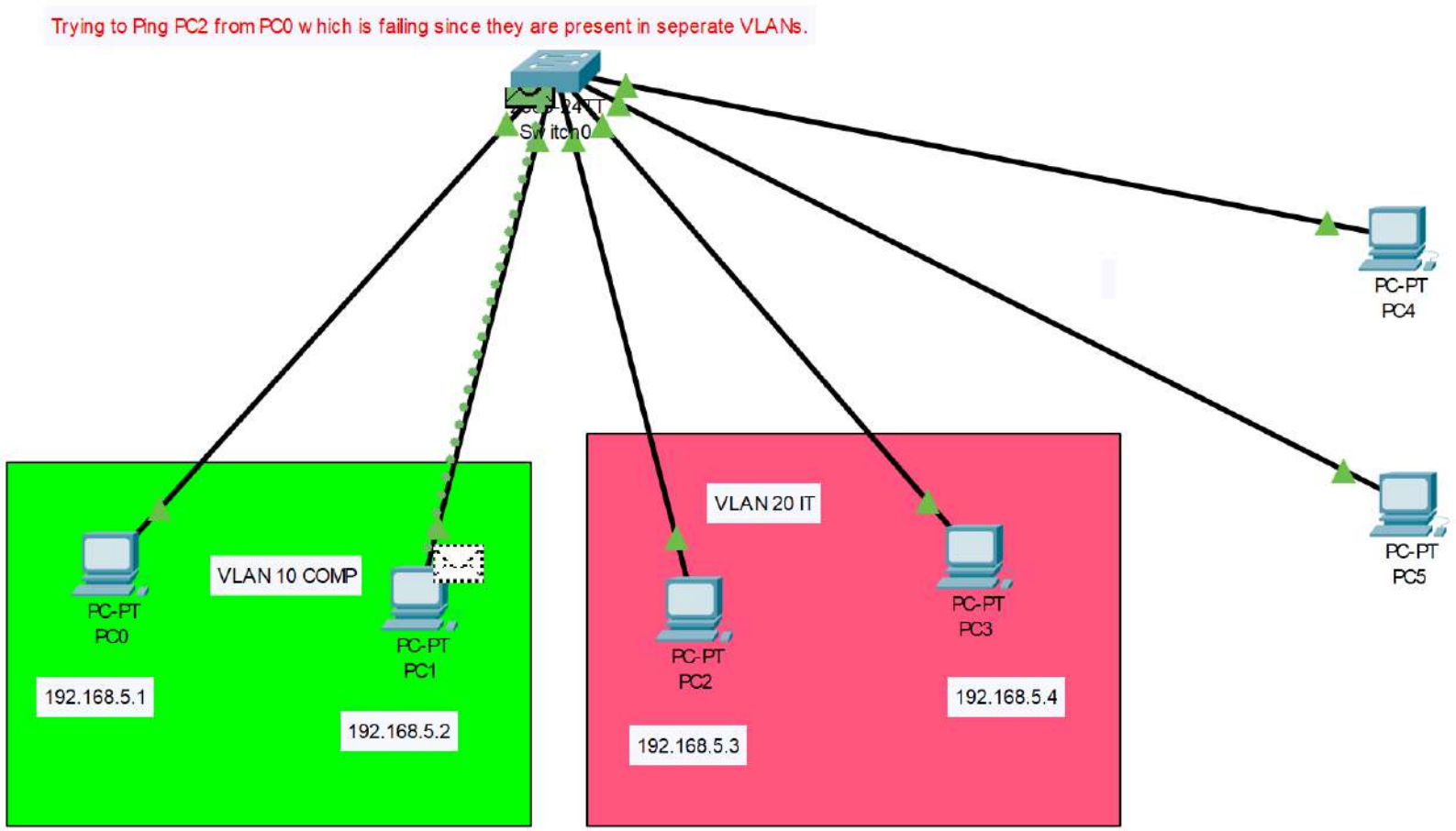

2960-24TT
Switch0
PC-PT
PC4
PC-PT
PC5
VLAN 10 COMP
PC-PT
PC0
PC-PT
PC1
192.168.5.1
192.168.5.2
VLAN 20 IT
PC-PT
PC2
PC-PT
PC3
192.168.5.3
192.168.5.4

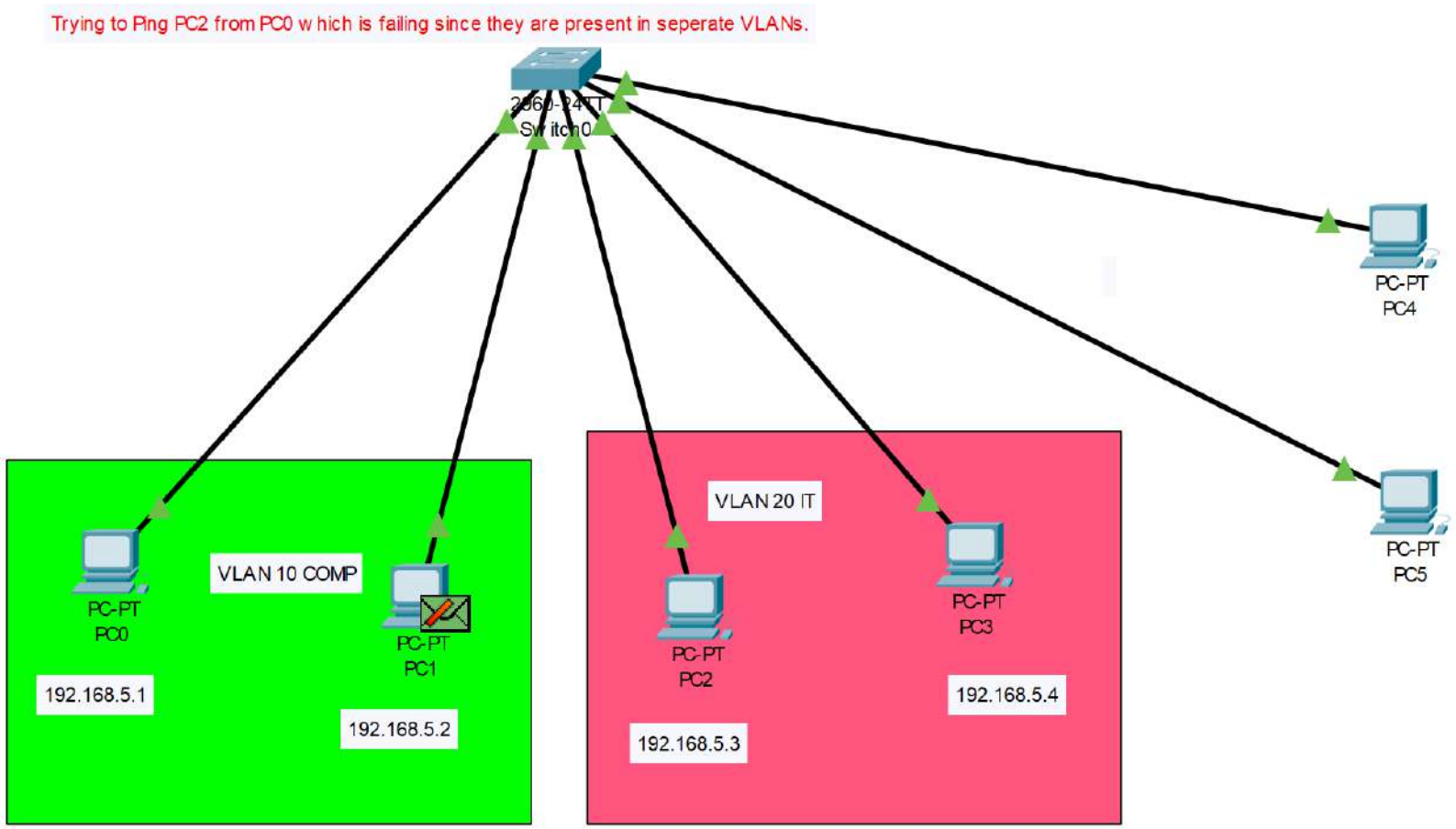

2960-24TT
Switch0
PC-PT
PC1
192.168.5.2
192
PC0
Physical
Config
Desktop
Programming
Attributes
Command Prompt
C:\>ipconfig
FastEthernet0 Connection:(default port)
Connection-specific DNS Suffix..:
Link-local IPv6 Address.....: FE80::230:F2FF:FEE6:8268
IPv6 Address.....: ::
IPv4 Address.....: 192.168.5.1
Subnet Mask.....: 255.255.255.0
Default Gateway.....: ::
0.0.0.0
Bluetooth Connection:
Connection-specific DNS Suffix..:
Link-local IPv6 Address.....: ::
IPv6 Address.....: ::
IPv4 Address.....: 0.0.0.0
Subnet Mask.....: 0.0.0.0
Default Gateway.....: ::
0.0.0.0
C:\>ping 192.168.5.2
Pinging 192.168.5.2 with 32 bytes of data:
Reply from 192.168.5.2: bytes=32 time<1ms TTL=128
Reply from 192.168.5.2: bytes=32 time<1ms TTL=128
Reply from 192.168.5.2: bytes=32 time<1ms TTL=128
Reply from 192.168.5.2: bytes=32 time<1ms TTL=128
Ping statistics for 192.168.5.2:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 0ms, Maximum = 0ms, Average = 0ms
C:\>ping 192.168.5.3
Pinging 192.168.5.3 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Ping statistics for 192.168.5.3:
Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
C:\>
Top


2960-24TT
Switch0
PC-PT
PC4
PC-PT
PC5
VLAN 10 COMP
PC-PT
PC0
PC-PT
PC1
192.168.5.1
192.168.5.2
VLAN 20 IT
PC-PT
PC2
PC-PT
PC3
192.168.5.3
192.168.5.4


2960-24TT
Switch0
PC-PT
PC4
PC-PT
PC5
VLAN 10 COMP
PC-PT
PC0
PC-PT
PC1
192.168.5.1
192.168.5.2
VLAN 20 IT
PC-PT
PC2
PC-PT
PC3
192.168.5.3
192.168.5.4


Trying to Ping PC2 from PC0 w hich is failing since they are present in seperate VLANs.
PC-PT
PC4
PC-PT
PC5
VLAN 10 COMP
PC-PT
PC0
PC-PT
PC1
192.168.5.1
192.168.5.2
VLAN 20 IT
PC-PT
PC2
PC-PT
PC3
192.168.5.3
192.168.5.4


Trying to Ping PC2 from PC0 which is failing since they are present in seperate VLANs.
Switch0
PC-PT
PC4
PC-PT
PC5
VLAN 10 COMP
PC-PT
PC0
PC-PT
PC1
192.168.5.1
192.168.5.2
VLAN 20 IT
PC-PT
PC2
PC-PT
PC3
192.168.5.3
192.168.5.4


Trying to Ping PC2 from PC0 w hich is failing since they are present in seperate VLANs.
2960-24TT
Switch0
PC-PT
PC4
PC-PT
PC5
VLAN 10 COMP
PC-PT
PC0
PC-PT
PC1
192.168.5.1
192.168.5.2
VLAN 20 IT
PC-PT
PC2
PC-PT
PC3
192.168.5.3
192.168.5.4

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO. 7

Aim :- Write a program to implement of IPv4 addressing concept along with subnet masking.

Theory :-

The implementation of IPv4 addressing, along with subnet masking, is a fundamental aspect of networking. Here's the theory on how IPv4 addressing and subnet masking work together:

1. IPv4 Addressing:

IPv4 (Internet Protocol version 4) is the fourth version of the Internet Protocol, which is used to identify and locate devices on a network.
An IPv4 address is a 32-bit numerical label assigned to each device on an IP network. It's represented as four sets of the decimal numbers, seperated by periods (eg., 192.168.1.1). IPv4 addresses are divided into two parts: the network portion and the host portion. The network portion identifies the network to which a device belongs, while the host portion identifies the specific device within that network.

2. Subnet Masking:

Subnet masking is a technique used to divide an IP address into network and host portions by using a subnet mask.
A subnet mask is a 32-bit value, like an IP address, but it consists of two parts: a string of consecutive 1s followed by a string of consecutive 0s. For example, 255.255.255.0. The subnet mask defines which bits in an IP address are for the network and which bits are for the host. In the example mask, (255.255.255.0), the first 24 bits are for the network, and the last 8 bits are for the host.
Applying the subnet mask to an IP address results in the network address, which represents the network itself.

3. Implementation:

To implement IPv4 addressing along with subnet masking, you follow these steps:

a. Choose IP Address Range:
Determine the range of IP addresses you want to use for your network. This typically involves selecting a network address and a range of host addresses.

b. Select a Subnet Mask : Choose an appropriate subnet mask based on your network's requirements The mask determines the size of subnets and number of host addresses with each subnet.

c. Calculate Subnets : Divide your chosen IP address range into subnets based on the subnet mask. Each subnet will have its own network address and a range of host addresses.

d. Assign Addresses : Assign IP addresses to devices within each subnet. Devices within each subnet will share same network address and will have unique host addresses.

Default Subnet Masks :-

Class	Range	Default Subnet Mask
A	1-126	255.0.0.0
B	128-191	255.255.0.0
C	192-223	255.255.255.0

Conclusion :- Designed a program to implement the IPv4 addressing and subnetting of networks using Java programming language.

A+
26/9/23

```
// Online IDE - Code Editor, Compiler, Interpreter
import java.util.*;

public class exp7 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter an IP Address");
        String IP = sc.next();
        // String arr[] = IP.split(".");
        if (!IP.contains(".")) {
            System.out.println("IP address " + IP + " is invalid");
            sc.close();
            return;
        }
        int IPClass;
        if (IP.charAt(1) == '.')
            IPClass = Integer.parseInt(IP.substring(0, 1));
        else if (IP.charAt(2) == '.')
            IPClass = Integer.parseInt(IP.substring(0, 2));
        else
            IPClass = Integer.parseInt(IP.substring(0, 3));
        if (IPClass >= 1 && IPClass <= 126)
            System.out.println("The IP address " + IP + " belongs to
class A\nNet ID: " + IP
                    + "\nTotal no. of IP addresses possible:
256*256*256\nNetwork mask: 255.0.0.0");
        else if (IPClass >= 128 && IPClass <= 191)
            System.out.println("The IP address " + IP + " belongs to
class B\nNet ID: " + IP
                    + "\nTotal no. of IP addresses possible:
256*256\nNetwork mask: 255.255.0.0");
        else if (IPClass >= 192 && IPClass <= 223)
            System.out.println("The IP address " + IP + " belongs to
class C\nNet ID: " + IP
                    + "\nTotal no. of IP addresses possible:
256\nNetwork mask: 255.255.255.0");
        else if (IPClass >= 224 && IPClass <= 239)
            System.out.println("The IP address " + IP + " belongs to
class D\nNet ID: " + IP
                    + "\nTotal no. of IP addresses possible:
256\nNetwork mask: 255.255.255.0");
        else if (IPClass >= 240 && IPClass <= 255)
            System.out.println("The IP address " + IP + " belongs to
class E\nNet ID: " + IP
                    + "\nTotal no. of IP addresses possible:
256\nNetwork mask: 255.255.255.0");
        else {
            System.out.println("IP address " + IP + " is invalid");
            sc.close();
            return;
        }
        System.out.println("Now enter the number of subnets(power of
2)");
        int subnets = sc.nextInt();
        if ((subnets & 1) == 1)
```

```
            System.out.println("Number of subnets is not in the
power of 2");
        String binary = Integer.toBinaryString(subnets);
        System.out.println("Number of subnets: " + subnets);
        System.out.println("Number of bits in subnets ID: " +
(binary.length() - 1));
        int noOfSubnetAddress = ((int) Math.pow(2, 8 -
(binary.length() - 1)));
        System.out.println(
                "Total no of IP addresses possible in each subnet: "
+ ((int) Math.pow(2, 8 - (binary.length() - 1))));
        int temp = -1;
        for (int i = 0; i < subnets; i++) {
            System.out.println("\nSubnet " + i + ": -");
            System.out.println("Subnet address - " + IP.substring(0,
12) + (temp + 1));
            temp += noOfSubnetAddress;
            System.out.println("Broadcast address - " +
IP.substring(0, 12) + temp);
            System.out.println(
                    "Valid range of host IP address - " +
IP.substring(0, 12) + (temp - noOfSubnetAddress + 2) + " - "
                            + IP.substring(0, 13) + (temp - 1));
        }
        sc.close();

    }
}
```

C:\Users\Rishab\OneDrive\Desktop\CN Experiments>java exp7

Enter an IP Address

192.168.10.00

The IP address 192.168.10.00 belongs to class C

Net ID: 192.168.10.00

Total no. of IP addresses possible: 256

Network mask: 255.255.255.0

Now enter the number of subnets(power of 2)

4

Number of subnets: 4

Number of bits in subnets ID: 2

Total no of IP addresses possible in each subnet: 64

Subnet 0: -

Subnet address - 192.168.10.0

Broadcast address - 192.168.10.63

Valid range of host IP address - 192.168.10.1 - 192.168.10.62

Subnet 1: -

Subnet address - 192.168.10.64

Broadcast address - 192.168.10.0127

Valid range of host IP address - 192.168.10.65 - 192.168.10.00126

Subnet 2: -

Subnet address - 192.168.10.128

Broadcast address - 192.168.10.191

Valid range of host IP address - 192.168.10.129 - 192.168.10.190

Subnet 3: -

Subnet address - 192.168.10.192

Broadcast address - 192.168.10.255

Valid range of host IP address - 192.168.10.193 - 192.168.10.254

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO. 8

Aim :- Use basic networking commands in Linux (ping, tracert, nslookup, netstat, ARP, RARP, ip, ipconfig, dig, route)

Theory :-

Following are some of the basic networking commands in the Linux :-

1. Ping :
ping is a command used to test the network connectivity between two devices by sending ICMP echo request packets and receiving ICMP echo reply packets.
It's commonly used to check if a remote host or IP address is reachable.

2. Traceroute (Tracert in Windows) :
traceroute is a command used to trace the route that packets take from your local machine to a destination host or IP address. It shows the IP addresses of immediate routers and the time it takes for packets to reach each hop.

3. nslookup (or dig) :
nslookup is used to query DNS (Domain Name System) servers to look up domain names and retrieve associated IP addresses. dig (Domain

Information (Groper) is another tool for DNS queries and provides more detailed information.

4. netstat :
netstat is a command used to display network statistics and active network connections. It shows information about listening ports, active connections, routing tables, and more.

5. ARP (Address Resolution Protocol) and RARP (Reverse ARP) :
arp is used to view and manipulate the ARP cache, which maps IP addresses to MAC (Media Access Control) addresses on local network. rarp (Reverse ARP) is used to map MAC addresses to IP addresses, but it's less commonly used.

6. ip :
The ip command is a versatile tool for configuring and managing network interfaces, routes, and tunnels. It can be used to assign the IP addresses, set up virtual interfaces, and manage routing tables.

7. ifconfig :
ifconfig (Interface Configuration) is used to view and configure network interfaces, including enabling or disabling interfaces, setting IP addresses, and managing network parameters.

8. route :
The route command is used to view and also manipulate kernel's IP routing table.
It's used to add or delete routes, change default gateways, and configure routing settings.

9. Usage Examples :
To ping : ping example.com
To trace route to server : traceroute example.com
To query DNS for IP address : nslookup example.com
or dig example.com
To display network statistics : netstat -i
To view ARP cache : arp -a
To configure network interfaces : ip addr add 192.168.1.2/24
To add a route : route add -net 192.168.2.0/24 gw 192.168.1.1

These basic networking commands are essential for diagnosing network issues, configuring network settings, and troubleshooting network connectivity in a Linux environment.

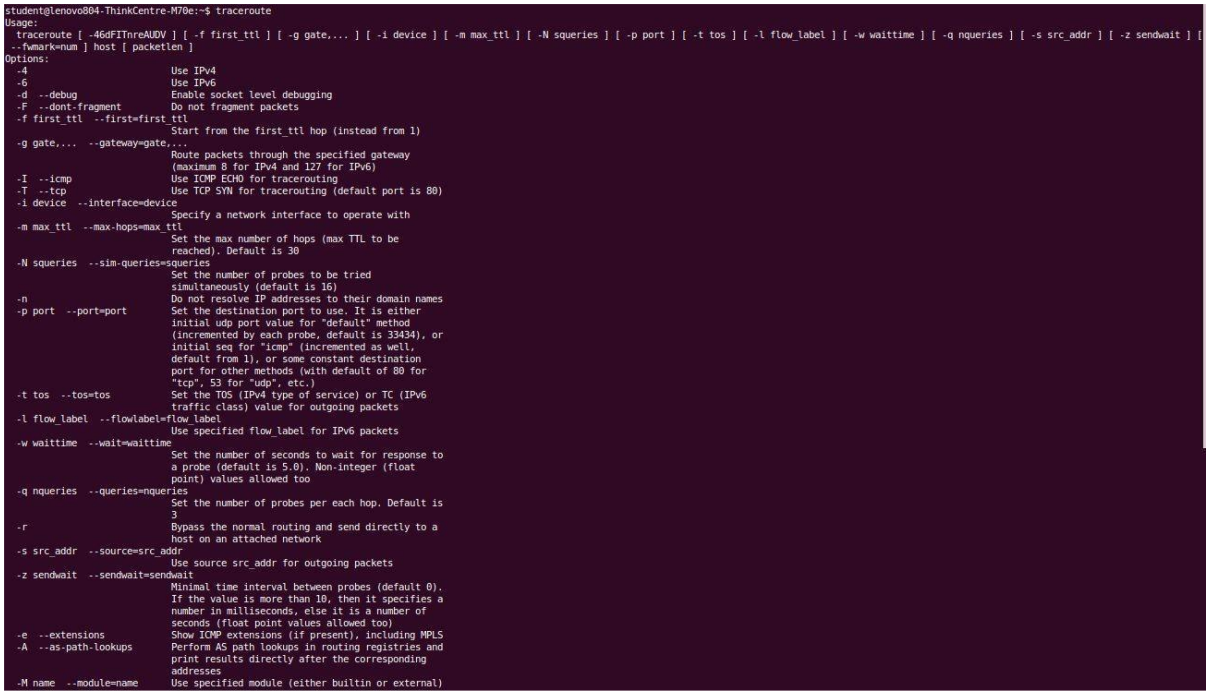
Conclusion :- Thus, we implemented the basic networking commands in Linux as they provide valuable insights into network status and help in maintaining functional network.

31/10/23

```
student@lenovo804-ThinkCentre-M70e: ~
student@lenovo804-ThinkCentre-M70e:~$ ifconfig
docker0   Link encap:Ethernet  HWaddr 02:42:cf:c7:15:71
          inet addr:172.17.0.1  Bcast:0.0.0.0  Mask:255.255.0.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0      Link encap:Ethernet  HWaddr 44:37:e6:4d:df:1b
          inet addr:10.1.8.4  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::4637:e6ff:fe4d:df1b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:51944 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18626 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:27621649 (27.6 MB)  TX bytes:2682227 (2.6 MB)
          Interrupt:17

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:2173 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2173 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:193433 (193.4 KB)  TX bytes:193433 (193.4 KB)

student@lenovo804-ThinkCentre-M70e:~$
```

```
student@lenovo804-ThinkCentre-M70e: ~
student@lenovo804-ThinkCentre-M70e:~$ nslookup www.atharvacoe.ac.in
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
www.atharvacoe.ac.in    canonical name = atharvacoe.ac.in.
Name:   atharvacoe.ac.in
Address: 192.185.180.65

student@lenovo804-ThinkCentre-M70e:~$
```

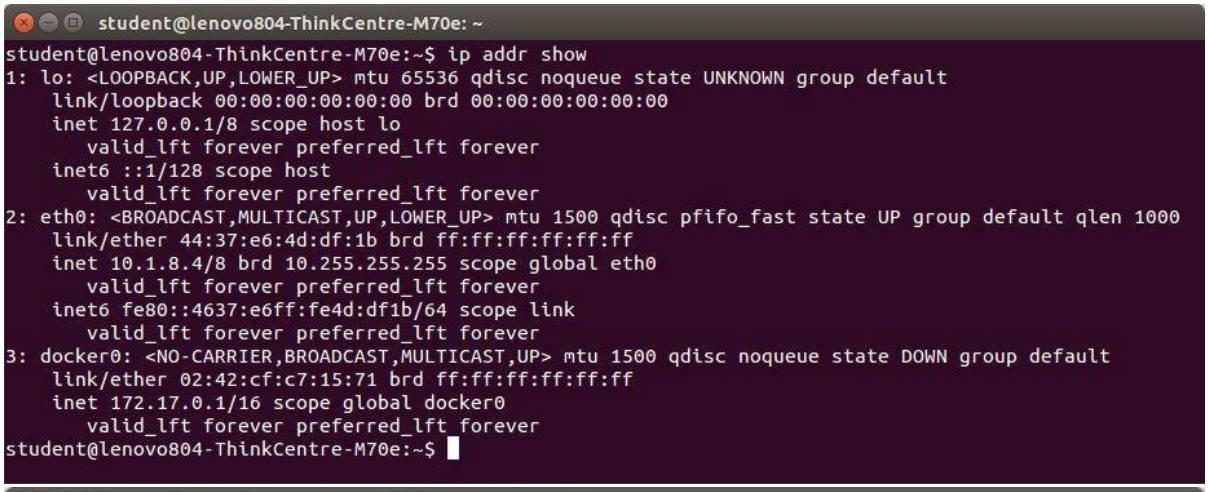
```
student@lenovo804-ThinkCentre-M70e: ~
student@lenovo804-ThinkCentre-M70e:~$ ping -c 4 10.1.8.3
PING 10.1.8.3 (10.1.8.3) 56(84) bytes of data.
64 bytes from 10.1.8.3: icmp_seq=1 ttl=64 time=0.324 ms
64 bytes from 10.1.8.3: icmp_seq=2 ttl=64 time=0.333 ms
64 bytes from 10.1.8.3: icmp_seq=3 ttl=64 time=0.316 ms
64 bytes from 10.1.8.3: icmp_seq=4 ttl=64 time=0.302 ms

--- 10.1.8.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.302/0.318/0.333/0.024 ms
student@lenovo804-ThinkCentre-M70e:~$
```

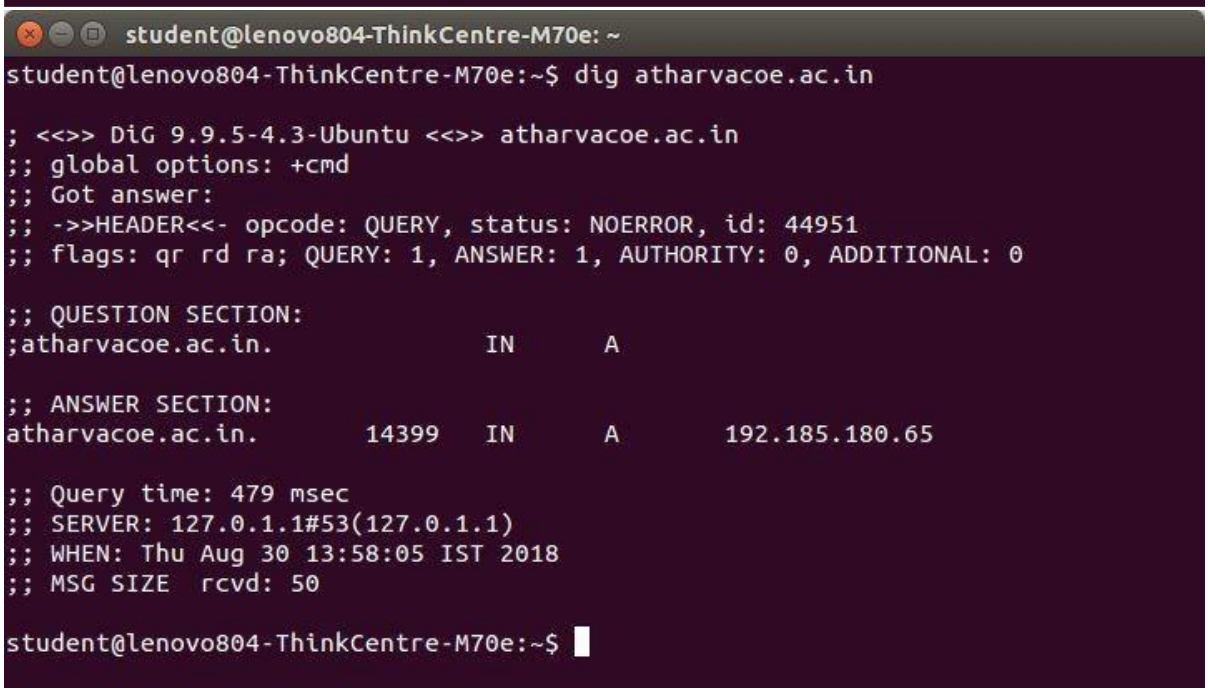


```
student@lenovo804-ThinkCentre-M70e:~$ traceroute
Usage:
  traceroute [ -46dFITnreAUDV ] [ -f first_ttl ] [ -g gate,... ] [ -i device ] [ -m max_ttl ] [ -N squeries ] [ -p port ] [ -t tos ] [ -l flow_label ] [ -w waittime ] [ -q nqueries ] [ -s src_addr ] [ -z sendwait ] [
  --fwmark=num ] host [ packetlen ]
Options:
  -4                      Use IPv4
  -6                      Use IPv6
  -d  --debug             Enable socket level debugging
  -F  --dont-fragment     Do not fragment packets
  -f first_ttl  --first=first_ttl
                          Start from the first_ttl hop (instead from 1)
  -g gate,...  --gateway=gate,...
                          Route packets through the specified gateway
                          (maximum 8 for IPv4 and 127 for IPv6)
  -I  --icmp              Use ICMP ECHO for tracerouting
  -T  --tcp               Use TCP SYN for tracerouting (default port is 80)
  -i device  --interface=device
                          Specify a network interface to operate with
  -m max_ttl  --max-hops=max_ttl
                          Set the max number of hops (max TTL to be
                          reached). Default is 30
  -N squeries  --sim-queries=squeries
                          Set the number of probes to be tried
                          simultaneously (default is 16)
  -n                      Do not resolve IP addresses to their domain names
  -p port  --port=port    Set the destination port to use. It is either
                          initial udp port value for "default" method
                          (incremented by each probe, default is 33434), or
                          initial seq for "icmp" (incremented as well,
                          default from 1), or some constant destination
                          port for other methods (with default of 80 for
                          "tcp", 53 for "udp", etc.)
  -t tos  --tos=tos       Set the TOS (IPv4 type of service) or TC (IPv6
                          traffic class) value for outgoing packets
  -l flow_label  --flowlabel=flow_label
                          Use specified flow_label for IPv6 packets
  -w waittime  --wait=waittime
                          Set the number of seconds to wait for response to
                          a probe (default is 5.0). Non-integer (float
                          point) values allowed too
  -q nqueries  --queries=nqueries
                          Set the number of probes per each hop. Default is
                          3
  -r                      Bypass the normal routing and send directly to a
                          host on an attached network
  -s src_addr  --source=src_addr
                          Use source src_addr for outgoing packets
  -z sendwait  --sendwait=sendwait
                          Minimal time interval between probes (default 0).
                          If the value is more than 10, then it specifies a
                          number in milliseconds, else it is a number of
                          seconds (float point values allowed too)
  -e  --extensions        Show ICMP extensions (if present), including MPLS
  -A  --as-path-lookups   Perform AS path lookups in routing registries and
                          print results directly after the corresponding
                          addresses
  -M name  --module=name  Use specified module (either builtin or external)
```

```
student@lenovo804-ThinkCentre-M70e: ~
student@lenovo804-ThinkCentre-M70e:~$ netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 lenovo804-ThinkC:domain *:*                     LISTEN
tcp        0      0 localhost:ipp           *:*                     LISTEN
tcp        0      0 10.1.8.4:40190          bom05s11-in-f2.1e:https TIME_WAIT
tcp        0      0 10.1.8.4:52797          151.101.2.114:https     TIME_WAIT
tcp        0      0 10.1.8.4:38575          bom05s15-in-f14.1:https ESTABLISHED
tcp        0      0 10.1.8.4:38576          bom05s15-in-f14.1:https ESTABLISHED
tcp        0      0 10.1.8.4:52065          bom05s15-in-f4.1e:https TIME_WAIT
tcp        0      0 10.1.8.4:52796          151.101.2.114:https     TIME_WAIT
tcp        0      0 10.1.8.4:40191          bom05s11-in-f2.1e:https TIME_WAIT
tcp        0      0 10.1.8.4:38634          bom05s15-in-f14.1:https ESTABLISHED
tcp        0      0 10.1.8.4:38637          bom05s15-in-f14.1:https TIME_WAIT
tcp        0      0 10.1.8.4:38573          bom05s15-in-f14.1:https ESTABLISHED
tcp        0      0 10.1.8.4:37409          server-52-222-135:https TIME_WAIT
tcp        0      0 10.1.8.4:41299          a184-30-54-102.de:https TIME_WAIT
```

```
student@lenovo804-ThinkCentre-M70e: ~
student@lenovo804-ThinkCentre-M70e:~$ arp -v
Address                  HWtype  HWaddress           Flags Mask            Iface
10.8.1.3                         (incomplete)                              eth0
10.0.0.3                 ether   08:35:71:f0:35:c0   C                     eth0
10.1.8.3                 ether   44:37:e6:4d:e0:f7   C                     eth0
Entries: 3      Skipped: 0      Found: 3
student@lenovo804-ThinkCentre-M70e:~$
```


student@lenovo804-ThinkCentre-M70e: ~
student@lenovo804-ThinkCentre-M70e:~$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
link/ether 44:37:e6:4d:df:1b brd ff:ff:ff:ff:ff:ff
inet 10.1.8.4/8 brd 10.255.255.255 scope global eth0
valid_lft forever preferred_lft forever
inet6 fe80::4637:e6ff:fe4d:df1b/64 scope link
valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
link/ether 02:42:cf:c7:15:71 brd ff:ff:ff:ff:ff:ff
inet 172.17.0.1/16 scope global docker0
valid_lft forever preferred_lft forever
student@lenovo804-ThinkCentre-M70e:~$


student@lenovo804-ThinkCentre-M70e: ~
student@lenovo804-ThinkCentre-M70e:~$ dig atharvacoe.ac.in
; <<>> DiG 9.9.5-4.3-Ubuntu <<>> atharvacoe.ac.in
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44951
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;atharvacoe.ac.in. IN A
;; ANSWER SECTION:
atharvacoe.ac.in. 14399 IN A 192.185.180.65
;; Query time: 479 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Thu Aug 30 13:58:05 IST 2018
;; MSG SIZE rcvd: 50
student@lenovo804-ThinkCentre-M70e:~$

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

EXPERIMENT NO. 9

Aim :- Use Wire shark to understand operation of TCP/IP layers :-
Ethernet Layer : Frame header, Frame size etc.
DLL : MAC address, ARP
Network Layer : IP Packet, ICMP
Transport Layer : TCP Ports, TCP handshake

Theory :-

Using Wireshark, you can gain valuable insights into the operation of TCP/IP layers and examine the various aspects of the network communication. Here's a theoretical overview of how Wireshark can help you understand each of these layers :

1. Ethernet Layer :

Wireshark captures Ethernet frames, providing details such as frame header, frame size, source MAC address, and destination one.
You can inspect Ethernet frame headers to understand the physical layer characteristics of data being transmitted on network.

2. Data Link Layer :

Wireshark allows you to view MAC (Media Access Control) addresses within Ethernet frames.

Ethernet Header Format:

Preamble	Start Frame Delimiter	Destination Address	Soure Address	Length	Data	Frame Check Sequence (CRC)
7 byte	1 byte	6 byte	6 byte	2 byte	46 to 1500B	4 byte

TCP Header Format:

Source Port Address 16-bit								Destination Port Address 16-bit
Sequence Number 32-bit								
Acknowledgement Number 32-bit								
HLEN 4-bit	Reserved 6-bit	U R G	A C K	P S H	R S T	S Y N	F I N	Window Size 16-bit
Checksum (16-bit)								Urgent Pointer 16-bit
Options/ Padding 0 to 40 bytes								

UDP Header Format:

Source Port Address 16-bit	Destination Port Address 16-bit
Total length of UDP 16-bit	Checksum 16-bit

IPv4 Header Format :

VER 4-bit	HLEN 4-bit	TYPE OF SERVICE 8-bit	Total length 16-bit			
Identification 16-bit			Res	DF	MF	Fragment Offset 13-bit
Time to Live 8-bit	Protocol 8-bit		Header Checksum 16-bit			
Source IP (32-bit)						
Destination IP (32-bit)						
Options + Padding (0-40 bytes)						

IPv6 Header Format :

VER 4-bit	Priority 8-bit	Flow Label 20-bit	
Payload Length 16-bit		Next Header 8-bit	Hop Limits 8-bit
Source IP Address (128-bit)			
Destination IP Address (128-bit)			
Extension Headers			
Data			

3. Network Layer :

Wireshark provides information about IP packets, including the IP header and payload.
You can examine IP packet headers to understand routing, Time-to-Live (TTL), and fragmentation.
ICMP (Internet Control Message Protocol) packets, such as Ping (Echo Request and Reply) and Traceroute (Time Exceeded), can be observed to diagnose network issues.

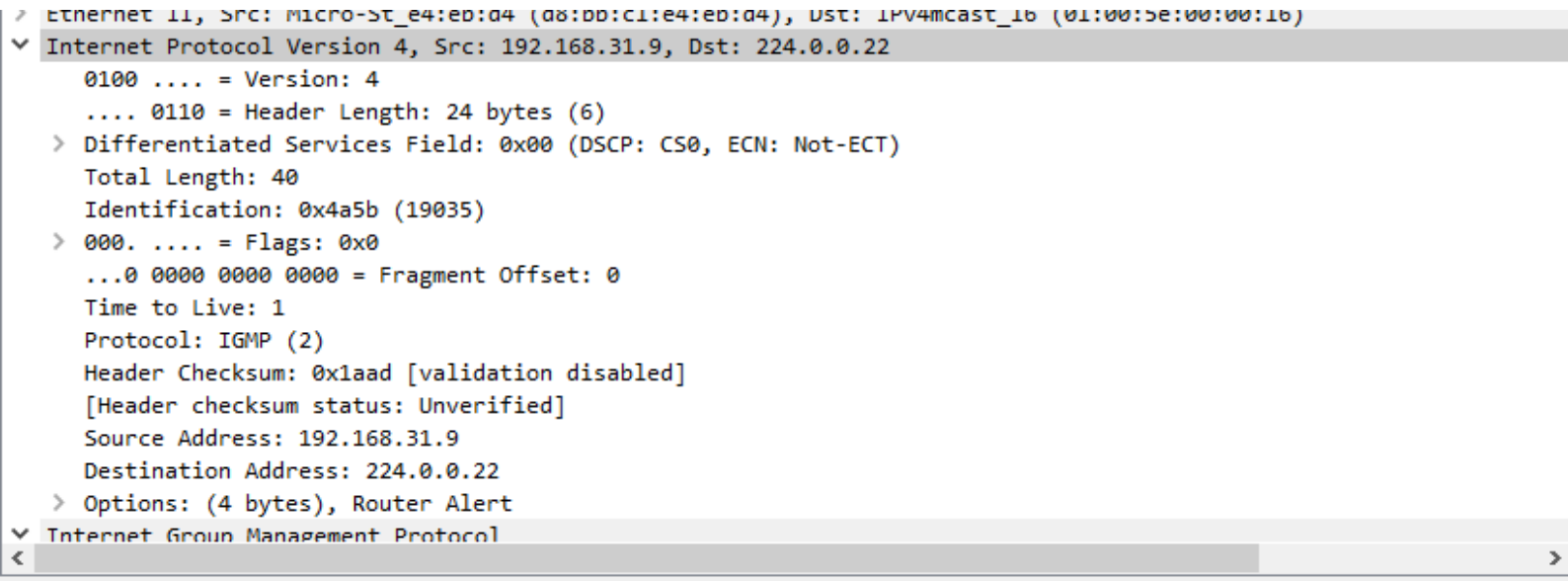
4. Transport Layer :

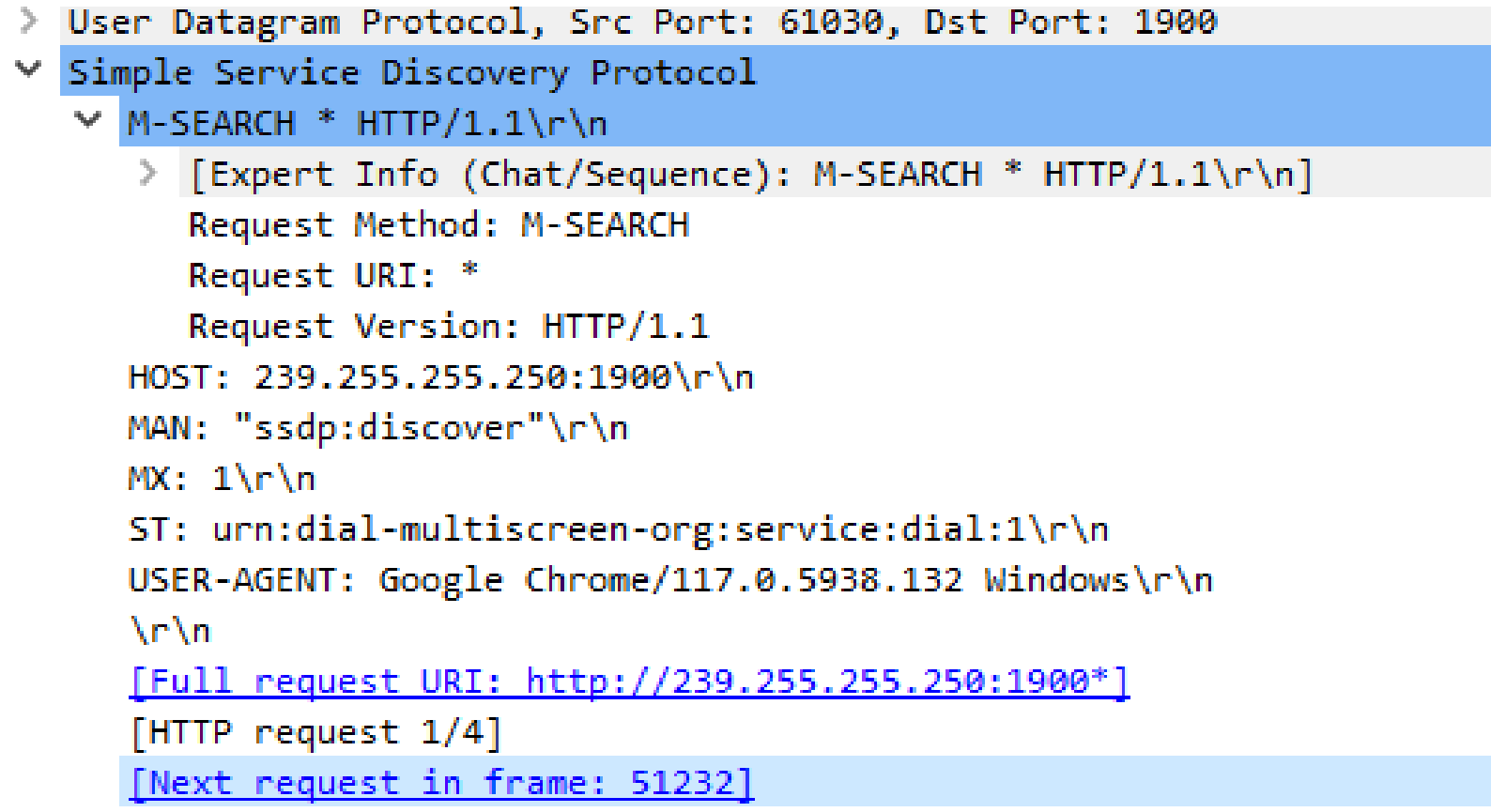
Wireshark captures TCP and UDP packets, allowing you to analyze their characteristics.
For TCP, you can inspect segments, including sequence and acknowledgement numbers.
The TCP handshake (SYN, SYN-ACK, ACK) can be observed, helping you understand how connections are established.
For UDP, you can identify source and destination ports.

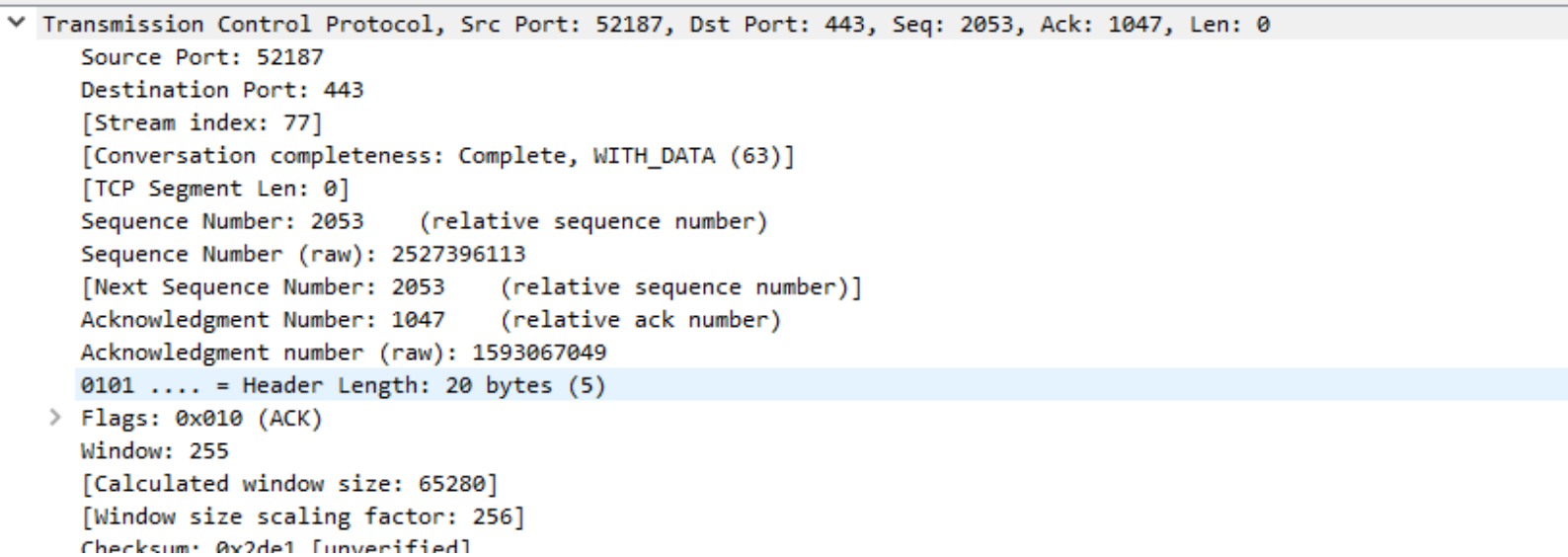
Conclusion :- Thus, we implemented operation of TCP/IP layers using the Wireshark software.

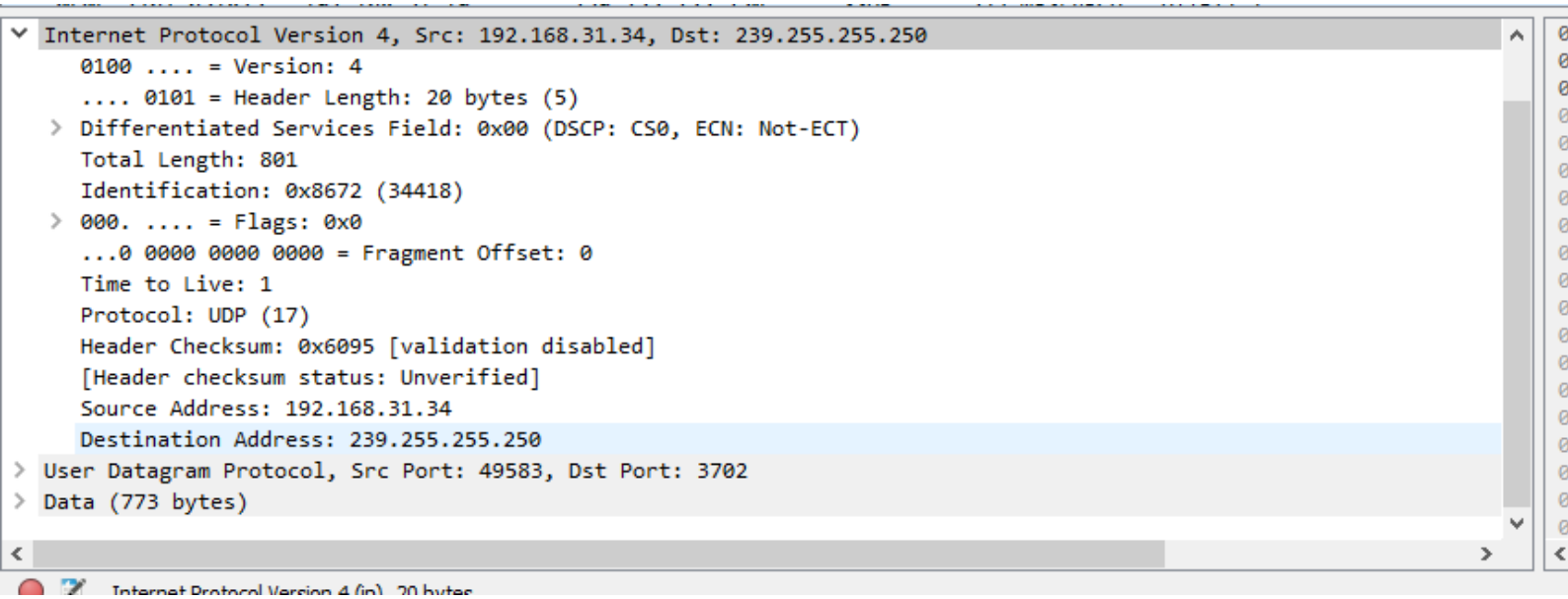
8/8/10/23

```
Ethernet II, Src: Micro-St_c2:99:83 (d8:bb:c1:c2:99:83), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (ARP Probe)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    [Is probe: True]
    Sender MAC address: Micro-St_c2:99:83 (d8:bb:c1:c2:99:83)
    Sender IP address: 0.0.0.0
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.31.18
```


Ethernet II, Src: Micro-St_e4:eb:d4 (d8:bb:c1:e4:eb:d4), Dst: IPv4mcast_16 (01:00:5e:00:00:16)
Internet Protocol Version 4, Src: 192.168.31.9, Dst: 224.0.0.22
0100 .... = Version: 4
.... 0110 = Header Length: 24 bytes (6)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 40
Identification: 0x4a5b (19035)
000. .... = Flags: 0x0
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 1
Protocol: IGMP (2)
Header Checksum: 0x1aad [validation disabled]
[Header checksum status: Unverified]
Source Address: 192.168.31.9
Destination Address: 224.0.0.22
Options: (4 bytes), Router Alert
Internet Group Management Protocol


User Datagram Protocol, Src Port: 61030, Dst Port: 1900
Simple Service Discovery Protocol
M-SEARCH * HTTP/1.1\r\n
[Expert Info (Chat/Sequence): M-SEARCH * HTTP/1.1\r\n]
Request Method: M-SEARCH
Request URI: *
Request Version: HTTP/1.1
HOST: 239.255.255.250:1900\r\n
MAN: "ssdp:discover"\r\n
MX: 1\r\n
ST: urn:dial-multiscreen-org:service:dial:1\r\n
USER-AGENT: Google Chrome/117.0.5938.132 Windows\r\n
\r\n
[Full request URI: http://239.255.255.250:1900*]
[HTTP request 1/4]
[Next request in frame: 51232]


Transmission Control Protocol, Src Port: 52187, Dst Port: 443, Seq: 2053, Ack: 1047, Len: 0
Source Port: 52187
Destination Port: 443
[Stream index: 77]
[Conversation completeness: Complete, WITH_DATA (63)]
[TCP Segment Len: 0]
Sequence Number: 2053 (relative sequence number)
Sequence Number (raw): 2527396113
[Next Sequence Number: 2053 (relative sequence number)]
Acknowledgment Number: 1047 (relative ack number)
Acknowledgment number (raw): 1593067049
0101 .... = Header Length: 20 bytes (5)
Flags: 0x010 (ACK)
Window: 255
[Calculated window size: 65280]
[Window size scaling factor: 256]
Checksum: 0x2de1 [unverified]


Internet Protocol Version 4, Src: 192.168.31.34, Dst: 239.255.255.250
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 801
Identification: 0x8672 (34418)
000. .... = Flags: 0x0
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 1
Protocol: UDP (17)
Header Checksum: 0x6095 [validation disabled]
[Header checksum status: Unverified]
Source Address: 192.168.31.34
Destination Address: 239.255.255.250
User Datagram Protocol, Src Port: 49583, Dst Port: 3702
Data (773 bytes)

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

EXPERIMENT NO. 10

Aim :- Socket programming using TCP or UDP

Theory :-

Socket programming using TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) involves establishing network communication between two devices or processes. Here's a theoretical overview of these two approaches :

1. TCP (Transmission Control Protocol) :

Connection Oriented : TCP is a connection-oriented protocol, which means it establishes a reliable, ordered and error-checked connection between sender and receiver.

Reliability : TCP ensures that data is transmitted accurately and completely. It uses mechanisms like acknowledgements and retransmissions to guarantee data delivery.

Stream-Oriented : TCP is stream-oriented, which implies that it sends and receives data as a continuous stream. It provides a byte-stream service, ensuring data integrity but not preserving message boundaries.

Usage : UDP is suitable for applications where low overhead and speed are more important than data integrity, such as real-time multimedia streaming, online gaming, and DNS.

Socket Functions : In UDP socket programming, commonly used functions include socket(), bind(), sendto(), etc.

In both TCP and UDP socket programming :-

Socket Creation : A socket is created using socket() function, specifying protocol (TCP or UDP) and addressing information.

Binding : The bind() function associates socket with a specific address and port.

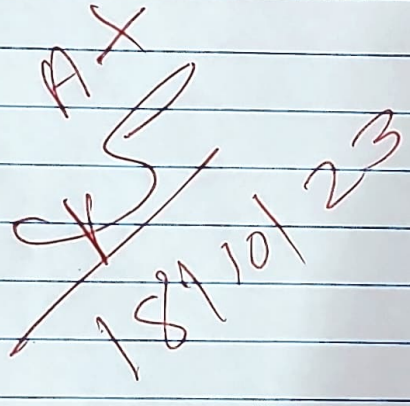
Listening (TCP only) : In TCP, the listen() function prepares socket to accept incoming connections.

Connection Establishment (TCP only) : The connect() function establishes a connection to another device in TCP.

Data Transfer : Data is sent using send() and received using recv() for TCP, while sendto() and recvfrom() are used for UDP.

Termination (TCP only) :- TCP sockets involve a connection termination process, typically initiated by close() function.

Conclusion :- Thus, we implemented socket programming using JAVA language.



```
import java.net.*;
import java.io.*;
import java.util.*;
class MyClient{
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
System.out.print("Enter number: ");
//str=br.readLine();
Scanner sc = new Scanner(System.in);
int n = 10;
n = sc.nextInt();
//dout.writeUTF(str);
//dout.writeUTF("Hello guys Chai Peelo");
dout.writeUTF(Integer.toString(n));
dout.flush();
str2=din.readUTF();
System.out.println("Server says: "+str2);
}

dout.close();
s.close();
}}

import java.net.*;
```

```
import java.io.*;
class Server{
public static void main(String args[])throws Exception{
ServerSocket ss=new ServerSocket(3333);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String str="",str2="";String num;
while(!str.equals("stop")){
num=din.readUTF();
//System.out.println("client says: "+str);
System.out.println("Number given by client: "+num);
int n = Integer.parseInt(num);
int square = n*n;
System.out.println("Square of Number given by client: "+square);
//str2=br.readLine();
dout.writeUTF("Square of Number given by you: "+square);
dout.flush();
}
din.close();
s.close();
ss.close();
}}
```

Output:

Client:

```
Microsoft Windows [Version 10.0.22621.2428]
```

(c) Microsoft Corporation. All rights reserved.

C:\Users\Rishab\OneDrive\Desktop\CN Experiments>javac MyClient.java

C:\Users\Rishab\OneDrive\Desktop\CN Experiments>java MyClient

Enter number: 4

Server says: Square of Number given by you: 16

Server:

Microsoft Windows [Version 10.0.22621.2428]

(c) Microsoft Corporation. All rights reserved.

C:\Users\Rishab\OneDrive\Desktop\CN Experiments>javac Server.java

C:\Users\Rishab\OneDrive\Desktop\CN Experiments>java Server

Number given by client: 4

Square of Number given by client: 16