Online Internet Protocols and Networking

(CSE 414L) – Lab Report

Thadimarri Sameer

AP20110010028

CSE-A

LAB – 1

(NS-3)

Regarding lab: 1)Install Ubuntu 22.XX (xx -means any version after in 22)

2) install ns3

3) run sample file

4) take a screenshot of the example file output and make a PDF document.

Note: you can refer to the following YouTube links for installation.

1) https://www.youtube.com/watch?v=3lWeCGPiWWM&t=51s

2) installing ns3 on ubuntu: https://www.youtube.com/watch?v=3lWeCGPiWWM&t=51s

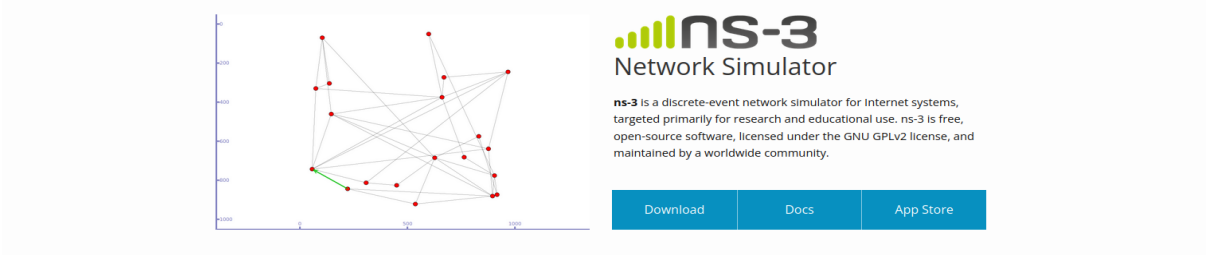
3) Those who are using Mac can work either with ns3 or omnet++

Step-1: Prerequisites

```
┌──(sameer㉿kali)-[~]
└─$ sudo apt update
[sudo] password for sameer:
Get:1 http://kali.download/kali kali-rolling InRelease [41.2 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [19.5 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [46.0
MB]
Get:4 https://mirrors.ocf.berkeley.edu/kali kali-rolling InRelease [41.2 kB]
Get:5 https://mirrors.ocf.berkeley.edu/kali kali-rolling/main amd64 Packages
[19.5 MB]
Get:6 https://mirrors.ocf.berkeley.edu/kali kali-rolling/main amd64 Contents
(deb) [46.0 MB]
Get:7 http://kali.download/kali kali-rolling/contrib amd64 Packages [124 kB]
Get:8 http://kali.download/kali kali-rolling/contrib amd64 Contents (deb) [29
7 kB]
Get:9 http://kali.download/kali kali-rolling/non-free amd64 Packages [226 kB]
Get:10 https://mirrors.ocf.berkeley.edu/kali kali-rolling/contrib amd64 Packa
ges [124 kB]
Get:11 https://mirrors.ocf.berkeley.edu/kali kali-rolling/contrib amd64 Conte
nts (deb) [297 kB]
Get:12 https://mirrors.ocf.berkeley.edu/kali kali-rolling/non-free amd64 Pack
ages [226 kB]
Fetched 132 MB in 21s (6,200 kB/s)
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
1053 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

```
┌──(sameer㉿kali)-[~]
└─$ sudo apt install g++ python3 python3-dev pkg-config sqlite3 cmake python3-setuptools git qtbase5-dev qtchooser q
t5-qmake qtbase5-dev-tools gir1.2-goocanvas-2.0 python3-gi python3-gi-cairo python3-pygraphviz gir1.2-gtk-3.0 ipytho
n3 openmpi-bin openmpi-common openmpi-doc libopenmpi-dev autoconf cvs bzr unrar gsl-bin libgsl-dev libgslcblas0 wire
shark tcpdump sqlite3 libsqlite3-dev  libxml2 libxml2-dev libc6-dev libc6-dev-i386 libclang-dev llvm-dev automake py
thon3-pip libxml2 libxml2-dev libboost-all-dev
```

Step-2: Installing NS3 Network Simulator



```
┌──(sameer㉿kali)-[~/Downloads]
└─$ tar jxvf ns-allinone-3.40.tar.bz2
```

```
┌──(sameer㉿kali)-[~/ns-allinone-3.40]
└─$ ./build.py --enable-examples --enable-tests
```

Step-3: Running NS3 Network Simulator

```
┌──(sameer㉿kali)-[~]
└─$ cd ns-allinone-3.40

┌──(sameer㉿kali)-[~/ns-allinone-3.40]
└─$ cd ns-3.40

┌──(sameer㉿kali)-[~/ns-allinone-3.40/ns-3.40]
└─$ ./ns3 run hello-simulator
Hello Simulator
```

```
┌──(sameer㉿kali)-[~/ns-allinone-3.40/ns-3.40]
└─$ cp examples/tutorial/first.cc scratch/

┌──(sameer㉿kali)-[~/ns-allinone-3.40/ns-3.40]
└─$ ./ns3 run scratch/first
-- Using default output directory /home/sameer/ns-allinone-3.40/ns-3.40/build
```

LAB – 2

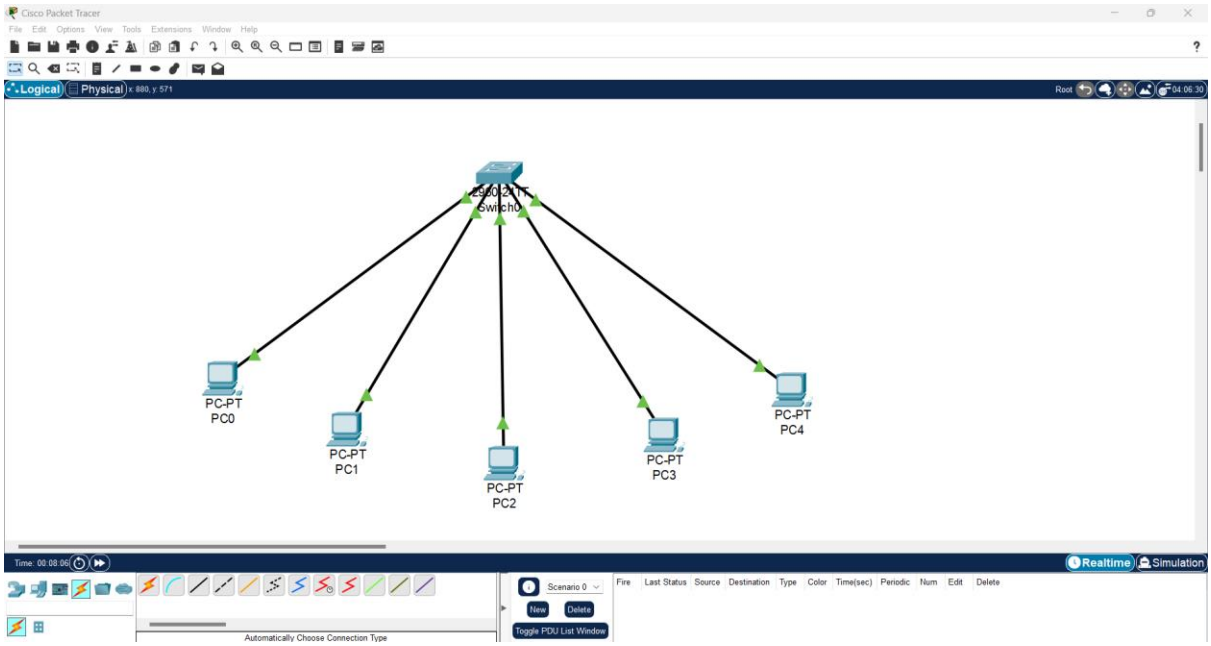
(Packet Tracer)

Regarding lab: Using Packet Tracer
task 1: Lan Network: using switch
task 2: Perform Static Routing
task 3: Perform Dynamic Routing
task 4: Do the VLAN configuration

TASK – 1: Lan Network Using Switch

Packet Tracer Used – Cisco Packet Tracer (CPT)

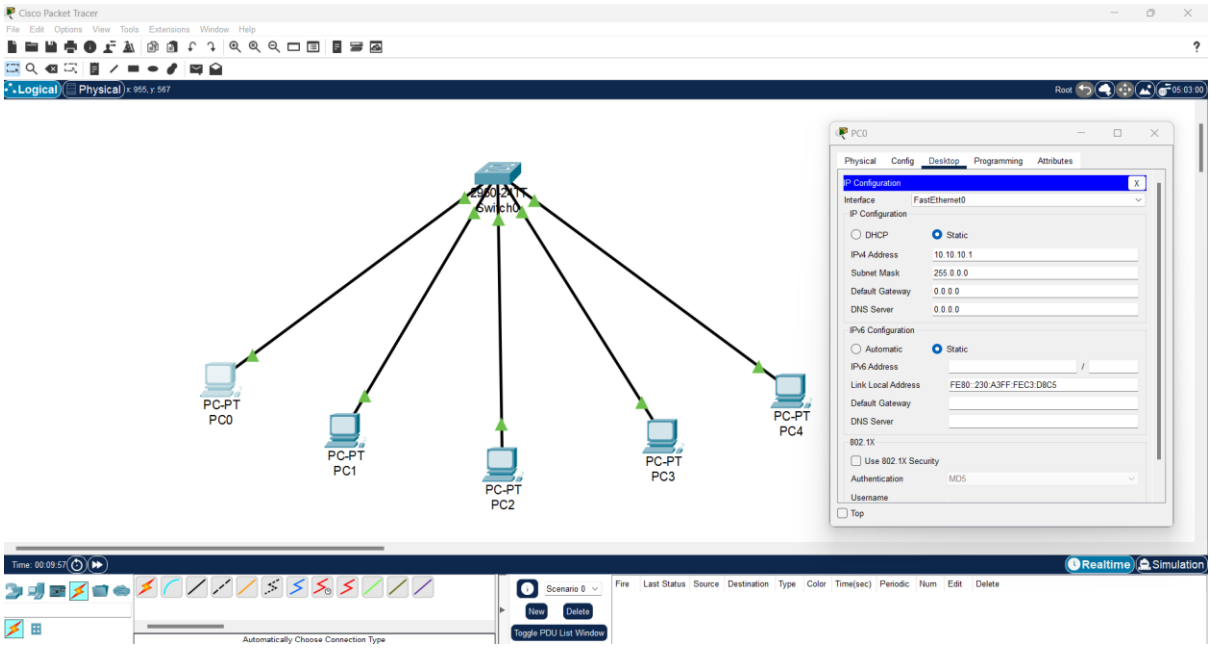
Step-1: Creating a LAN (Local Area Network) Using Switch.

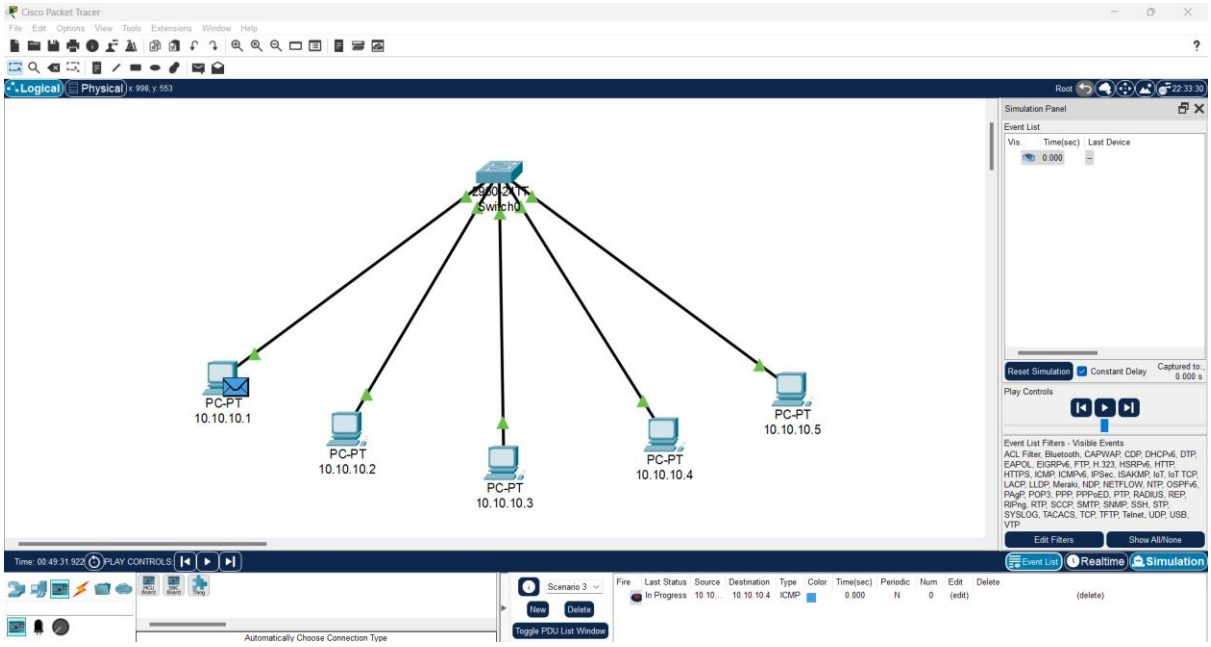


Switch is used to connect 5 PC's

Step-2: IP Configuration

Configuring IP address and Subnet Mask of each PC in the LAN





PC0 – 10.10.10.1 (IPV4 Address)

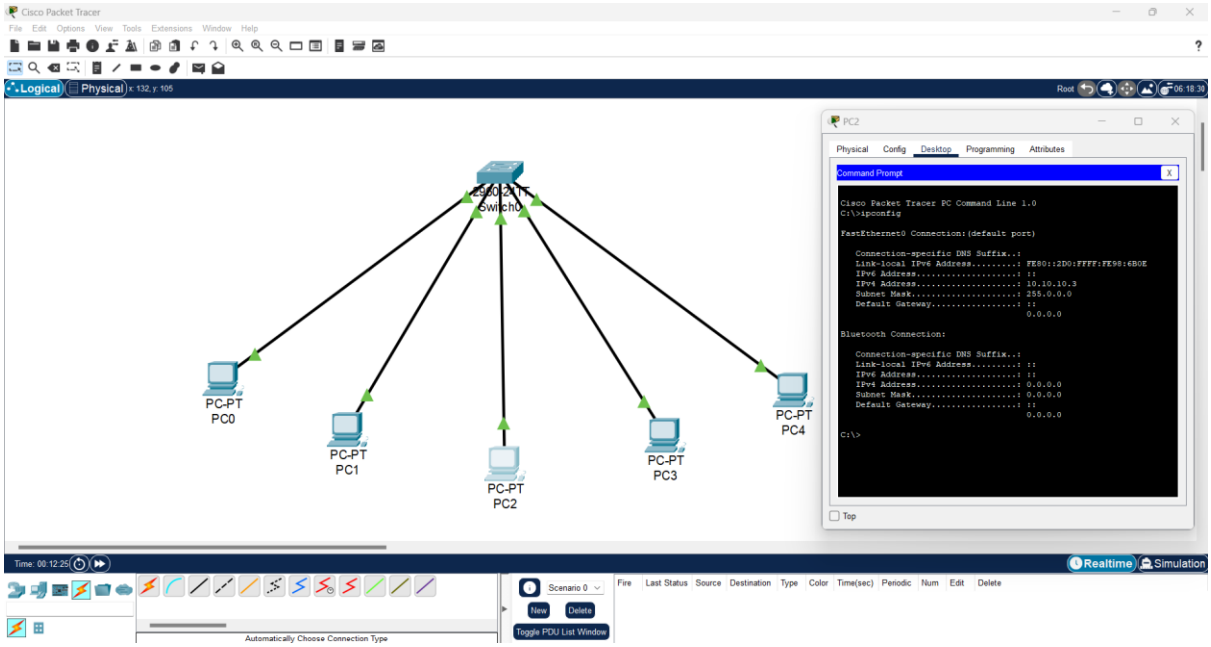
PC1 – 10.10.10.2 (IPV4 Address)

PC3 – 10.10.10.3 (IPV4 Address)

PC4 – 10.10.10.4 (IPV4 Address)

PC5 – 10.10.10.5 (IPV4 Address)

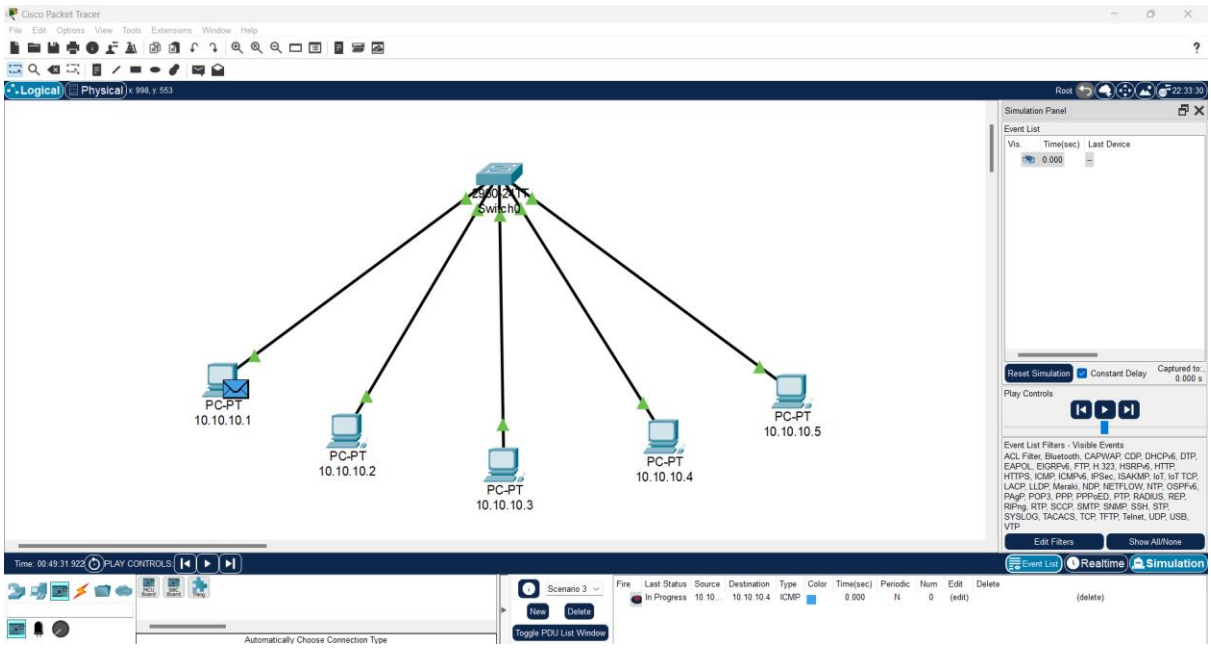
Step-3: Checking IP Address Using IP Config in Command Prompt

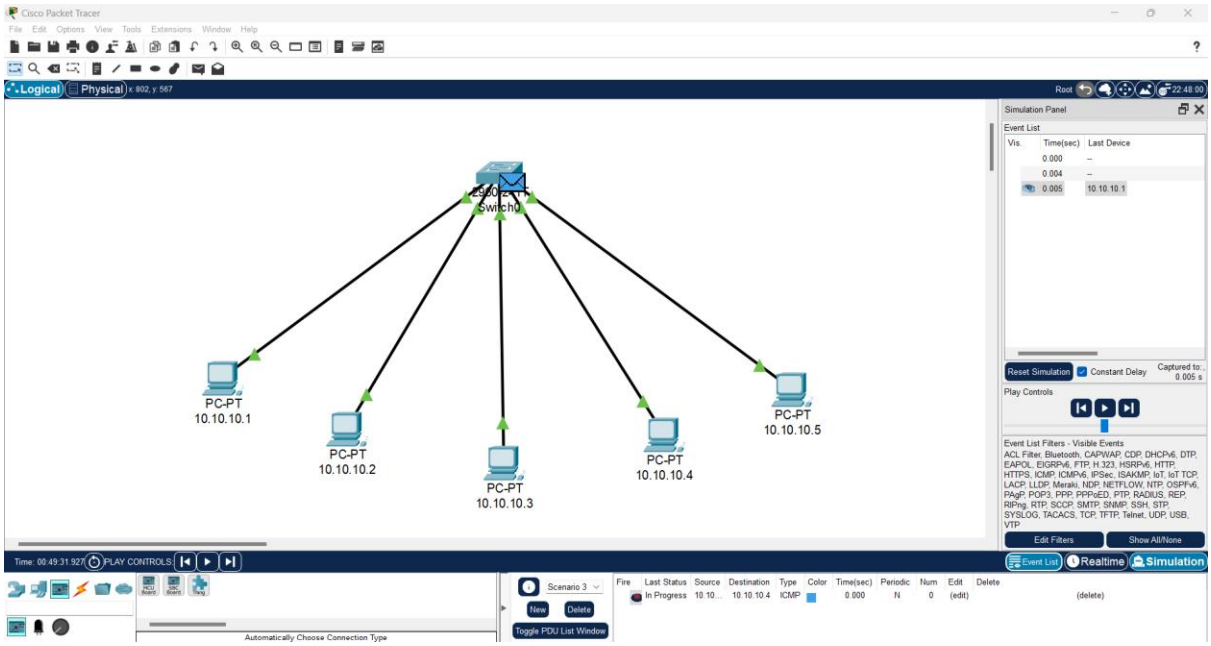


Step-4: Simulation

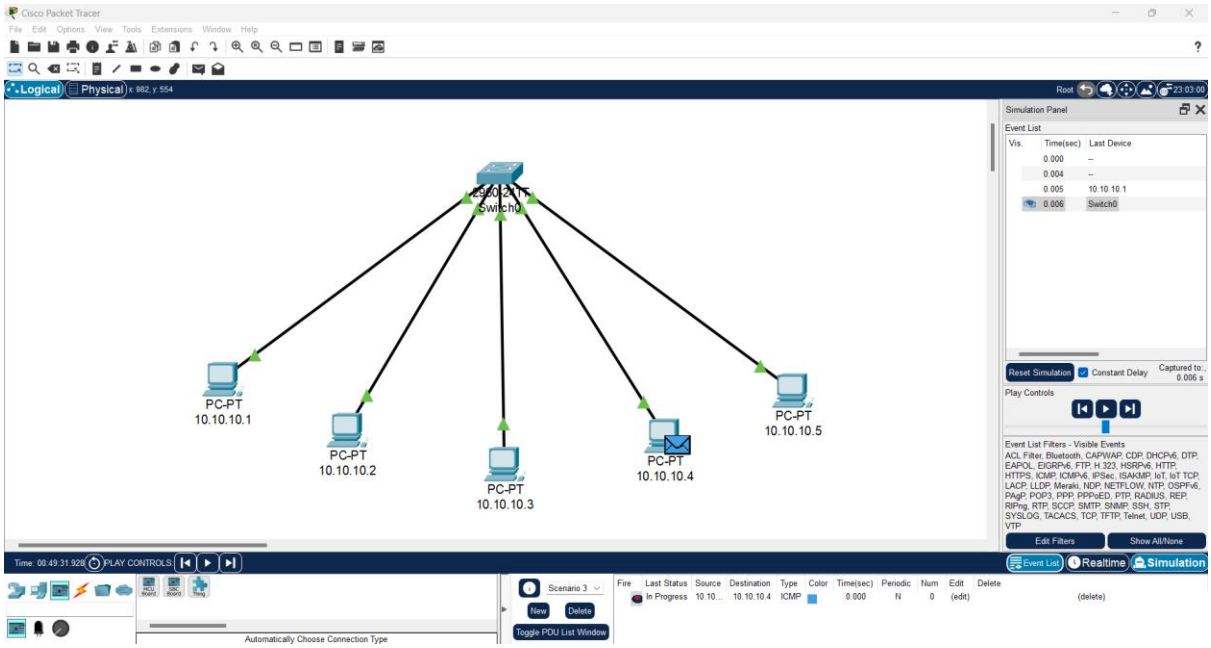
- Send ICMP (Internet Control Message Protocol) Packet from One PC to other PC through Switch
- Sending ICMP Packet From Packet–1 (10.10.10.1) to Packet – 4(10.10.10.4)

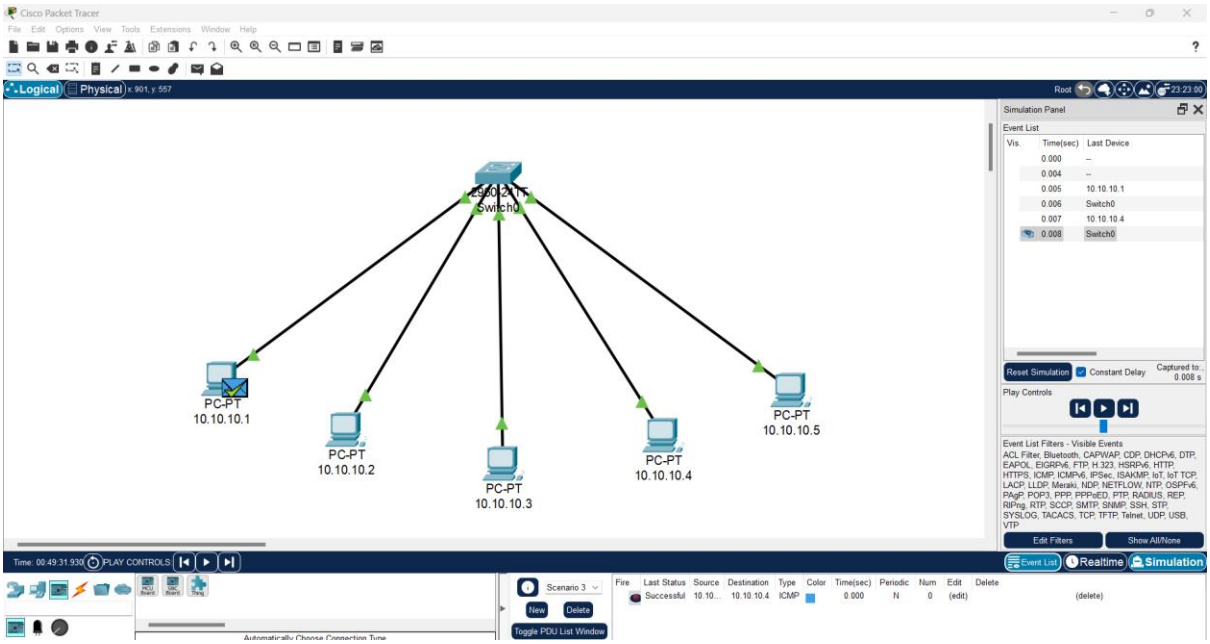
Packet-1(10.10.10.1) → Switch



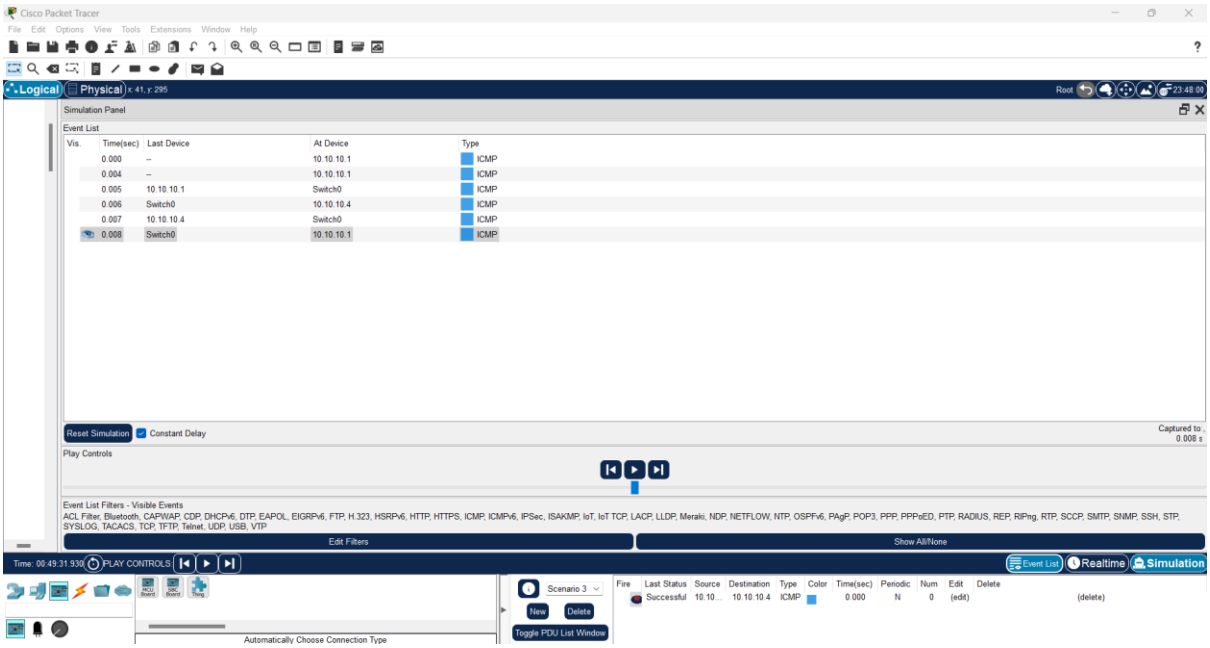


Switch → Packet-4(10.10.10.4)

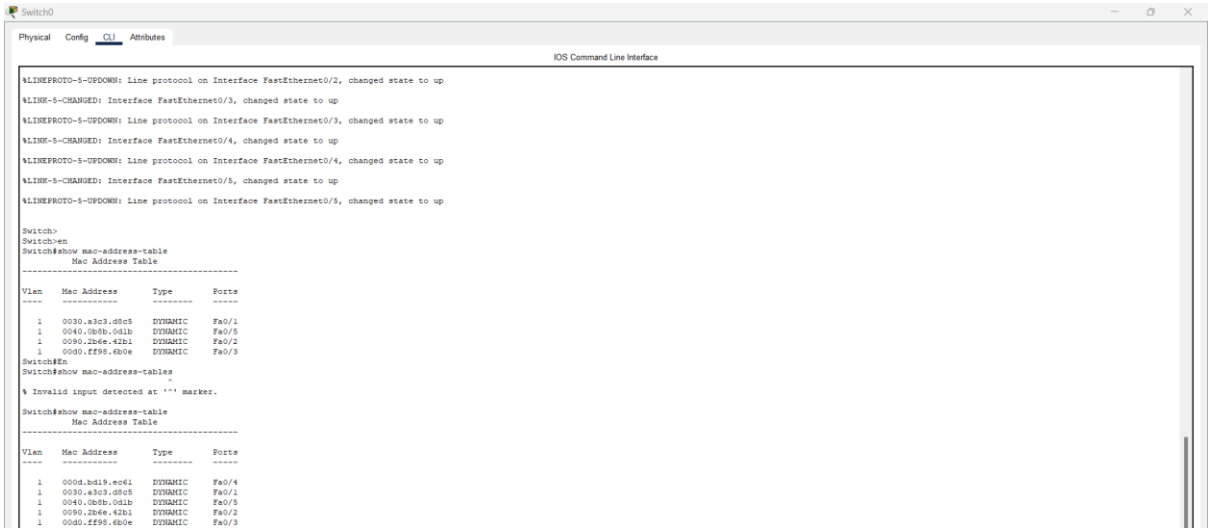




Packet Trace Information



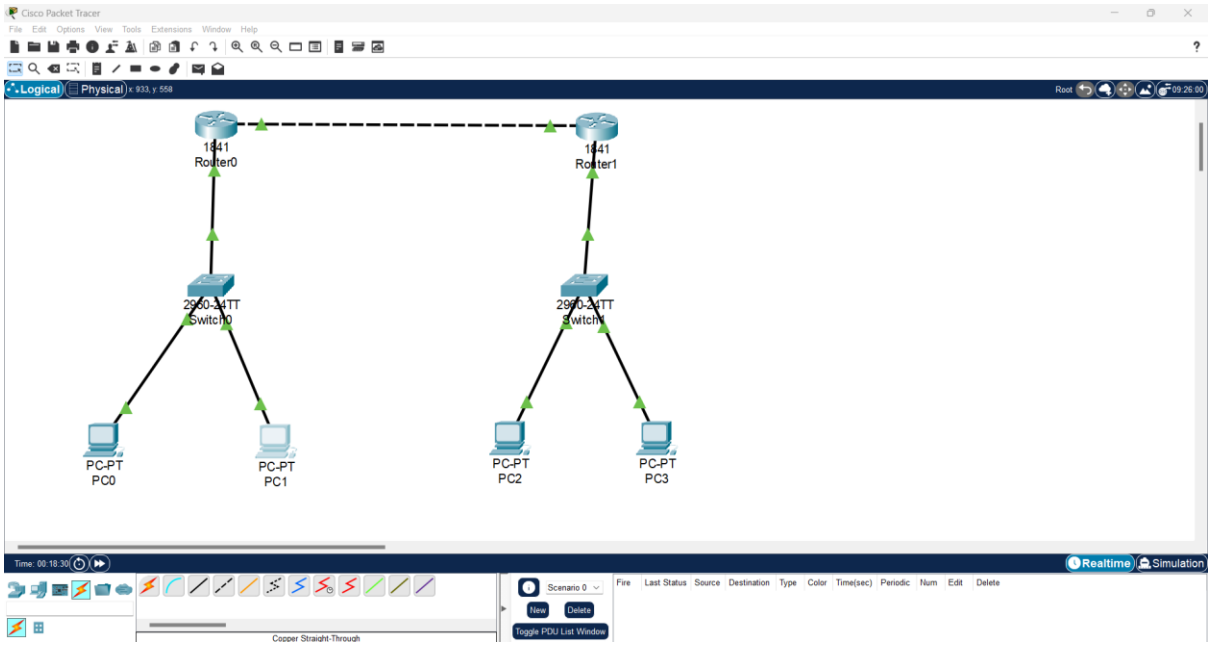
STEP-5: MAC (Medium Access Control) Table



TASK – 2: Performing Static Routing

Packet Tracer Used: Cisco Packet Tracer (CPT)

Step-1: Creating a Static Routing Network



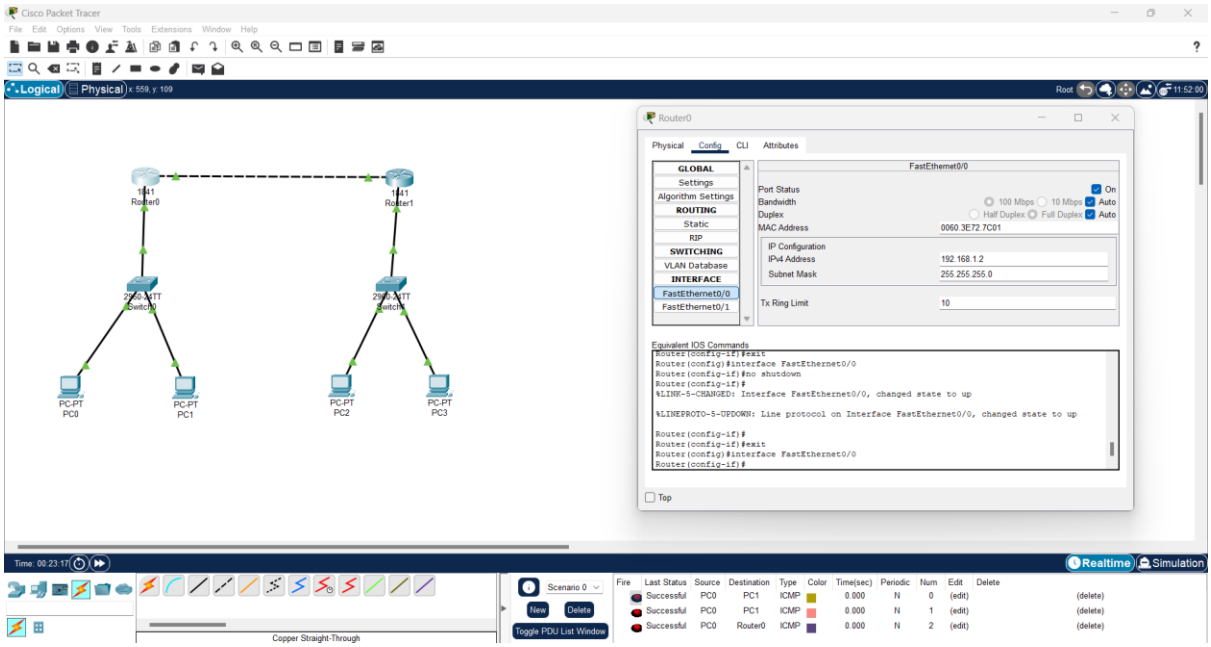
2 Routers

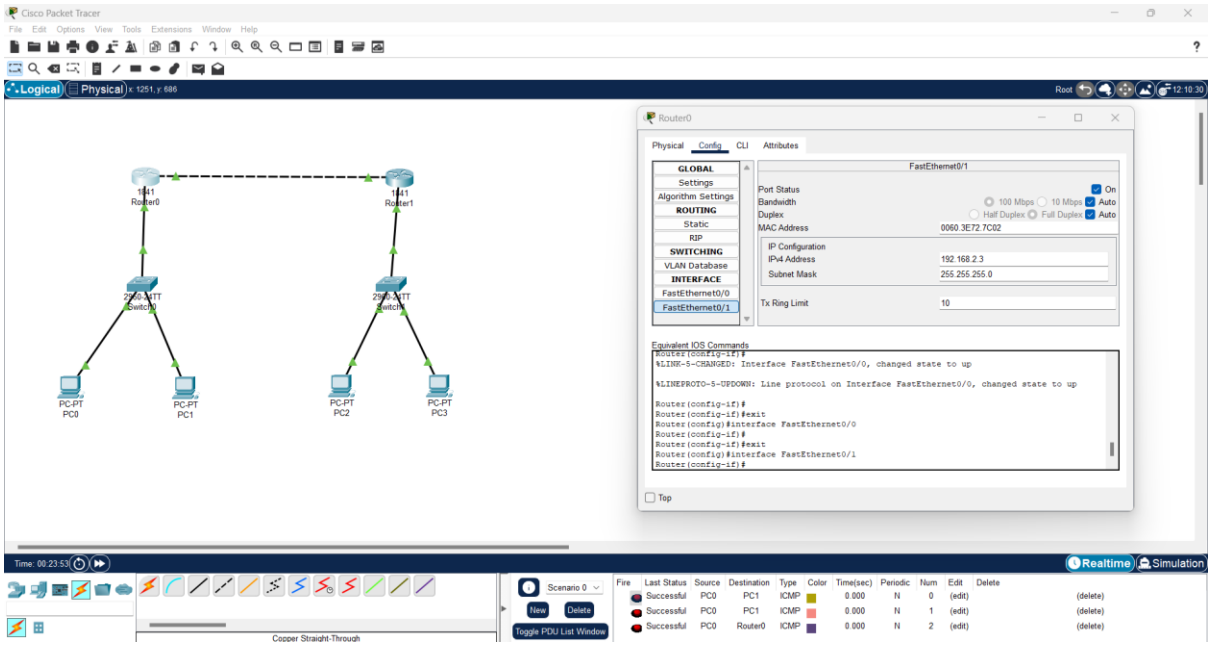
2 Switches

4 PC's

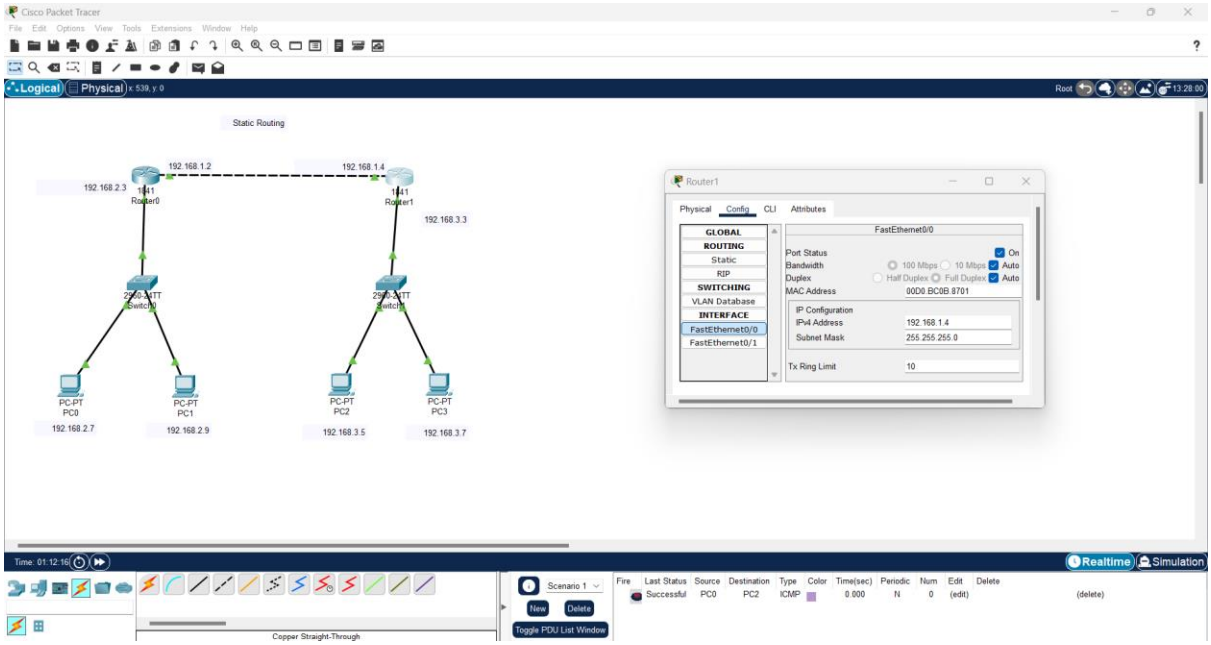
Step-2: IP Configuration of Routers and PC's

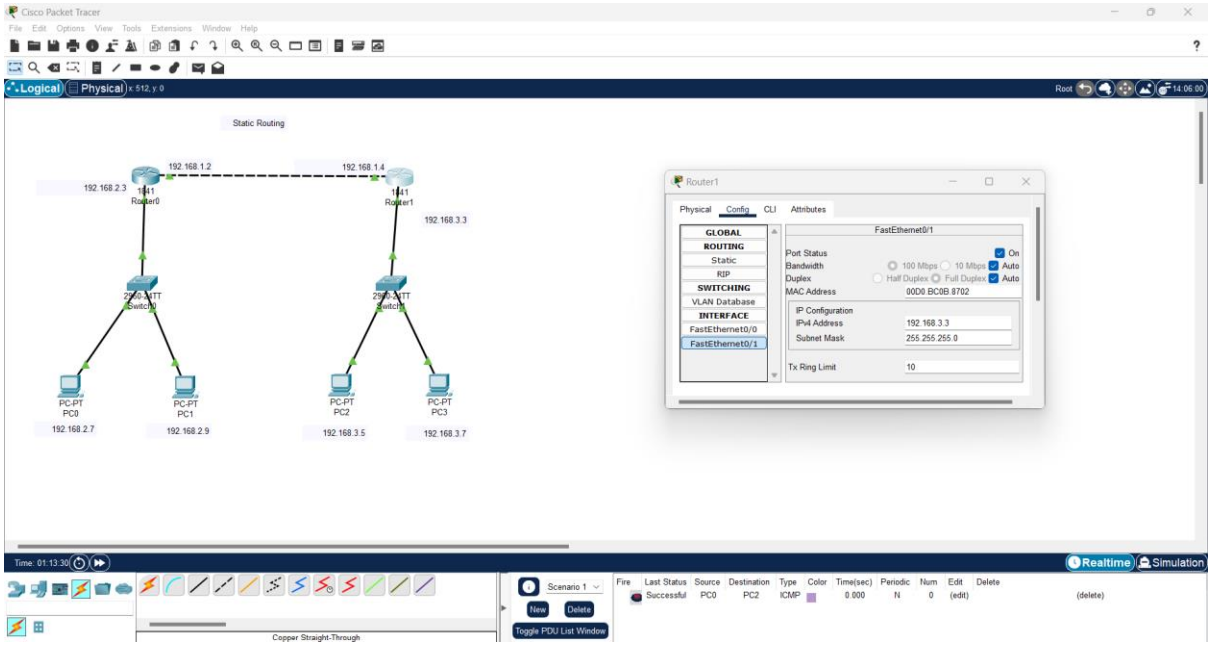
Router-1:



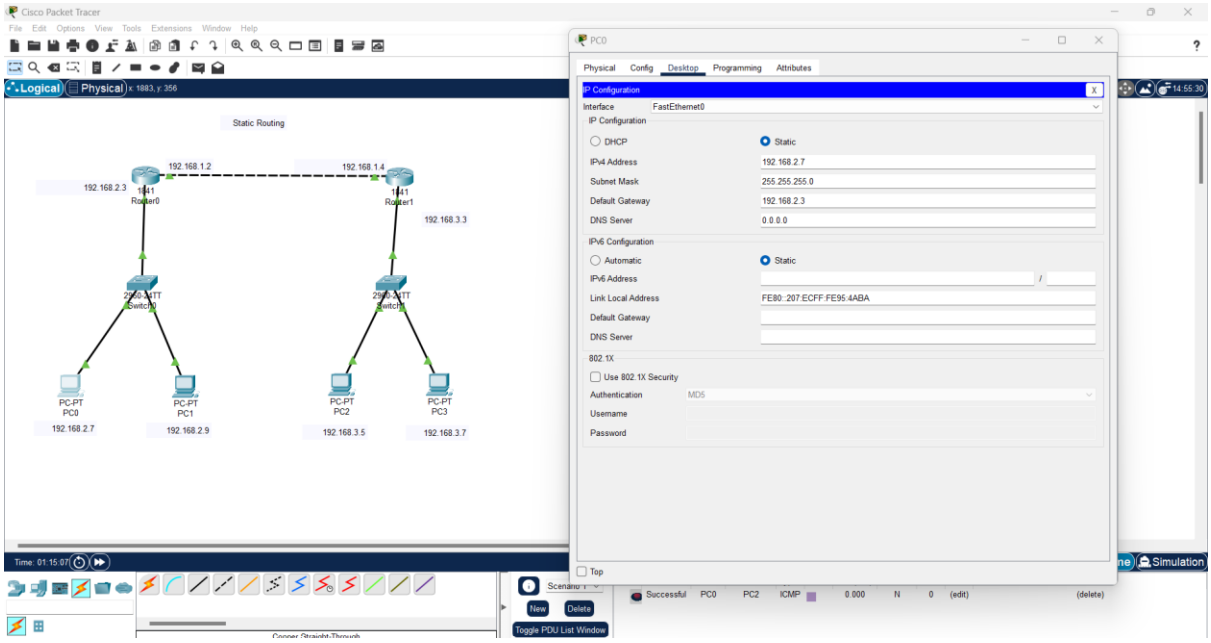


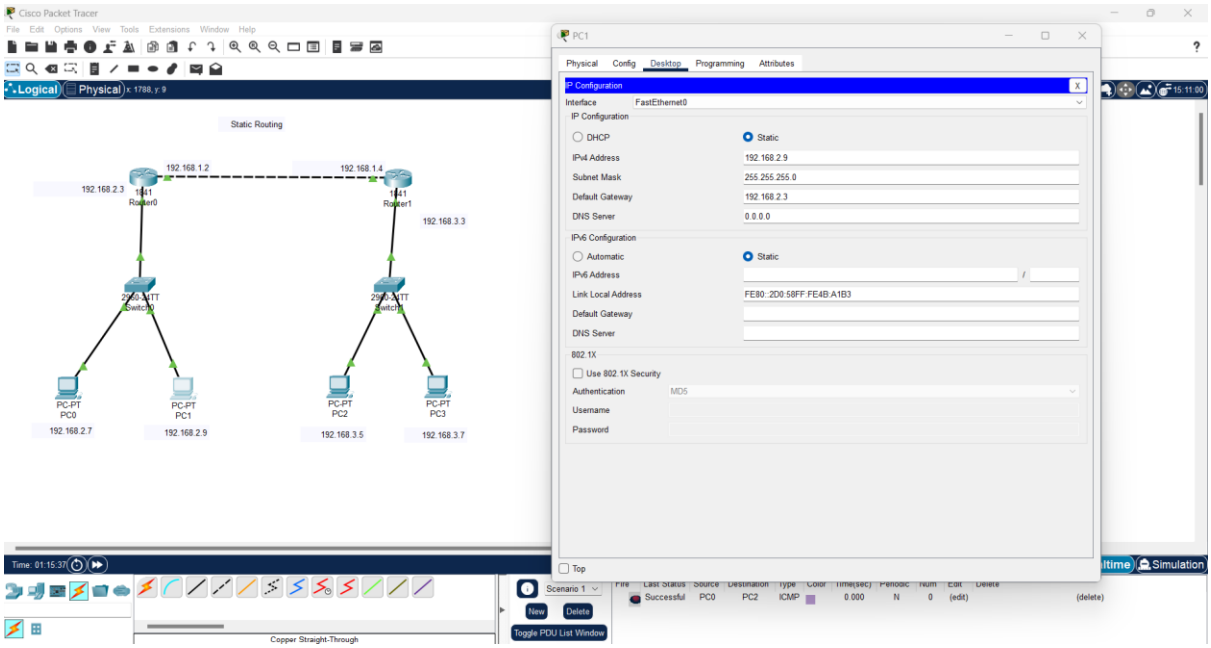
Router-2:

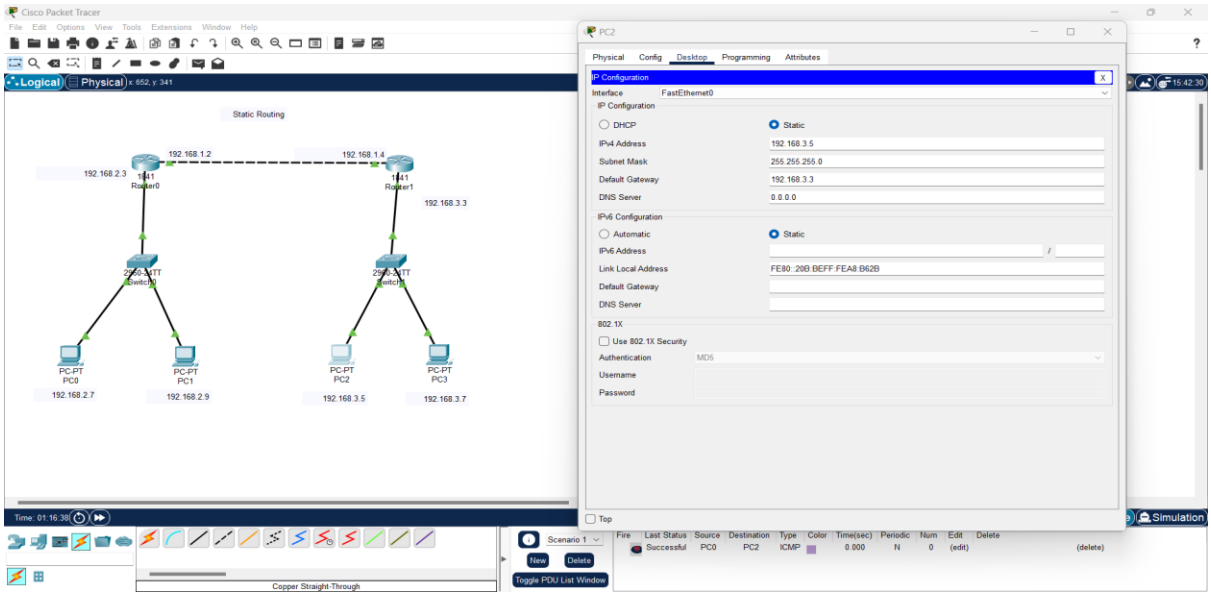

Cisco Packet Tracer
Logical
Physical
Static Routing
192.168.1.2
192.168.1.4
192.168.2.3
192.168.3.3
Router0
Router1
PC-PT PC0
PC-PT PC1
PC-PT PC2
PC-PT PC3
192.168.2.7
192.168.2.9
192.168.3.5
192.168.3.7
Router1
Physical
Config
CLI
Attributes
GLOBAL
ROUTING
Static
RIP
SWITCHING
VLAN Database
INTERFACE
FastEthernet0/0
FastEthernet0/1
Port Status
Bandwidth
Duplex
MAC Address
00D0.BC0B.8701
IP Configuration
IPv4 Address
192.168.1.4
Subnet Mask
255.255.255.0
Tx Ring Limit
10
Realtime
Simulation
Scenario 1
New
Delete
Toggle PDU List Window
Copper Straight-Through
Fire
Last Status
Source
Destination
Type
Color
Time(sec)
Periodic
Num
Edit
Delete
Successful
PC0
PC2
ICMP
0.000
N
0
(edit)
(delete)

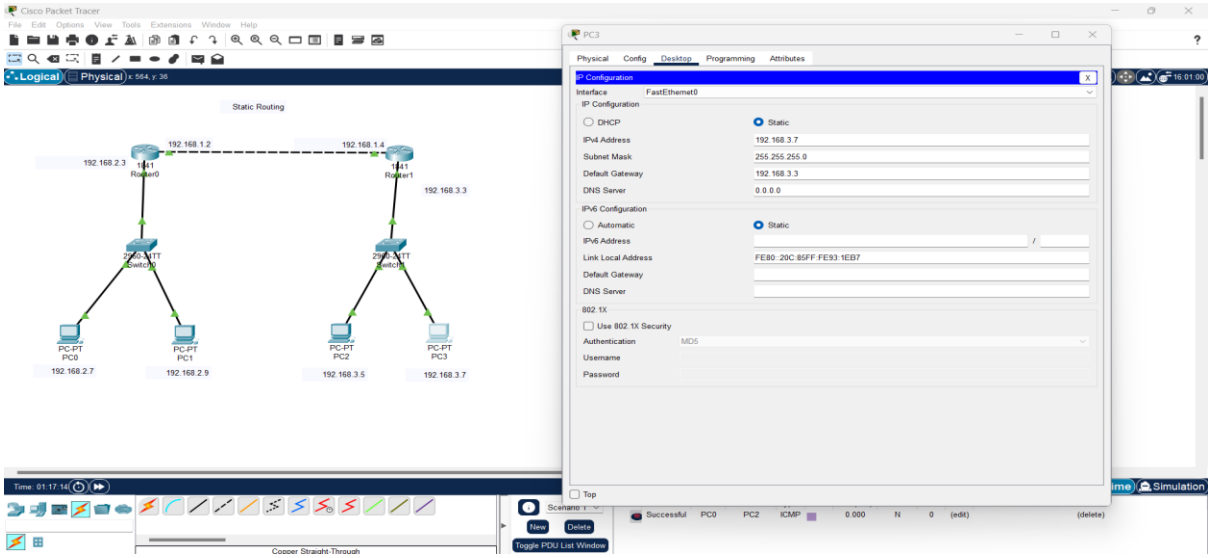

Cisco Packet Tracer
Logical
Physical
Static Routing
192.168.1.2
192.168.1.4
192.168.2.3
192.168.3.3
Router0
Router1
PC-PT PC0
PC-PT PC1
PC-PT PC2
PC-PT PC3
192.168.2.7
192.168.2.9
192.168.3.5
192.168.3.7
Router1
Physical
Config
CLI
Attributes
GLOBAL
ROUTING
Static
RIP
SWITCHING
VLAN Database
INTERFACE
FastEthernet0/0
FastEthernet0/1
Port Status
Bandwidth
Duplex
MAC Address
00D0.BC0B.8702
IP Configuration
IPv4 Address
192.168.3.3
Subnet Mask
255.255.255.0
Tx Ring Limit
10
Realtime
Simulation
Scenario 1
New
Delete
Toggle PDU List Window
Copper Straight-Through
Fire
Last Status
Source
Destination
Type
Color
Time(sec)
Periodic
Num
Edit
Delete
Successful
PC0
PC2
ICMP
0.000
N
0
(edit)
(delete)

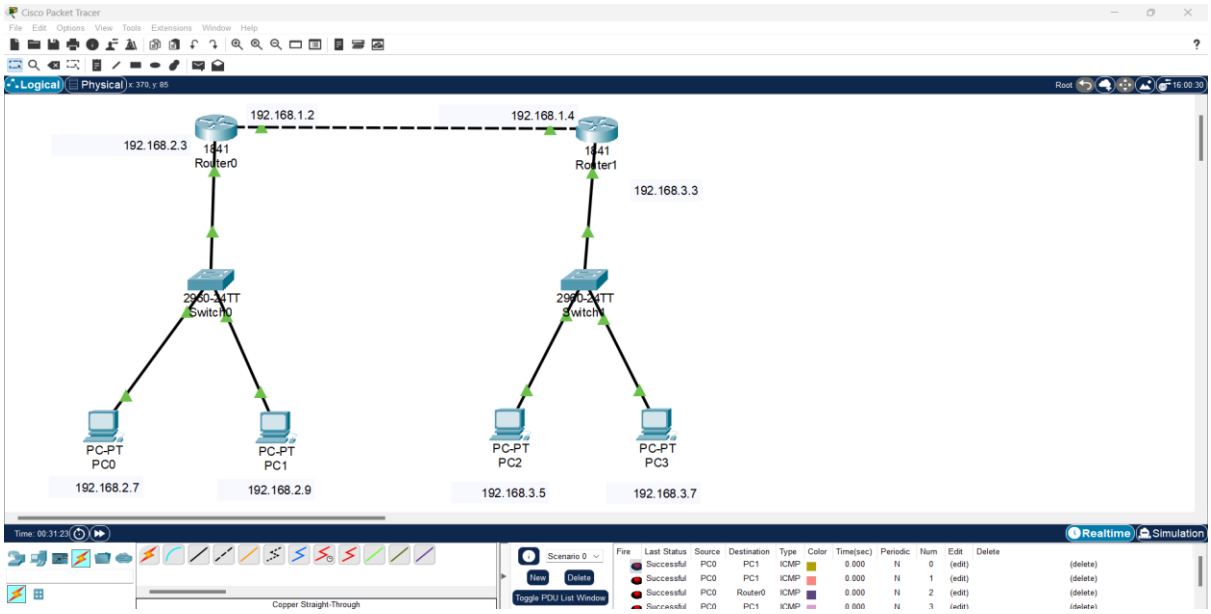
PC's IP Configuration:



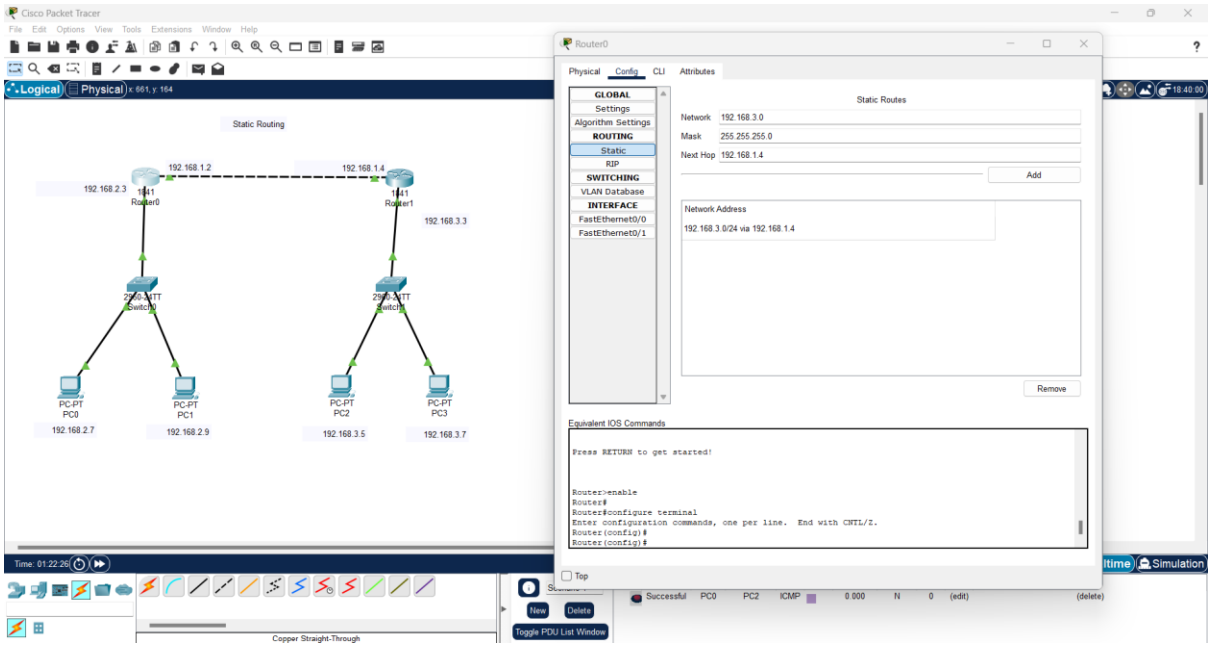


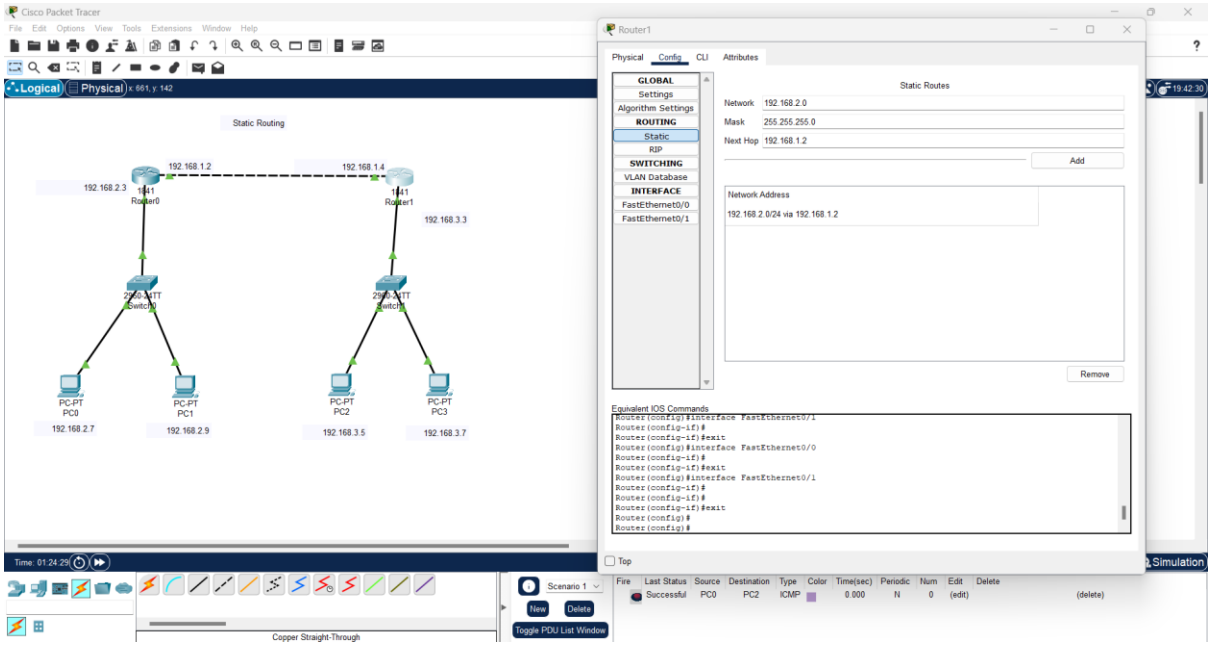

Cisco Packet Tracer
Static Routing
192.168.1.2
192.168.1.4
192.168.2.3
1841
Router0
Router1
192.168.3.3
2960-24TT
Switch0
Switch1
PC-PT
PC0
PC1
PC2
PC3
192.168.2.7
192.168.2.9
192.168.3.5
192.168.3.7
PC2
Physical
Config
Desktop
Programming
Attributes
IP Configuration
Interface
FastEthernet0
DHCP
Static
IPv4 Address
192.168.3.5
Subnet Mask
255.255.255.0
Default Gateway
192.168.3.3
DNS Server
0.0.0.0
IPv6 Configuration
Automatic
Static
IPv6 Address
Link Local Address
FE80::20B:BEFF:FEA8:B62B
802.1X
Use 802.1X Security
Authentication
MD5
Username
Password
Top
Logical
Physical
Simulation
Copper Straight-Through
Scenario 1
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time(sec)
Periodic
Num
Edit
Delete
Successful
PC0
PC2
ICMP
0.000
N
0
(edit)
(delete)


Cisco Packet Tracer
Static Routing
192.168.1.2
192.168.1.4
192.168.2.3
1841
Router0
Router1
192.168.3.3
2960-24TT
Switch0
Switch1
PC-PT
PC0
PC1
PC2
PC3
192.168.2.7
192.168.2.9
192.168.3.5
192.168.3.7
PC3
Physical
Config
Desktop
Programming
Attributes
IP Configuration
Interface
FastEthernet0
DHCP
Static
IPv4 Address
192.168.3.7
Subnet Mask
255.255.255.0
Default Gateway
192.168.3.3
DNS Server
0.0.0.0
IPv6 Configuration
Automatic
Static
IPv6 Address
Link Local Address
FE80::20C:85FF:FE93:1EB7
802.1X
Use 802.1X Security
Authentication
MD5
Username
Password
Top
Logical
Physical
Simulation
Copper Straight-Through
Scenario 1
New
Delete
Toggle PDU List Window
Successful
PC0
PC2
ICMP
0.000
N
0
(edit)
(delete)


Cisco Packet Tracer
Logical
Physical
Root
192.168.1.2
192.168.1.4
192.168.2.3
1841
Router0
Router1
192.168.3.3
2960-24TT
Switch0
Switch1
PC-PT
PC0
PC1
PC2
PC3
192.168.2.7
192.168.2.9
192.168.3.5
192.168.3.7
Realtime
Simulation
Copper Straight-Through
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time(sec)
Periodic
Num
Edit
Delete
Successful
PC0
PC1
ICMP
0.000
N
0
(edit)
(delete)
Successful
PC0
PC1
ICMP
0.000
N
1
(edit)
(delete)
Successful
PC0
Router0
ICMP
0.000
N
2
(edit)
(delete)

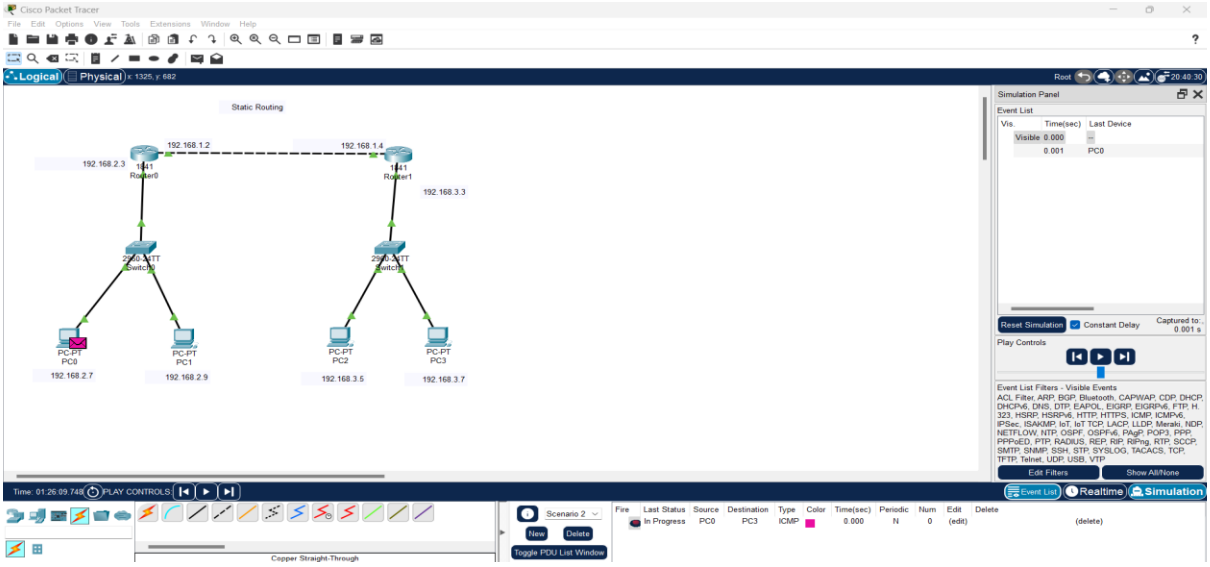
Step-3: Configuring Static Routes in the Network

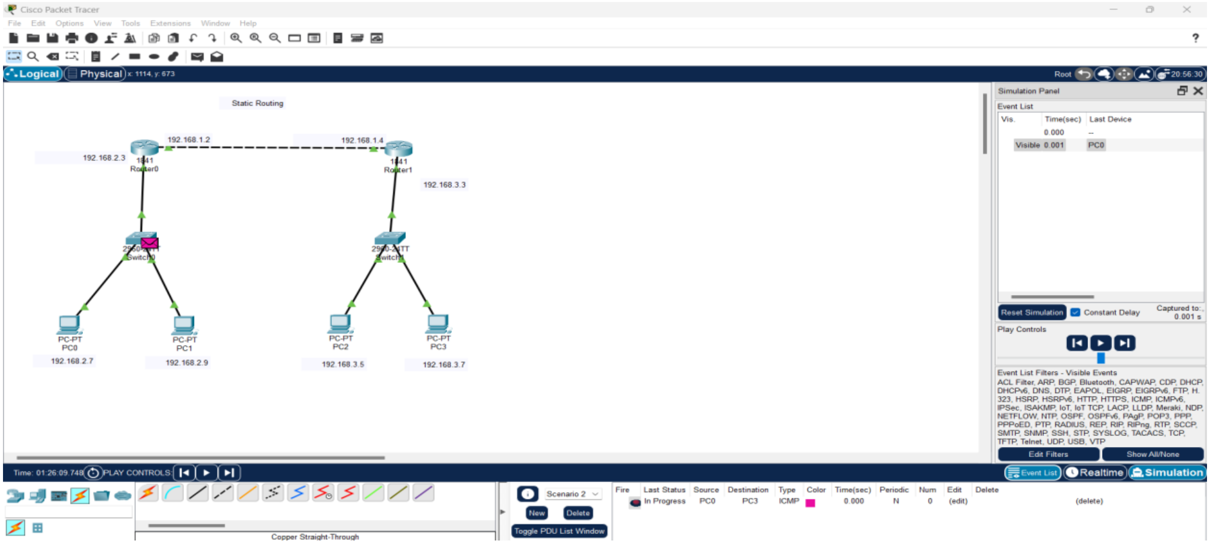


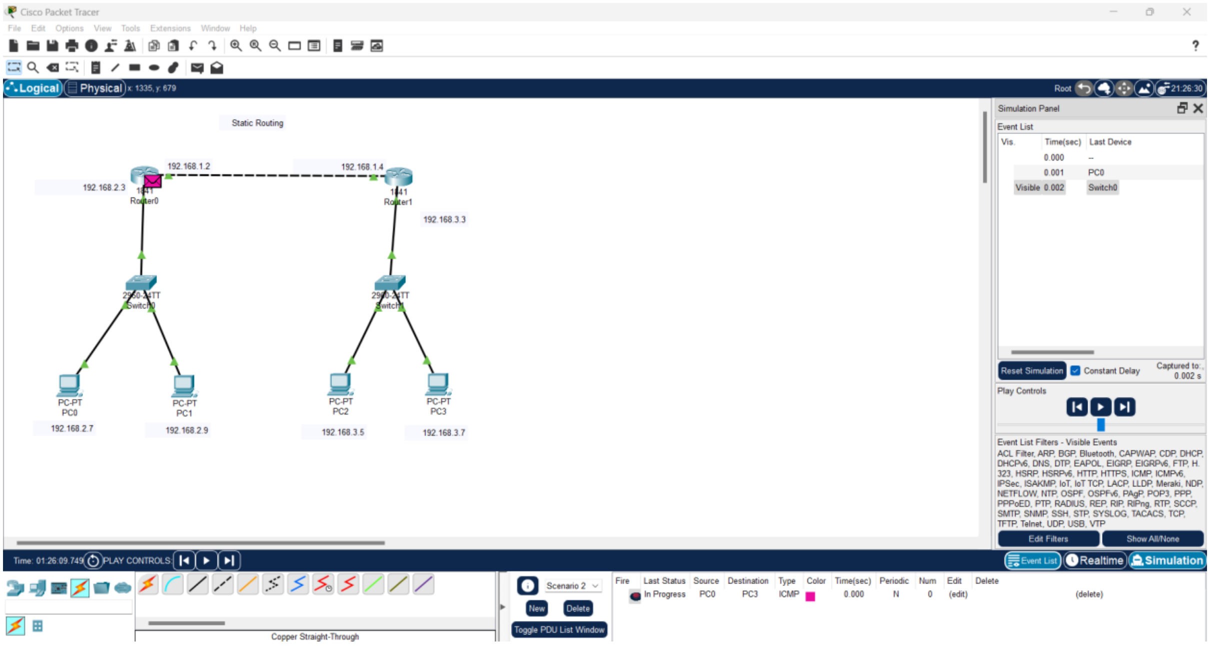


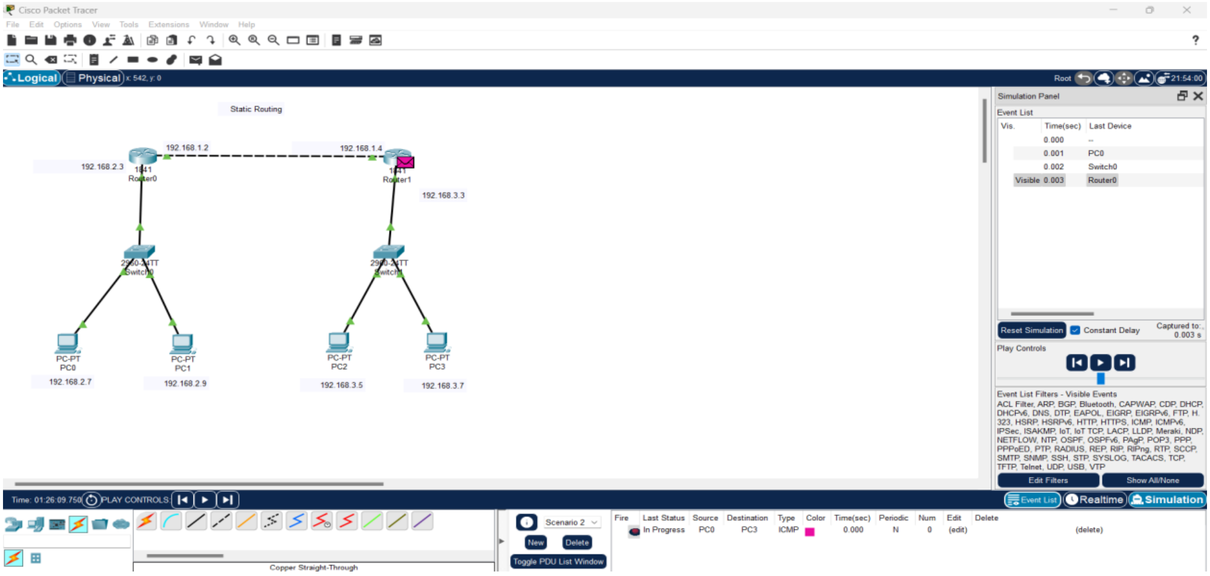
Step-4: Simulating Static Routing

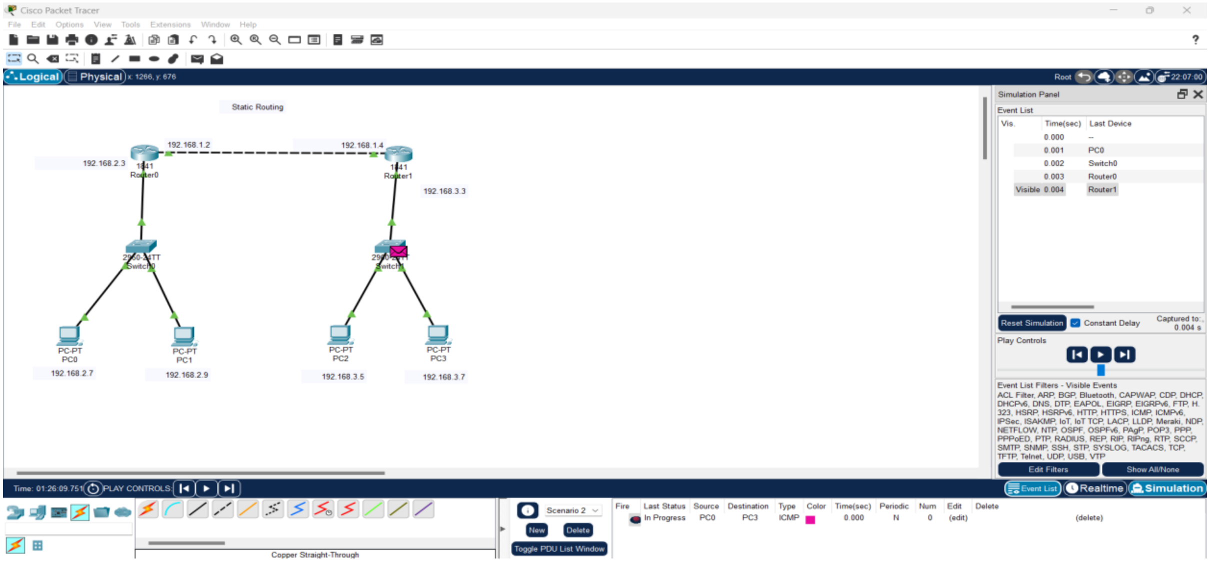
- Send ICMP (Internet Control Message Protocol) Packet from One PC to other PC through Switch and Router
- Sending ICMP Packet From Packet–1 (192.168.2.7) to Packet – 4(192.168.3.7)

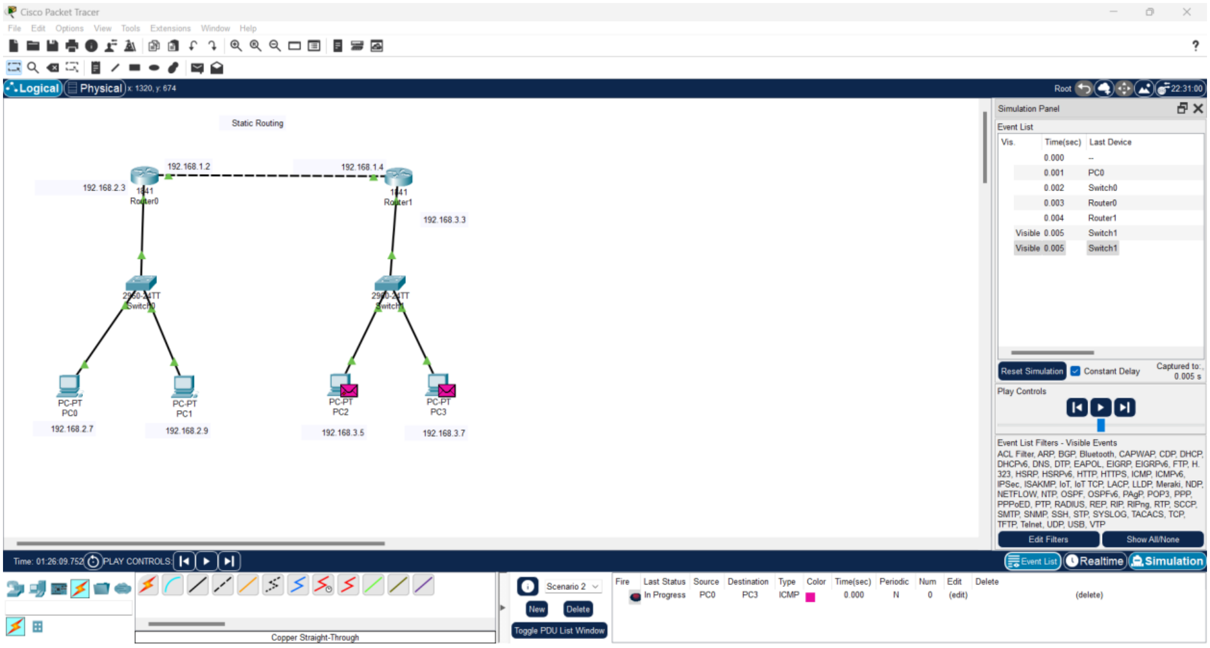

Cisco Packet Tracer
Static Routing
192.168.1.2
192.168.1.4
192.168.2.3
192.168.3.3
Router0
Router1
PC0
PC1
PC2
PC3
192.168.2.7
192.168.2.9
192.168.3.5
192.168.3.7
Simulation Panel
Event List
Reset Simulation
Constant Delay
Play Controls
Copper Straight-Through
Toggle PDU List Window
Realtime
Simulation


Cisco Packet Tracer
Static Routing
192.168.1.2
192.168.1.4
192.168.2.3
192.168.3.3
Router0
Router1
PC0
PC1
PC2
PC3
192.168.2.7
192.168.2.9
192.168.3.5
192.168.3.7
Simulation Panel
Event List
Reset Simulation
Constant Delay
Play Controls
Copper Straight-Through
Toggle PDU List Window
Realtime
Simulation


Cisco Packet Tracer
Static Routing
192.168.1.2
192.168.1.4
192.168.2.3
192.168.3.3
Router0
Router1
PC0
PC1
PC2
PC3
192.168.2.7
192.168.2.9
192.168.3.5
192.168.3.7
Simulation Panel
Event List
Reset Simulation
Constant Delay
Play Controls
Copper Straight-Through
Toggle PDU List Window
Realtime
Simulation


Cisco Packet Tracer
Static Routing
192.168.1.2
192.168.1.4
192.168.2.3
192.168.3.3
1941 Router0
1941 Router1
2960-24TT Switch0
2960-24TT Switch1
PC-PT PC0
PC-PT PC1
PC-PT PC2
PC-PT PC3
192.168.2.7
192.168.2.9
192.168.3.5
192.168.3.7
Simulation Panel
Event List
Time: 01:26:09.750
PLAY CONTROLS
Scenario 2
Toggle PDU List Window
Copper Straight-Through

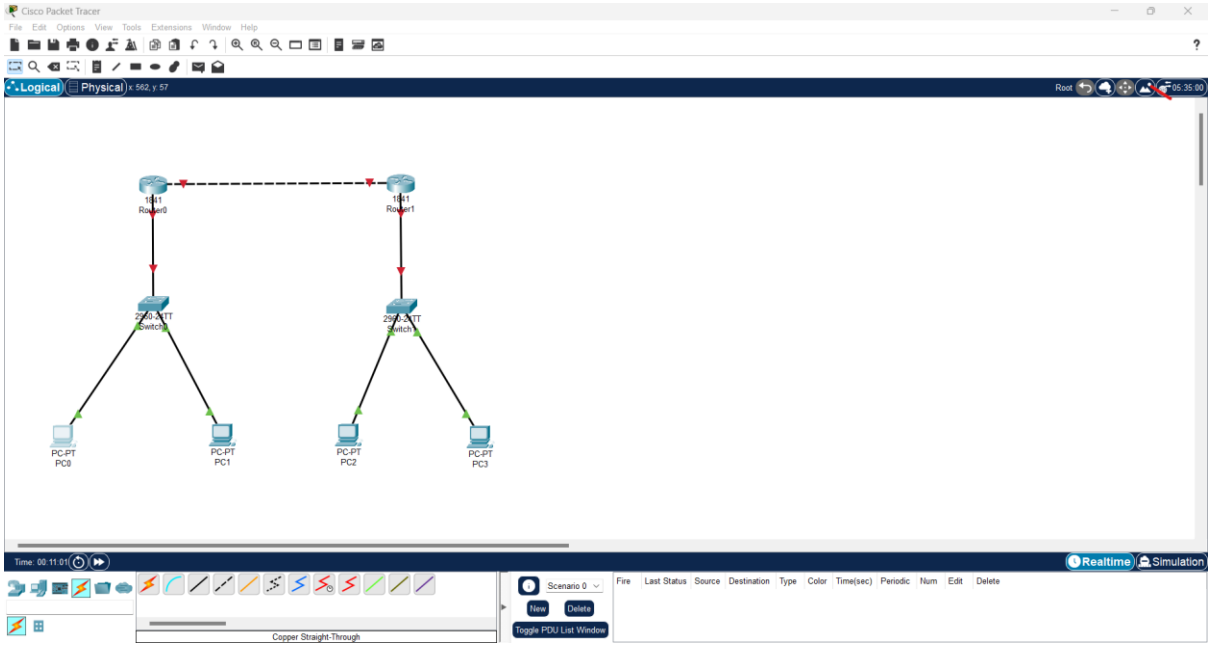

Cisco Packet Tracer
Static Routing
192.168.1.2
192.168.1.4
192.168.2.3
192.168.3.3
1941 Router0
1941 Router1
PC-PT PC0
PC-PT PC1
PC-PT PC2
PC-PT PC3
192.168.2.7
192.168.2.9
192.168.3.5
192.168.3.7
Simulation Panel
Event List
Time: 01:26:09.751
PLAY CONTROLS
Scenario 2
Toggle PDU List Window
Copper Straight-Through


Cisco Packet Tracer
Static Routing
192.168.1.2
192.168.1.4
192.168.2.3
192.168.3.3
1941 Router0
1941 Router1
PC-PT PC0
PC-PT PC1
PC-PT PC2
PC-PT PC3
192.168.2.7
192.168.2.9
192.168.3.5
192.168.3.7
Simulation Panel
Event List
Time: 01:26:09.752
PLAY CONTROLS
Scenario 2
Toggle PDU List Window
Copper Straight-Through

TASK – 3: Performing Dynamic Routing

Packet Tracer Used: Cisco Packet Tracer (CPT)

Step-1: Creating a Routing Network



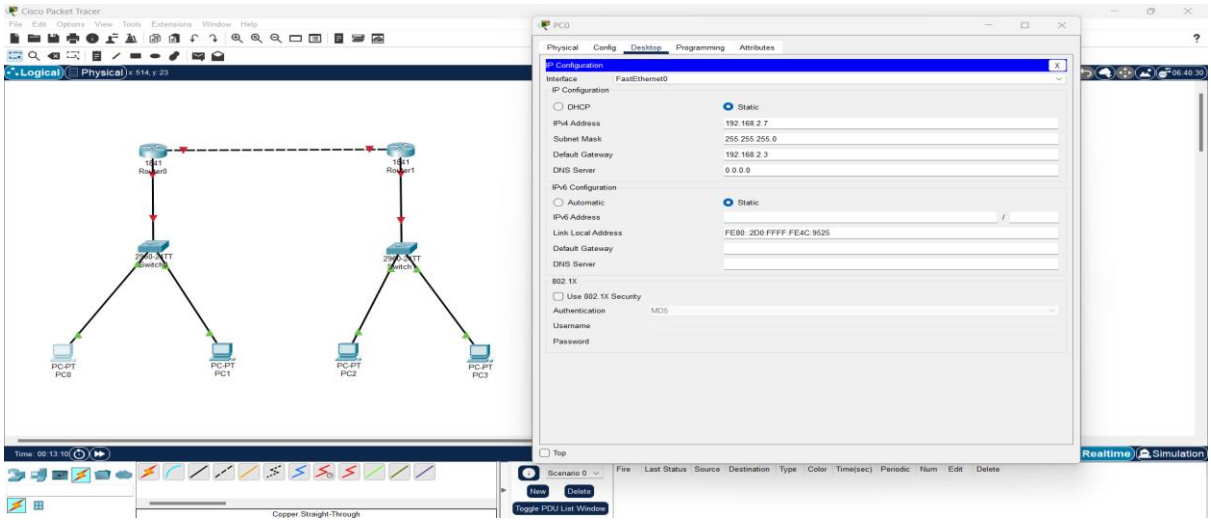
2 Routers

2 Switch's

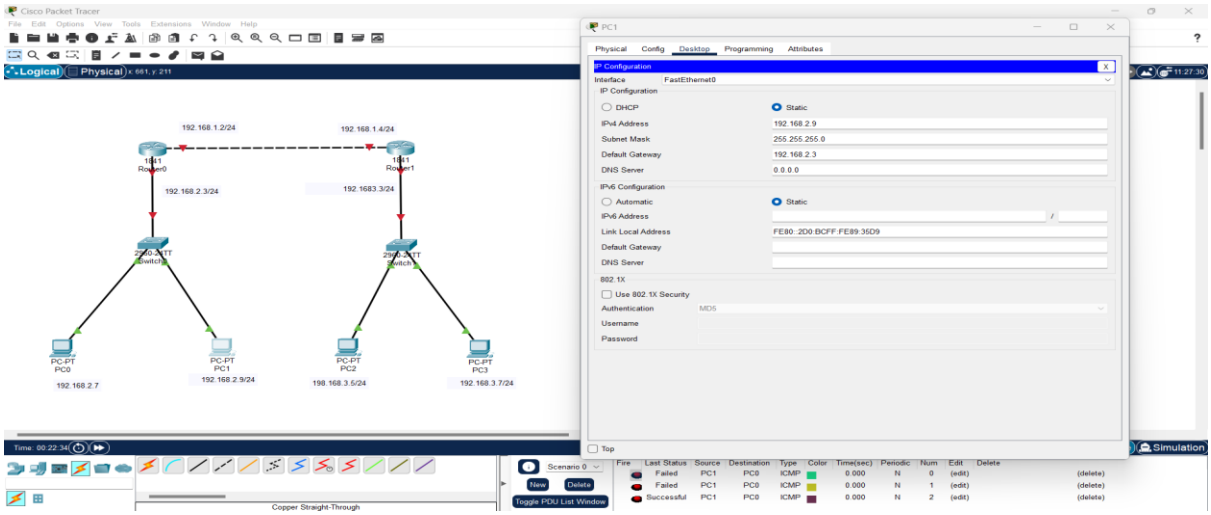
4 PC's

Step-2: IP Configuration

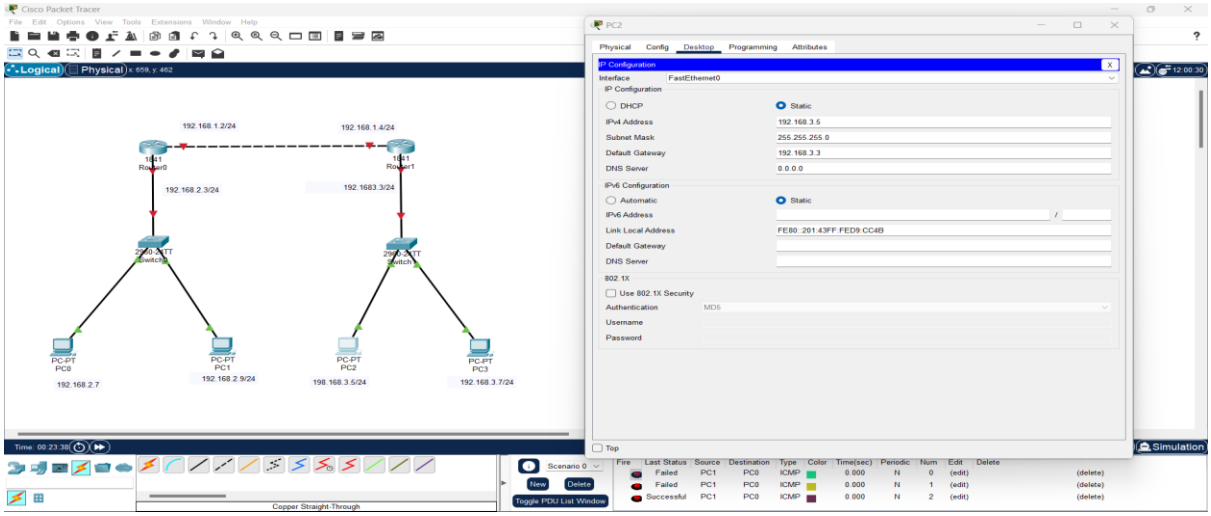
1st PC's:



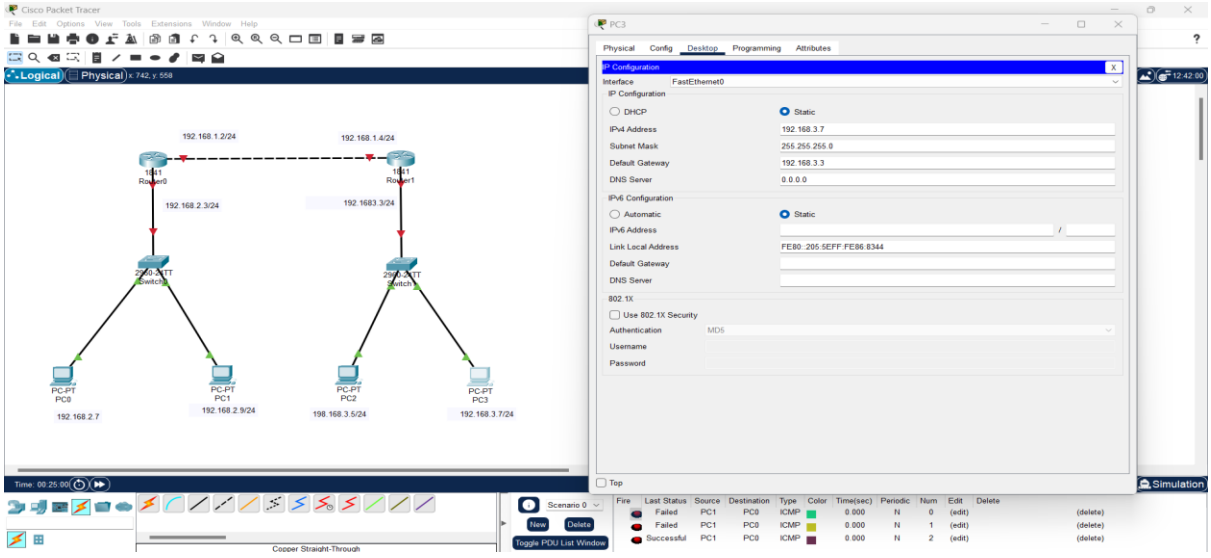
2nd PC's:



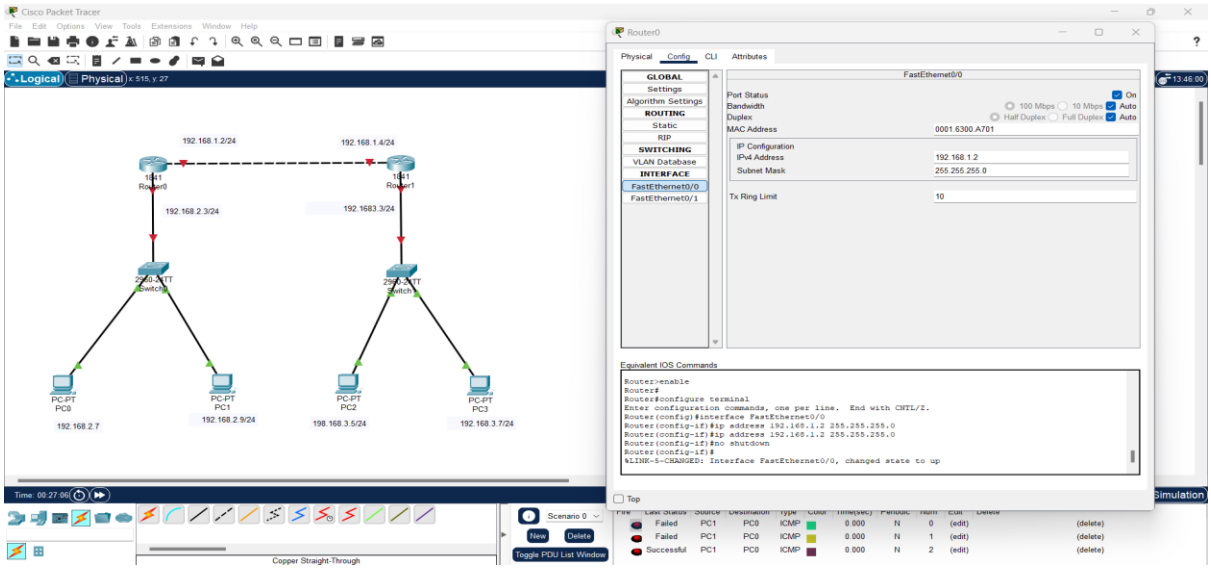
3rd PC's:

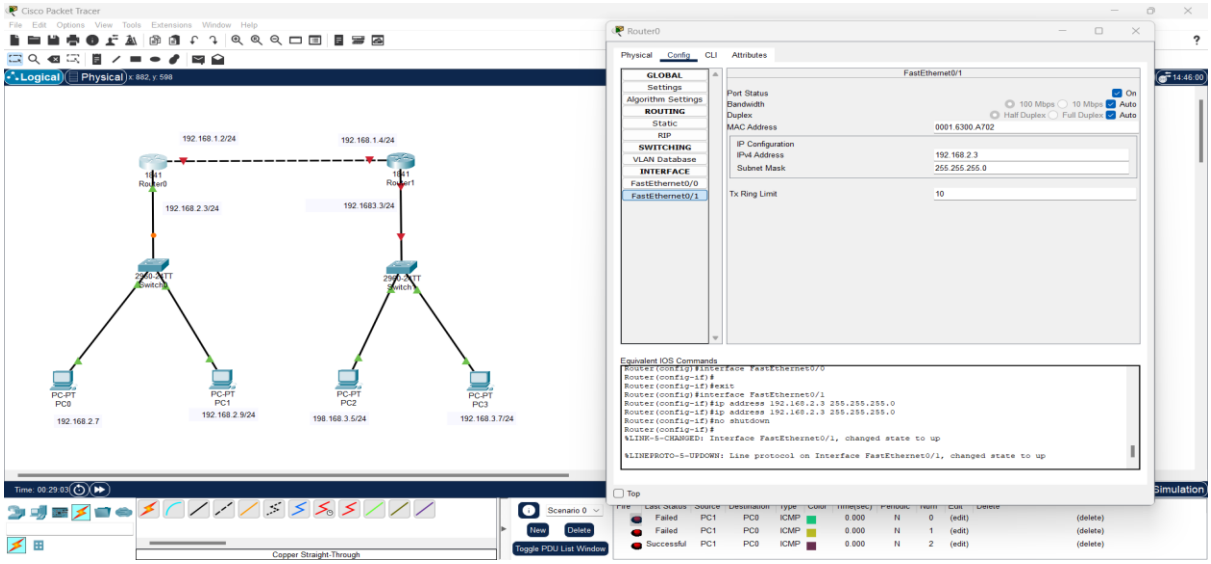


4th PC's:

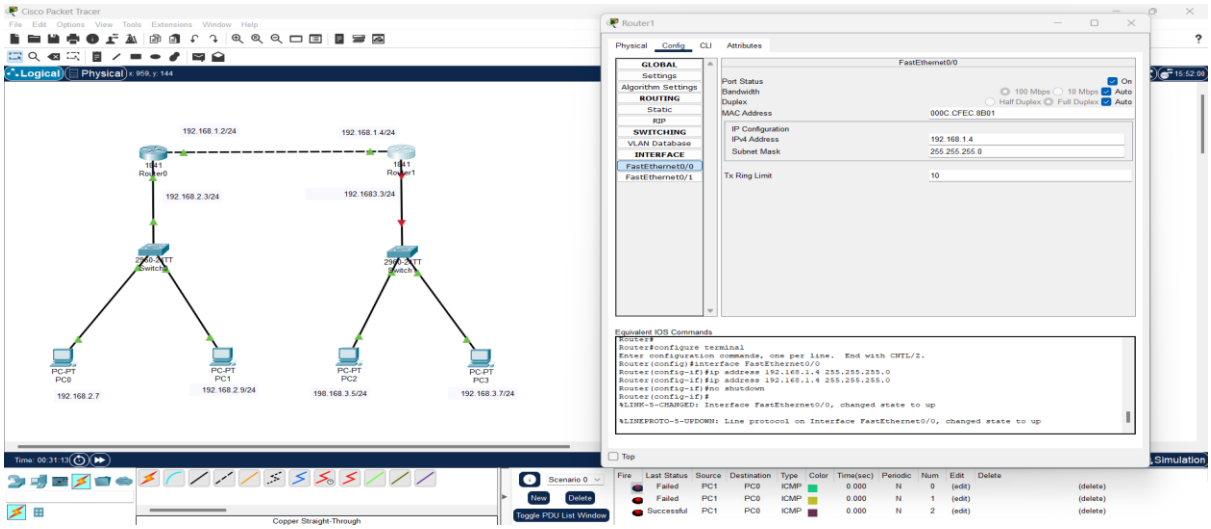


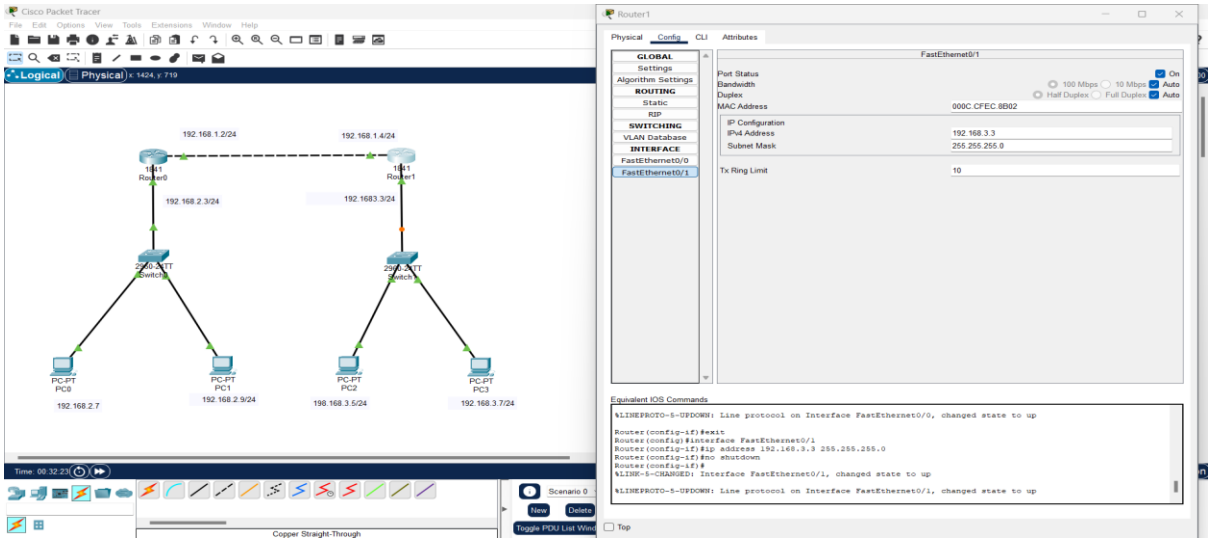
1st Router:

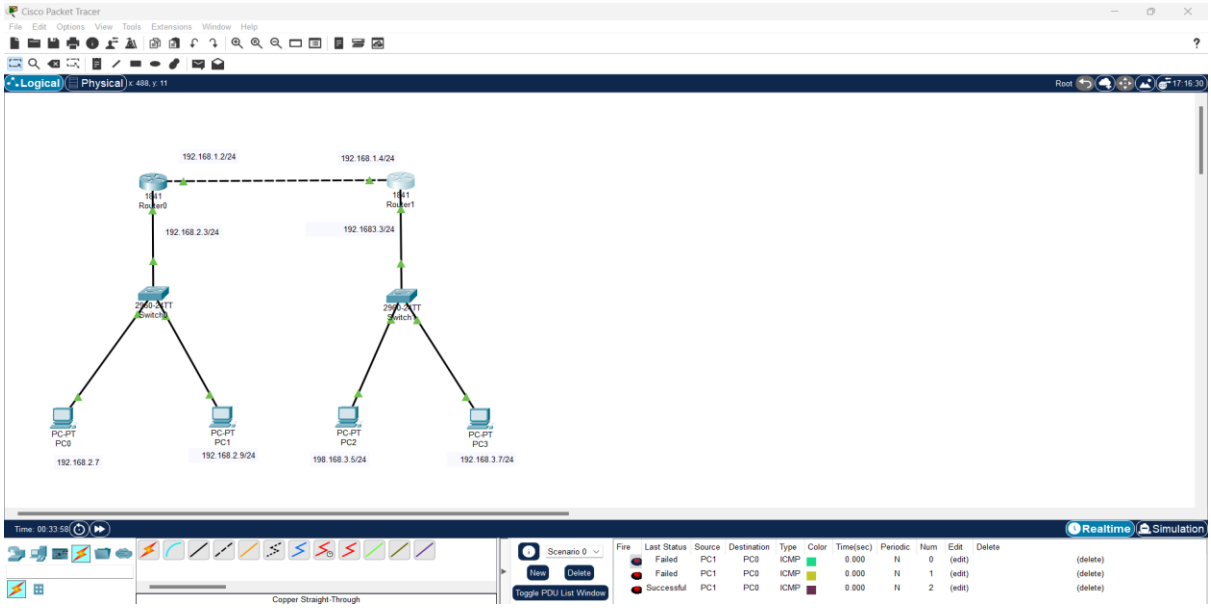




2nd Router:







Before Configuring Dynamic Routing:

Realtime Simulation

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Successful	PC0	PC1	ICMP		0.000	N	0	(edit)	(delete)
	Successful	PC1	PC0	ICMP		0.000	N	1	(edit)	(delete)
	Successful	PC2	PC3	ICMP		0.000	N	2	(edit)	(delete)

Realtime Simulation

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Successful	PC3	PC2	ICMP		0.000	N	3	(edit)	(delete)
	Failed	PC0	PC2	ICMP		0.000	N	4	(edit)	(delete)
	Failed	PC0	PC3	ICMP		0.000	N	5	(edit)	(delete)

Realtime Simulation

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Failed	PC1	PC2	ICMP		0.000	N	6	(edit)	(delete)
	Failed	PC1	PC3	ICMP		0.000	N	7	(edit)	(delete)
	Failed	PC2	PC0	ICMP		0.000	N	8	(edit)	(delete)

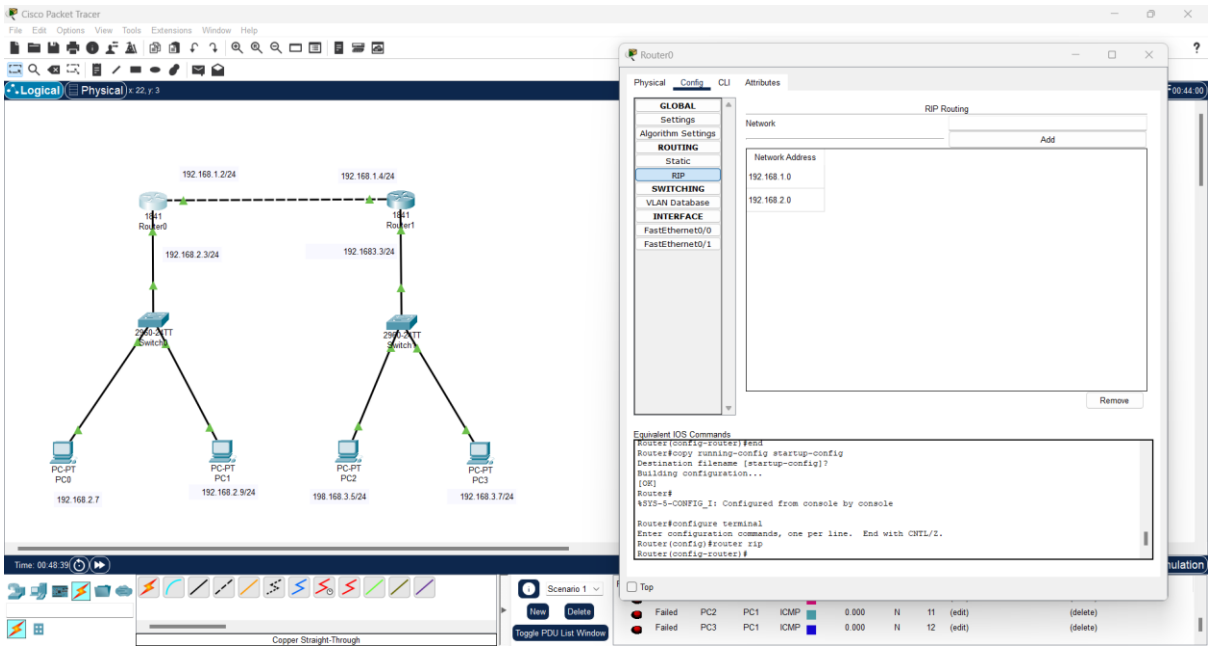
Realtime Simulation

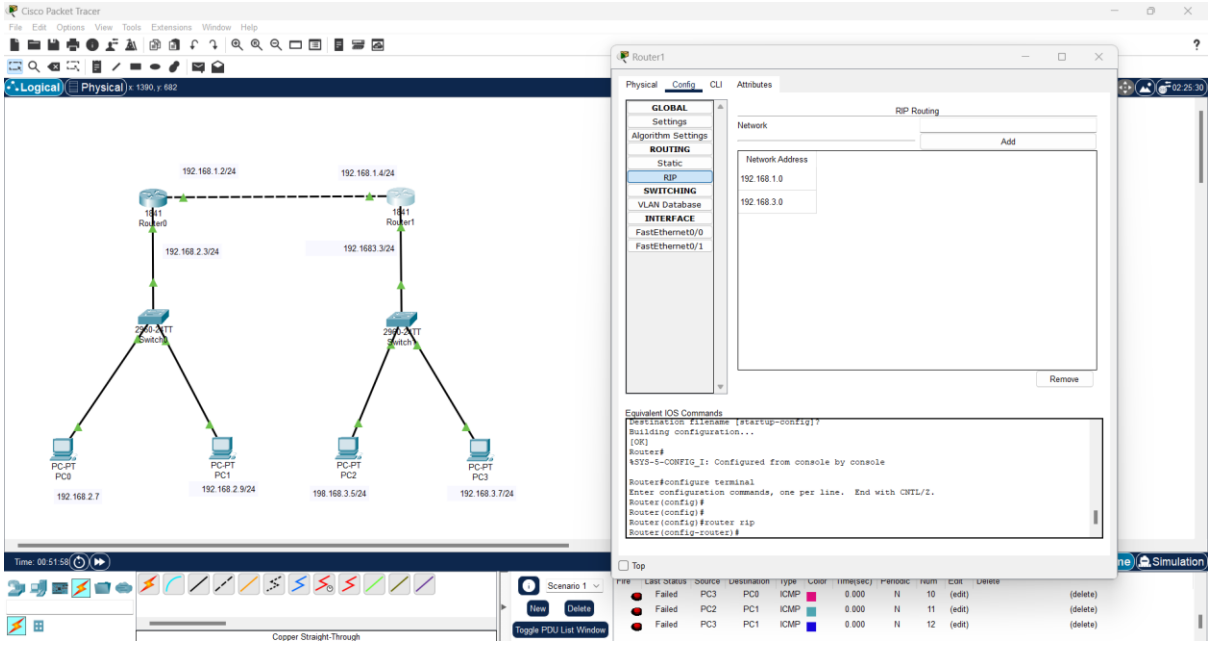
Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Failed	PC2	PC1	ICMP		0.000	N	9	(edit)	(delete)
	Failed	PC3	PC0	ICMP		0.000	N	10	(edit)	(delete)
	Failed	PC2	PC1	ICMP		0.000	N	11	(edit)	(delete)

Realtime Simulation

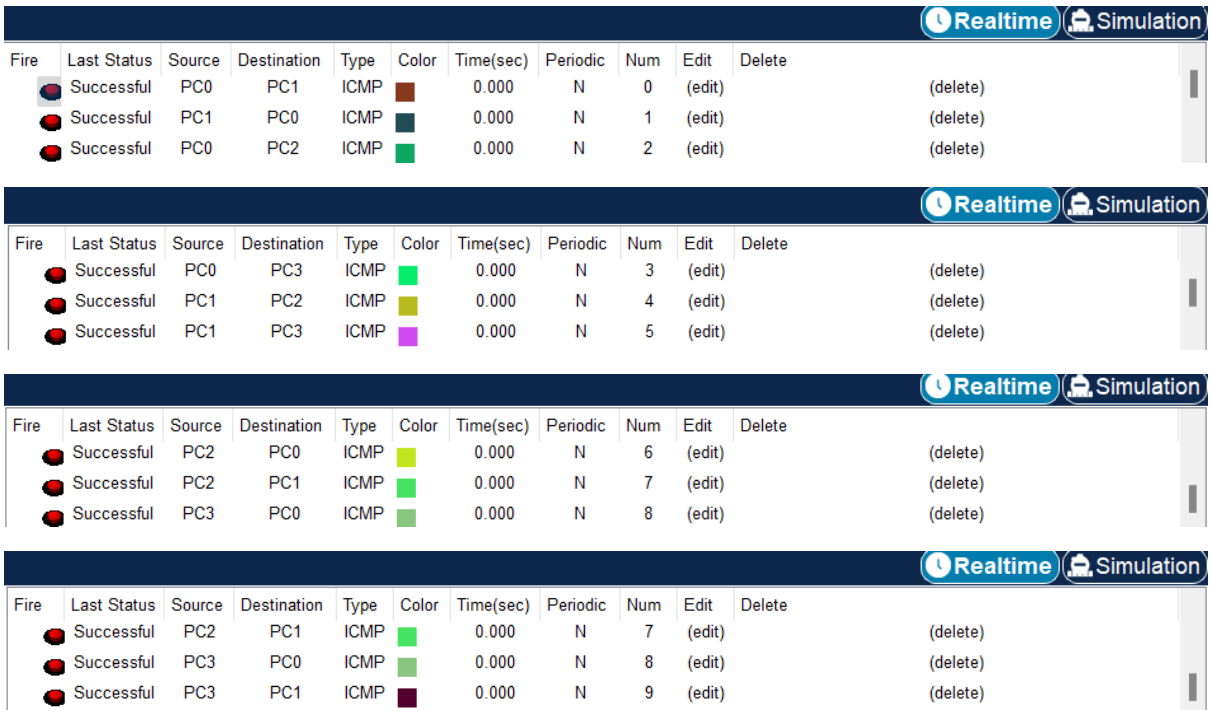
Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Failed	PC3	PC0	ICMP		0.000	N	10	(edit)	(delete)
	Failed	PC2	PC1	ICMP		0.000	N	11	(edit)	(delete)
	Failed	PC3	PC1	ICMP		0.000	N	12	(edit)	(delete)

Step-3: Configuring Dynamic Routing





After Configuring Dynamic Routing:



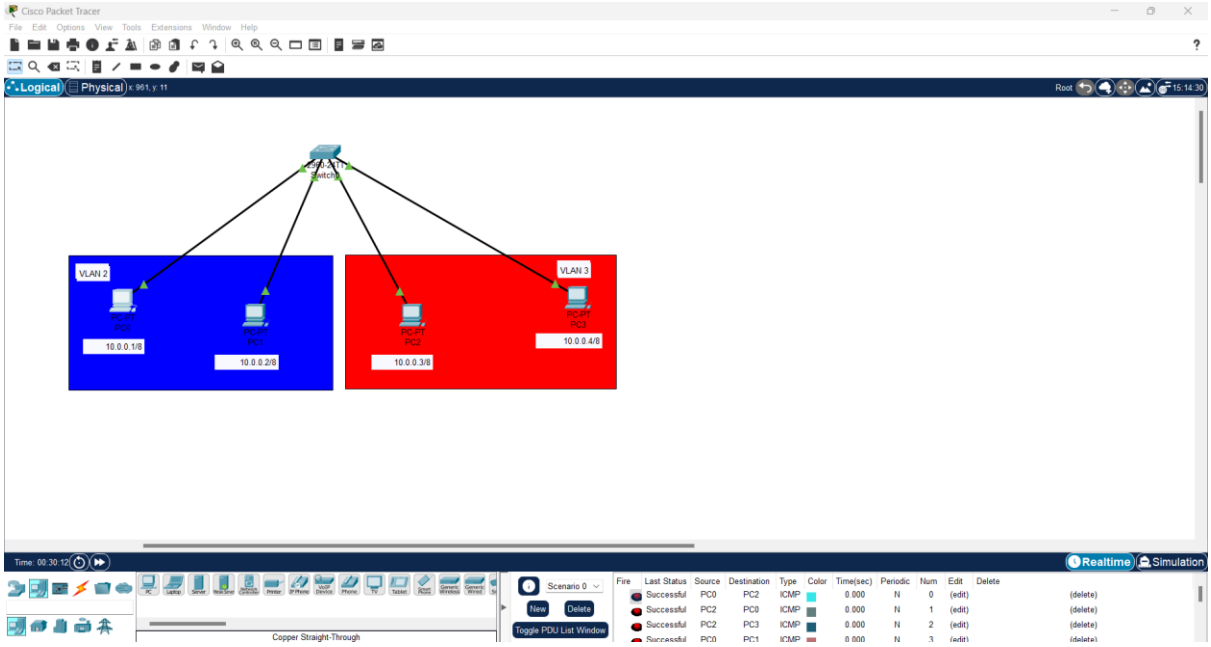
Realtime Simulation

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Successful	PC0	PC1	ICMP		0.000	N	0	(edit)	(delete)
	Successful	PC1	PC0	ICMP		0.000	N	1	(edit)	(delete)
	Successful	PC0	PC2	ICMP		0.000	N	2	(edit)	(delete)
	Successful	PC0	PC3	ICMP		0.000	N	3	(edit)	(delete)
	Successful	PC1	PC2	ICMP		0.000	N	4	(edit)	(delete)
	Successful	PC1	PC3	ICMP		0.000	N	5	(edit)	(delete)
	Successful	PC2	PC0	ICMP		0.000	N	6	(edit)	(delete)
	Successful	PC2	PC1	ICMP		0.000	N	7	(edit)	(delete)
	Successful	PC3	PC0	ICMP		0.000	N	8	(edit)	(delete)
	Successful	PC2	PC1	ICMP		0.000	N	7	(edit)	(delete)
	Successful	PC3	PC0	ICMP		0.000	N	8	(edit)	(delete)
	Successful	PC3	PC1	ICMP		0.000	N	9	(edit)	(delete)

TASK – 4: VLAN Configuration

Packet Tracer Used: Cisco Packet Tracer (CPT)

Step-1: Creating a VLAN Network



Before VLAN Configuration:

Realtime Simulation

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Successful	PC0	PC1	ICMP		0.000	N	0	(edit)	(delete)
	Successful	PC0	PC2	ICMP		0.000	N	1	(edit)	(delete)
	Successful	PC0	PC3	ICMP		0.000	N	2	(edit)	(delete)

Realtime Simulation

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Successful	PC1	PC2	ICMP		0.000	N	3	(edit)	(delete)
	Successful	PC1	PC3	ICMP		0.000	N	4	(edit)	(delete)
	Successful	PC1	PC0	ICMP		0.000	N	5	(edit)	(delete)

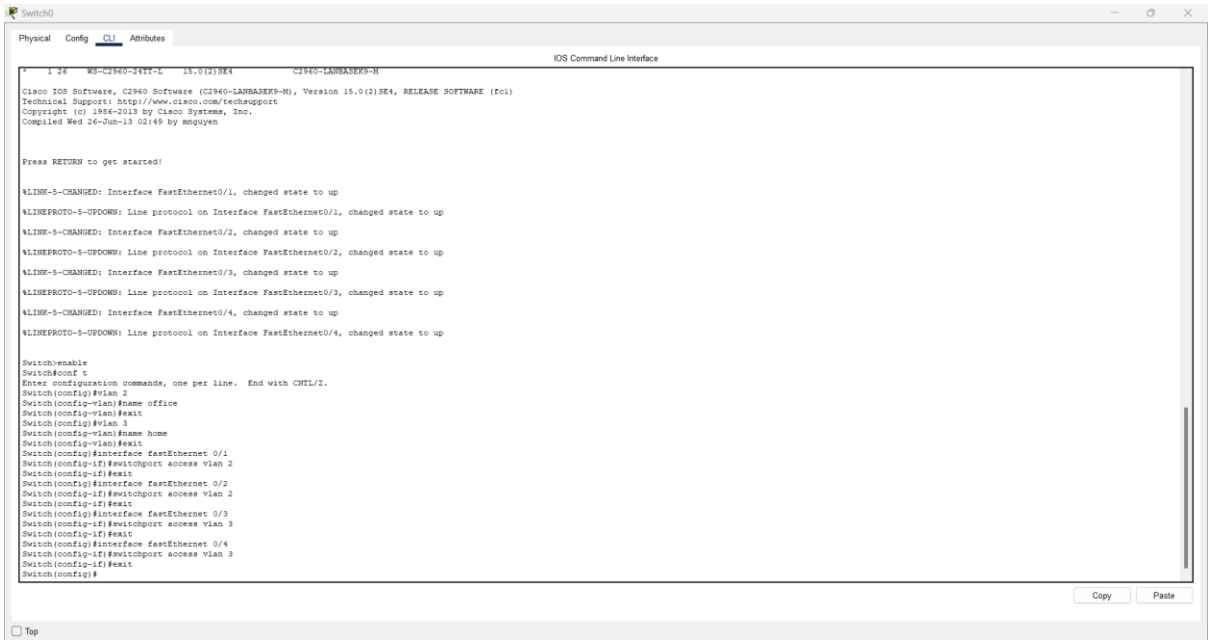
Realtime Simulation

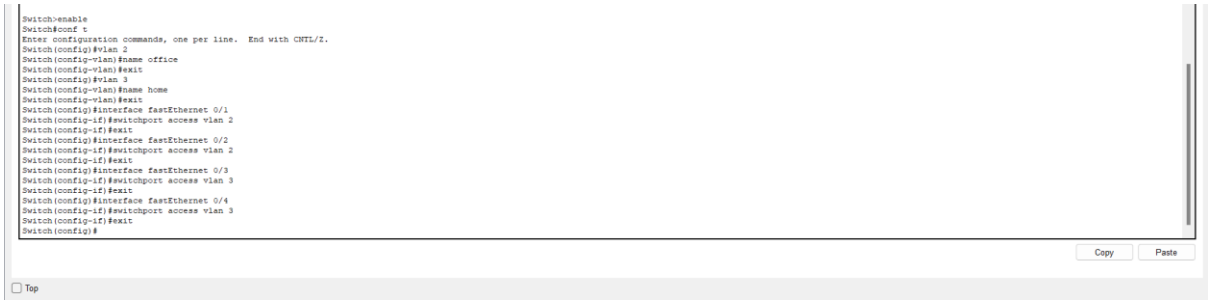
Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Successful	PC2	PC3	ICMP		0.000	N	6	(edit)	(delete)
	Successful	PC2	PC1	ICMP		0.000	N	7	(edit)	(delete)
	Successful	PC2	PC0	ICMP		0.000	N	8	(edit)	(delete)

Realtime Simulation

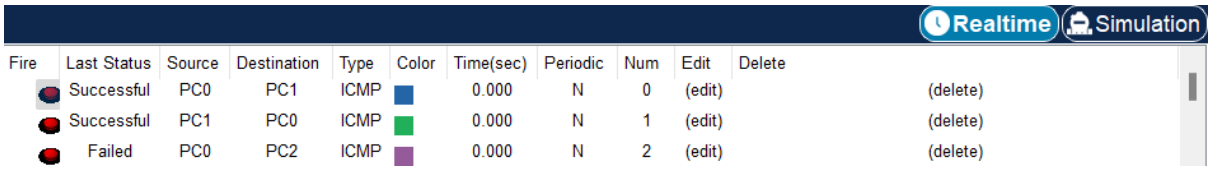
Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Successful	PC3	PC2	ICMP		0.000	N	9	(edit)	(delete)
	Successful	PC3	PC1	ICMP		0.000	N	10	(edit)	(delete)
	Successful	PC3	PC0	ICMP		0.000	N	11	(edit)	(delete)

Step-2: Configuring VLAN Network:

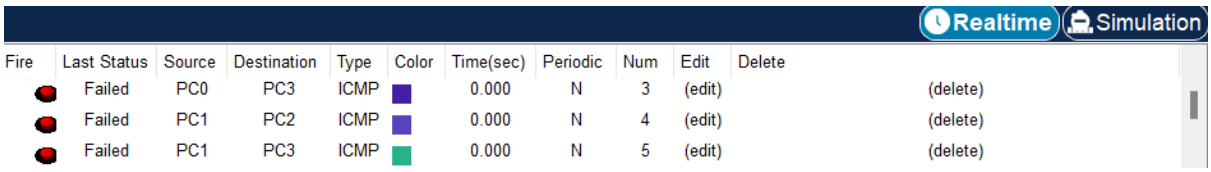




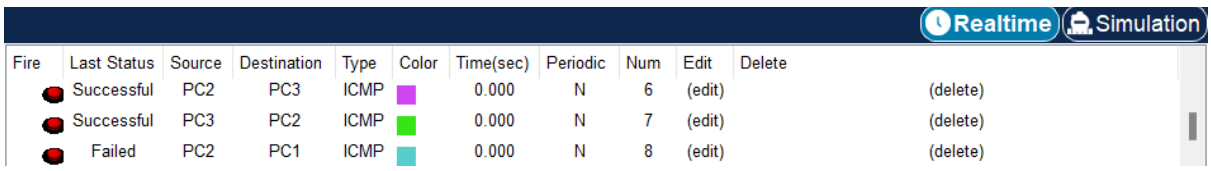
After Configuring VLAN:



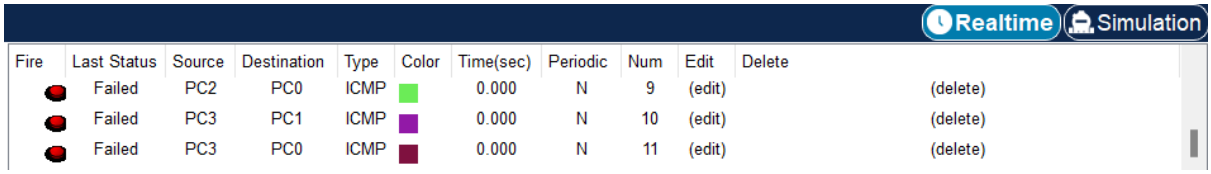
Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Successful	PC0	PC1	ICMP		0.000	N	0	(edit)	(delete)
	Successful	PC1	PC0	ICMP		0.000	N	1	(edit)	(delete)
	Failed	PC0	PC2	ICMP		0.000	N	2	(edit)	(delete)



Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Failed	PC0	PC3	ICMP		0.000	N	3	(edit)	(delete)
	Failed	PC1	PC2	ICMP		0.000	N	4	(edit)	(delete)
	Failed	PC1	PC3	ICMP		0.000	N	5	(edit)	(delete)



Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Successful	PC2	PC3	ICMP		0.000	N	6	(edit)	(delete)
	Successful	PC3	PC2	ICMP		0.000	N	7	(edit)	(delete)
	Failed	PC2	PC1	ICMP		0.000	N	8	(edit)	(delete)


Realtime
Simulation
Fire
Last Status
Source
Destination
Type
Color
Time(sec)
Periodic
Num
Edit
Delete
Failed PC2 PC0 ICMP 0.000 N 9 (edit) (delete)
Failed PC3 PC1 ICMP 0.000 N 10 (edit) (delete)
Failed PC3 PC0 ICMP 0.000 N 11 (edit) (delete)

LAB – 3

(Algorithms)

Need to Implement the Following algorithms (in any of your preferred programming language) :

- Implement flow control so that a fast sender will not overrun a slow receiver's buffer.
- Implement RED algorithm DEC Bit scheme in TCP.
- Implement the Drop Tail Buffer Management Policies.
- Implement the Drop Front Buffer Management Policies.
- Implement the Random Drop Buffer Management Policies.
- Implement the Early Random Drop Buffer Management Policies.

Title: Implement flow control so that a fast sender will not overrun a slow receiver's buffer.

To prevent a fast sender from overwhelming a slow receiver, flow control is used. In this system, the receiver tells the sender when it's ready for more data. This way, the sender only sends data at a speed the receiver can handle, avoiding data overload. It's like a conversation where one person speaks only when the other is ready to listen.

Server Side Code:

```
# Implement flow control so that a fast sender will not overrun a slow
receiver's buffer.
import socket
import time

def process_data(data):
    """Simulate processing data."""
    time.sleep(1)  # Simulate a delay in processing
    print(f"Processed data: {data}")

def server():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind(('localhost', 12345))
        s.listen()
        conn, addr = s.accept()
        with conn:
            print('Connected by', addr)
            while True:
                data = conn.recv(1024)
```

```
                if not data:
                    break
                process_data(data)
                conn.sendall(b'ACK')  # Send acknowledgment

server()
```

Client Side Code:

```
# Implement flow control so that a fast sender will not overrun a slow
receiver's buffer.
import socket

def client():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect(('localhost', 12345))
        for i in range(10):
            message = f"Message {i}".encode()
            s.sendall(message)
            ack = s.recv(1024)
            print(f"Received acknowledgment: {ack}")

client()
```

Output:

```
Microsoft Windows [Version 10.0.22631.2506]
(c) Microsoft Corporation. All rights reserved.

E:\Collage\BTECH-SEMESTER-7\Online Internet Protocols and Networking\Lab -
3>python Question-1(Server).py
Connected by ('127.0.0.1', 52408)
Processed data: b'Message 0'
Processed data: b'Message 1'
Processed data: b'Message 2'
Processed data: b'Message 3'
Processed data: b'Message 4'
Processed data: b'Message 5'
Processed data: b'Message 6'
Processed data: b'Message 7'
Processed data: b'Message 8'
Processed data: b'Message 9'

E:\Collage\BTECH-SEMESTER-7\Online Internet Protocols and Networking\Lab -
3>
```

```
Microsoft Windows [Version 10.0.22631.2506]
(c) Microsoft Corporation. All rights reserved.

E:\Collage\BTECH-SEMESTER-7\Online Internet Protocols and Networking\Lab
- 3>python Question-2(Client).py
Received acknowledgment: b'ACK'
Received acknowledgment: b'ACK'
Received acknowledgment: b'ACK'
Received acknowledgment: b'ACK'
Received acknowledgment: b'ACK'
Received acknowledgment: b'ACK'
Received acknowledgment: b'ACK'
Received acknowledgment: b'ACK'
Received acknowledgment: b'ACK'
Received acknowledgment: b'ACK'

E:\Collage\BTECH-SEMESTER-7\Online Internet Protocols and Networking\Lab
- 3>
```

Data is being sent from server only after getting acknowledgment from client. No matter how fast the server might be it cannot send the data before receiving acknowledgment from the client.

Title: Implement RED algorithm DEC Bit scheme in TCP.

The Random Early Detection (RED) algorithm in TCP, combined with the DEC Bit scheme, is a strategy for managing network congestion. RED works by monitoring the average queue length at network routers and probabilistically dropping incoming packets before the queue becomes full. This early packet dropping signals to the sender to reduce its transmission rate, helping to prevent network congestion. The DEC Bit (Explicit Congestion Notification Echo) is a part of this mechanism, where routers mark packets instead of dropping them when the queue starts to fill. The sender, upon receiving these marked packets, understands that there's congestion ahead and slows down its sending rate. This approach allows for more efficient and smoother control of network traffic, reducing the likelihood of packet loss and improving overall network performance.

Client Side Code:

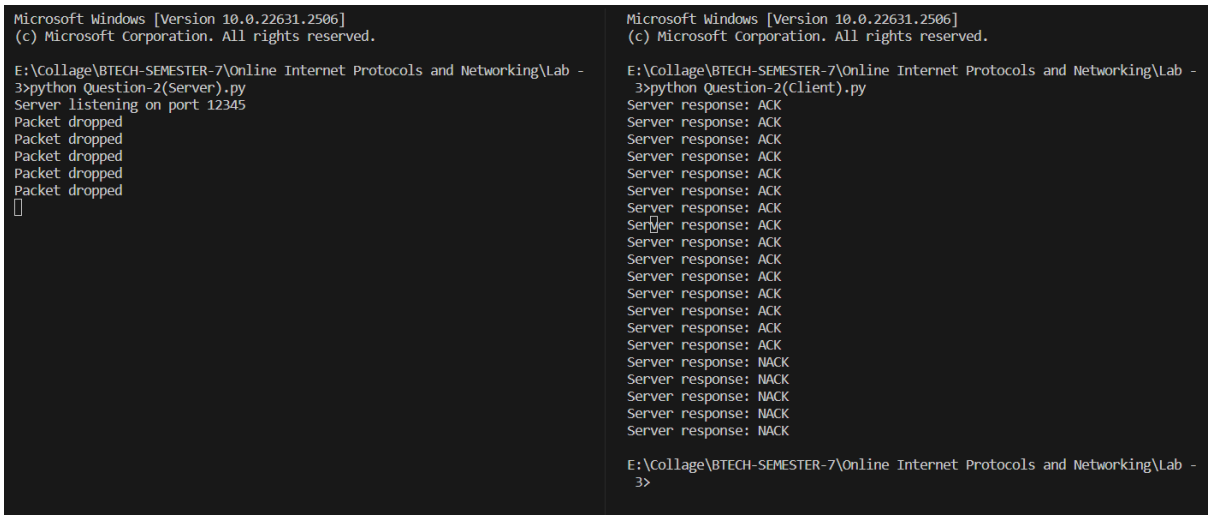
```
# Implement RED algorithm DEC Bit scheme in TCP.
import socket
import time

def client(server_ip, port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((server_ip, port))
        for i in range(20):
            s.sendall(f"Message {i}".encode())
            response = s.recv(1024)
            print(f"Server response: {response.decode()}")
            time.sleep(1)  # Simulate delay

client('localhost', 12345)
```

Server Side Code:

```
# Implement RED algorithm DEC Bit scheme in TCP.
import socket
import threading
import random

class RedServer:
    def __init__(self, port, min_threshold, max_threshold, max_buffer_size):
        self.min_threshold = min_threshold
        self.max_threshold = max_threshold
        self.max_buffer_size = max_buffer_size
        self.buffer = []
        self.lock = threading.Lock()
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind(('', port))
        self.server_socket.listen(5)
        print(f"Server listening on port {port}")

    def start(self):
        while True:
            client_socket, addr = self.server_socket.accept()
            threading.Thread(target=self.handle_client,
args=(client_socket,)).start()

    def handle_client(self, client_socket):
        while True:
            data = client_socket.recv(1024)
            if not data:
                break

            with self.lock:
                if len(self.buffer) < self.max_buffer_size:
                    self.buffer.append(data)
                    client_socket.send(b'ACK')
                else:
                    # Simulate RED algorithm
                    drop_probability = self.calculate_drop_probability()
                    if random.random() < drop_probability:
                        print("Packet dropped")
                        client_socket.send(b'NACK')
                    else:
                        self.buffer.append(data)
                        client_socket.send(b'ACK')

    def calculate_drop_probability(self):
        avg_queue_size = len(self.buffer)
        if avg_queue_size > self.max_threshold:
```

```
            return 1.0  # Always drop
        elif avg_queue_size > self.min_threshold:
            return (avg_queue_size - self.min_threshold) / (self.max_threshold
- self.min_threshold)
        else:
            return 0.0

server = RedServer(12345, 5, 10, 15)
server.start()
```

Output:



Client is sending the data to the server and for every packet sent there is server response which is “ACK” – acknowledgment and there is a queue within the buffer when the buffer is filled to its capacity the packets are being dropped and the server sends a response which is “NACK” – Negative acknowledgment which means server is telling that the buffer is full and client needs to adjust it transmission speed.

Title: Implement the Drop Tail Buffer Management Policies.

Drop Tail is a simple buffer management policy where incoming packets are dropped when the buffer capacity is reached. There's no prioritization; if the buffer is full when a new packet arrives, the packet is simply dropped.

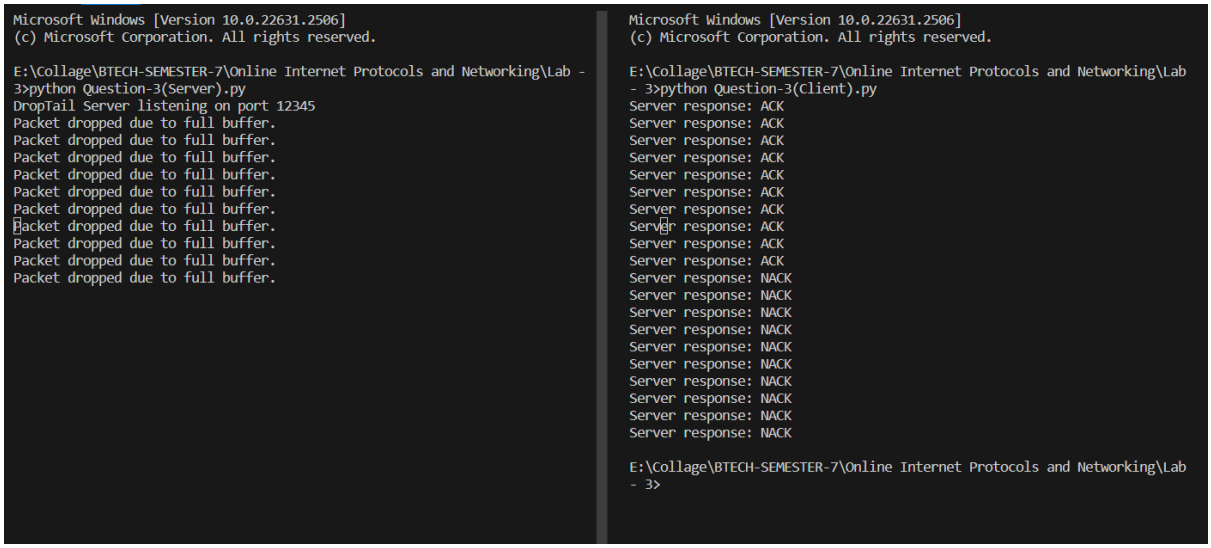
Server Side Code:

```
# Implement the Drop Tail Buffer Management Policies.
import socket
import threading

class DropTailServer:
    def __init__(self, port, buffer_size):
        self.buffer_size = buffer_size
        self.buffer = []
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind(('', port))
        self.server_socket.listen(5)
        print(f"DropTail Server listening on port {port}")

    def start(self):
        while True:
            client_socket, addr = self.server_socket.accept()
            threading.Thread(target=self.handle_client,
args=(client_socket,)).start()

    def handle_client(self, client_socket):
        while True:
            data = client_socket.recv(1024)
            if not data:
                break

            if len(self.buffer) < self.buffer_size:
                self.buffer.append(data)
                client_socket.sendall(b'ACK')
            else:
                client_socket.sendall(b'NACK')
                print("Packet dropped due to full buffer.")

server = DropTailServer(12345, 10)
server.start()
```

Client Side Code:

```python
# Implement the Drop Tail Buffer Management Policies.
import socket
import time

def client(server_ip, port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((server_ip, port))
        for i in range(20):  # Send 20 packets as an example
            s.sendall(f"Packet {i}".encode())
            response = s.recv(1024)
            print(f"Server response: {response.decode()}")
            time.sleep(0.5)  # Wait for half a second between sends

client('localhost', 12345)
```

Output:



The Packets are dropped if the buffer is Full.

Title: Implement the Drop Front Buffer Management Policies.

The Drop Front buffer management policy is a simple queue management where the oldest packets are dropped when the buffer is full to make room for newer packets. This is opposite to the Drop Tail policy, which drops incoming new packets when the buffer is full.

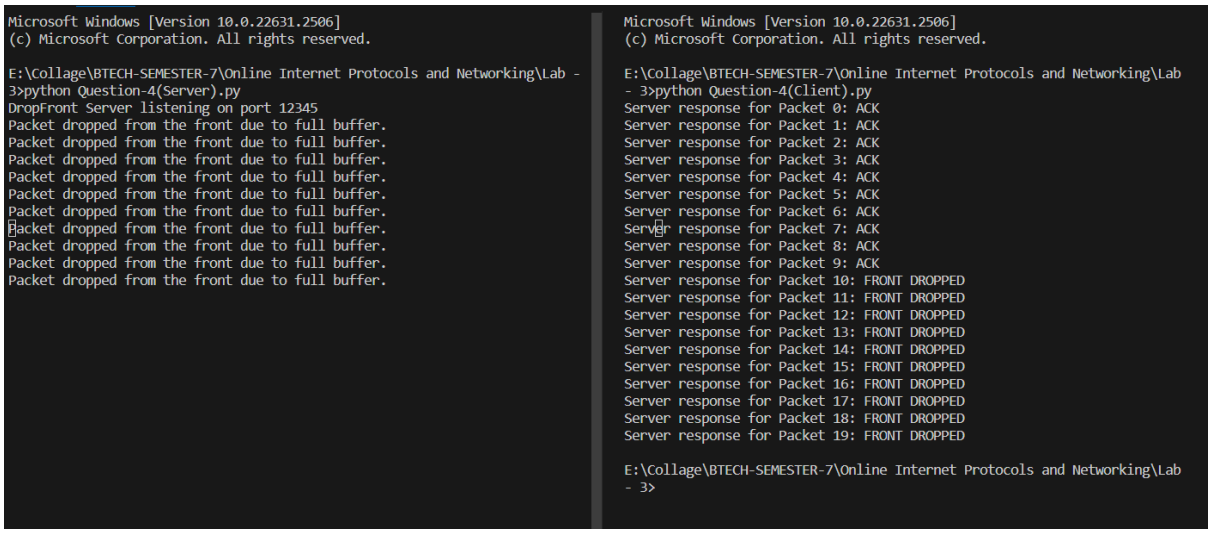
Server Side Code:

```
# Implement the Drop Front Buffer Management Policies.
import socket
import threading

class DropFrontServer:
    def __init__(self, port, buffer_size):
        self.buffer_size = buffer_size
        self.buffer = []
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind(('', port))
        self.server_socket.listen(5)
        print(f"DropFront Server listening on port {port}")

    def start(self):
        while True:
            client_socket, addr = self.server_socket.accept()
            threading.Thread(target=self.handle_client,
args=(client_socket,)).start()

    def handle_client(self, client_socket):
        while True:
            data = client_socket.recv(1024)
            if not data:
                break

            if len(self.buffer) < self.buffer_size:
                self.buffer.append(data)
                client_socket.sendall(b'ACK')
            else:
                self.buffer.pop(0)
                self.buffer.append(data)
                client_socket.sendall(b'FRONT DROPPED')
                print("Packet dropped from the front due to full buffer.")

server = DropFrontServer(12345, 10)
server.start()
```

Client Side Code:

```
# Implement the Drop Front Buffer Management Policies.
import socket
import time

def client(server_ip, port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((server_ip, port))
        for i in range(20):  # Send 20 packets as an example
            packet = f"Packet {i}".encode()
            s.sendall(packet)
            response = s.recv(1024)
            print(f"Server response for {packet.decode()}:
{response.decode()}")
            time.sleep(0.5)  # Wait for half a second between sends

client('localhost', 12345)
```

Output:



The Oldest Packets are dropped if the buffer is full.

Title: Implement the Random Drop Buffer Management Policies.

In the Random Drop buffer management policy, when the buffer is full, a packet is randomly chosen and dropped to make room for the new packet.

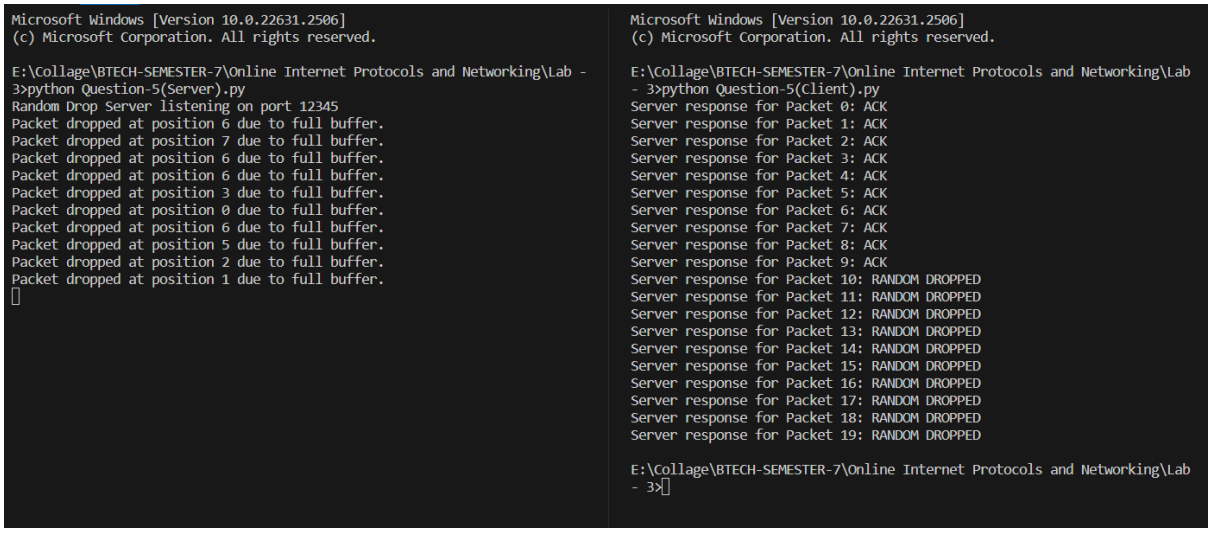
Server Side Code:

```
# Implement the Random Drop Buffer Management Policies.
import socket
import threading
import random

class RandomDropServer:
    def __init__(self, port, buffer_size):
        self.buffer_size = buffer_size
        self.buffer = []
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind(('', port))
        self.server_socket.listen(5)
        print(f"Random Drop Server listening on port {port}")

    def start(self):
        while True:
            client_socket, addr = self.server_socket.accept()
            threading.Thread(target=self.handle_client,
args=(client_socket,)).start()

    def handle_client(self, client_socket):
        while True:
            data = client_socket.recv(1024)
            if not data:
                break

            if len(self.buffer) < self.buffer_size:
                self.buffer.append(data)
                client_socket.sendall(b'ACK')
            else:
                drop_index = random.randint(0, self.buffer_size - 1)
                self.buffer.pop(drop_index)
                self.buffer.append(data)
                client_socket.sendall(b'RANDOM DROPPED')
                print(f"Packet dropped at position {drop_index} due to full
buffer.")
server = RandomDropServer(12345, 10)
server.start()
```

Client Side Code:

```
# Implement the Random Drop Buffer Management Policies.
import socket
import time

def client(server_ip, port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((server_ip, port))
        for i in range(20):  # Send 20 packets as an example
            packet = f"Packet {i}".encode()
            s.sendall(packet)
            response = s.recv(1024)
            print(f"Server response for {packet.decode()}:
{response.decode()}")
            time.sleep(0.5)  # Wait for half a second between sends

client('localhost', 12345)
```

Output:



Random Packets are dropped if the buffer is full.

Title: Implement the Early Random Drop Buffer Management Policies.

The Early Random Drop (ERD) buffer management policy is an extension of the Random Drop policy. It starts to randomly drop packets before the buffer is completely full, typically when the buffer occupancy reaches a predefined threshold. This is to prevent sudden surges of drops when the buffer is full and to give an early signal to the sender to slow down.

Server Side Code:

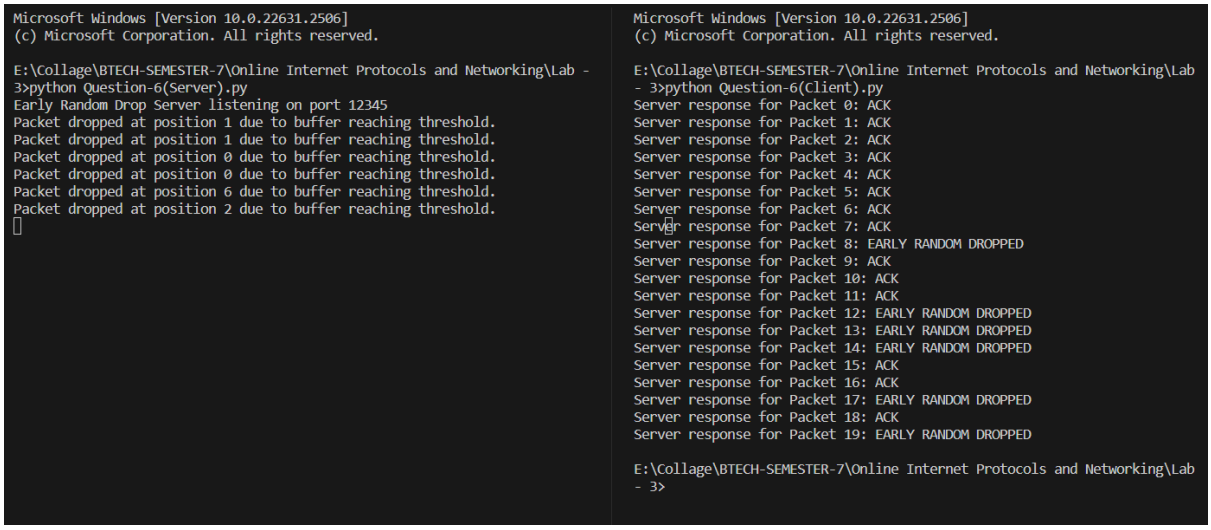
```
# Implement the Early Random Drop Buffer Management Policies.
import socket
import threading
import random

class EarlyRandomDropServer:
    def __init__(self, port, buffer_size, drop_threshold):
        self.buffer_size = buffer_size
        self.drop_threshold = drop_threshold
        self.buffer = []
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind(('', port))
        self.server_socket.listen(5)
        print(f"Early Random Drop Server listening on port {port}")

    def start(self):
        while True:
            client_socket, addr = self.server_socket.accept()
            threading.Thread(target=self.handle_client,
args=(client_socket,)).start()

    def handle_client(self, client_socket):
        while True:
            data = client_socket.recv(1024)
            if not data:
                break

            if len(self.buffer) < self.buffer_size:
                if len(self.buffer) > self.drop_threshold:
                    if random.random() < self.compute_drop_probability():
                        self.drop_packet(client_socket)
                        continue
```

```
                self.buffer.append(data)
                client_socket.sendall(b'ACK')
            else:
                self.drop_packet(client_socket)

    def compute_drop_probability(self):
        return (len(self.buffer) - self.drop_threshold) / (self.buffer_size -
self.drop_threshold)

    def drop_packet(self, client_socket):
        drop_index = random.randint(0, len(self.buffer) - 1)
        self.buffer.pop(drop_index)
        client_socket.sendall(b'EARLY RANDOM DROPPED')
        print(f"Packet dropped at position {drop_index} due to buffer reaching
threshold.")

server = EarlyRandomDropServer(12345, 10, 7)  # Example buffer size of 10 and
threshold of 7
server.start()
```

Client Side Code:

```
# Implement the Early Random Drop Buffer Management Policies.
import socket
import time

def client(server_ip, port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((server_ip, port))
        for i in range(20):  # Send 20 packets as an example
            packet = f"Packet {i}".encode()
            s.sendall(packet)
            response = s.recv(1024)
            print(f"Server response for {packet.decode()}:
{response.decode()}")
            time.sleep(0.5)  # Wait for half a second between sends

client('localhost', 12345)
```

Output:



```
Microsoft Windows [Version 10.0.22631.2506]
(c) Microsoft Corporation. All rights reserved.

E:\Collage\BTECH-SEMESTER-7\Online Internet Protocols and Networking\Lab - 3>python Question-6(Server).py
Early Random Drop Server listening on port 12345
Packet dropped at position 1 due to buffer reaching threshold.
Packet dropped at position 1 due to buffer reaching threshold.
Packet dropped at position 0 due to buffer reaching threshold.
Packet dropped at position 0 due to buffer reaching threshold.
Packet dropped at position 6 due to buffer reaching threshold.
Packet dropped at position 2 due to buffer reaching threshold.
```

```
Microsoft Windows [Version 10.0.22631.2506]
(c) Microsoft Corporation. All rights reserved.

E:\Collage\BTECH-SEMESTER-7\Online Internet Protocols and Networking\Lab - 3>python Question-6(Client).py
Server response for Packet 0: ACK
Server response for Packet 1: ACK
Server response for Packet 2: ACK
Server response for Packet 3: ACK
Server response for Packet 4: ACK
Server response for Packet 5: ACK
Server response for Packet 6: ACK
Server response for Packet 7: ACK
Server response for Packet 8: EARLY RANDOM DROPPED
Server response for Packet 9: ACK
Server response for Packet 10: ACK
Server response for Packet 11: ACK
Server response for Packet 12: EARLY RANDOM DROPPED
Server response for Packet 13: EARLY RANDOM DROPPED
Server response for Packet 14: EARLY RANDOM DROPPED
Server response for Packet 15: ACK
Server response for Packet 16: ACK
Server response for Packet 17: EARLY RANDOM DROPPED
Server response for Packet 18: ACK
Server response for Packet 19: EARLY RANDOM DROPPED

E:\Collage\BTECH-SEMESTER-7\Online Internet Protocols and Networking\Lab - 3>
```