5/6/24. PRACTICAL - 5

AIM:
Experiment on Packet Capture tool: Wireshark
* Capture Network
* Decode Packet
* Define filter
* Analyze Problem.

DONE:
View Network all networks & speeds
Viewing of all packets incoming.

QO:
(1) Allows card to interpret all network packet
(2) No, ARP packets do not have transport layer header
(3) HTTP protocol uses port number by default
(4) Broadcast Ip address which is used to send packets to all devices on a specific network segment.
(5) DNS primarily uses UDP for its transport layer protocol.

RESULT:
Thus, experiment on Packet Capture tool wireshark is studied and observed.

21/8/24

Practical-6

AIM:

To implement error detection and correction using Hamming code concept. Make a test run to input data stream and verify error correction feature.

Hamming code is a set of error-correction that can be used to detect and correct the errors that can occur when the data is transmitted from the sender to the receiver.

Code:

```
import os
def text_to_binary(text):
    return ' '.join(format(ord(char), '08b')
                    for char in text)
def calculate_redundant_bits(m):
    r=0
    while (2**r) < (m+r+1)
        r+=1
    return r
def position_redundant_bits(data,r):
    j=0
    k=1
    m=len(data)
    res=''
for i in range(1, m+r+1):
    if i==2**j:
        res+='0';
        j+=1
```

```
        else:
            res += data[-k]
            k += 1
        return res[::-1]
    def calculate_parity_bits(arr, r):
        n = len(arr)
        for i in range(r):
            val = 0
            for j in range(1, n+1):
                if j & (2**i) == (2**i):
                    val = val ^ int(arr[-j])
            arr = arr[:n-(2**i)] + str(val) +
                  arr[n-(2**i)+1:]
        return arr

    def apply_hamming_code(data):
        m = len(data)
        r = calculate_redundant_bits(m)
        arranged_data = position_redundant_bits
                        (data, r)
        hamming_code = calculate_parity_bits(
                       arranged_data, r)
        return hamming_code

    def save_to_channel(hamming_code):
        with open('channel', 'w') as file:
            file.write(hamming_code)

    if __name__ == "__main__":
        text = input("enter the text")
        binary_data = text_to_binary(text)
        hamming_code = apply_hamming_code(
                       binary_data)
        save_to_channel(hamming_code)
```

```
#receiver.py
def read_from_channel():
    with open('channel', 'r') as file:
        return file.read()
def calculate_redundant_bits(m):
    r=0
    while(2**r) < (m+r+1):
        r+=1
    return r
def detect_error(arr,nr):
    n=len(arr)
    res=0
    for i in range(nr):
        val=0
        for j in range(1,n+1):
            if j&(2**i)==(2**i):
                val=val^int(arr[-j])
        res=res+val*(10**i)
    return int(str(res),2)
def correct_error(arr,pos):
    if pos>=1:
        arr=arr[:len(arr)-pos]+str(1-int(
                arr[len(arr)-pos]))
    return arr
def remove_redundant_bits(arr,m):
    n=len(arr)
    j=0
    res=''
    for i in range(1,n+1):
        if i!=2**j:
            res+=arr[-i];
        else:
            j+=1
    return res[::-1]
```

```
def binary_to_text(binary_data):
    text = ''
    for i in range(0, len(binary_data), 8):
        byte = binary_data[i:i+8]
        text += chr(int(byte, 2))
    return text
```

OUTPUT:

```
python encoder.py
Enter the text: data
data has been encoded and saved to
channel file
python receiver.py
Error detection at position 8
error is corrected
The decoded data is: data
```

RESULT:
Thus the program for hamming code is executed successfully.

18/9/24

SLIDING WINDOW PROTOCOL

AIM:

Write a program to implement flow control at data link layer using sliding window protocol simulate the flow of frames from one node to another.

PROGRAM:

sender.py:

```
import time
import os
def input_window_size():
    return int(input("Enter window size:"))
def input_text_message():
    return input("Enter text message").
def create_frames(text_message):
    frames = [(i, char) for i, char in enumerate
                                (text_message)]
    frames.append((len(text_message), 'END'))
    return frames
def write_to_file(filename, data):
    with open(filename, 'w') as file:
        for frame in data:
            file.write(f"{frame[0]},
                        {frame[1]}\n")
def read_from_file(filename):
    if not os.path.exists(filename):
        return []
    with open(filename, 'r') as file:
        return [line.strip().split(',') for
    line in file.readlines()]
```

```
def send_frames(frames, window_size):
    i=0
    while i<len(frames):
        window=frames[i:i+window_size]
        print(f"sending frames:{window}")
        write_to_file('send_buffer.txt', window)
        time.sleep(3)

    reciever_buffer=read_from_file('Recieverbuffer.txt')
    if not reciever_buffer:
        print("No acknowledgement recieved yet")
        continue
    ack_frame=reciever_buffer[0]
    ack_number, ack_type=int(ack_frame[0]),
                         ack_frame[1]
    if ack_type='ACK':
        print(f"Ack received for frame {ack_no},
                sending next set of frames")
    i+=window_size
    elif ack_type='NACK':
        print(f"NAck recieved for frame {ack_no},
    Resending frames from frame {ack_no}")
    i=ack_no
def main_sender():
    window_size=input_window_size()
    text_message=input_text_message()
    frames=create_frames(text_message)
    send_frames(frames, window_size)

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

Receiver.py

```
import random
import time
import os
def write_to_file(filename, data):
    with open(filename, 'w') as file:
        file.write(data)
def read_from_file(filename):
    if not os.path.exists(filename):
        return []
    with open(filename, 'r') as file:
        return [line.strip().split(',') for line in file.readlines()]
def process_frames(frames):
    acks = []
    frame_seen = set()
    for frame in frames:
        frame_number = int(frame[0])
        data = frame[1]
        if frame_number in frame_seen:
            continue
        print(f"Received frame {frame_number}: {data}")
        if random.choice([True, False]):
            print(f"Sending ACK for frame {frame_number}")
            acks.append(f"{frame_number},ACK\n")
            break
    return ''.join(acks)
```

```
def main_receiver():
    while True:
        time.sleep(3)
        frames = read_from_file("Sender_buffer.txt")
        if not frames:
            print("No frames to process, waiting")
            continue
        acks = process_frames(frames)
        write_to_file("Receiver_buffer.txt", acks)
        if any(frame[1] == 'END' for frame in frame):
            print("End of transmission received")
            break

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

OUTPUT :

```
Python sender.py
Enter window size: 3
Enter text message: hello
Sending frame: [(0, 'h'), (1, 'e'), (2, 'l')]
ACK received for frame 0, sending next
                set of frames
Sending frames: [(3, 'l'), (4, 'o'), (5, 'END')]
ACK received for frames 3, sending next
                set of frames.

python receiver.py
Receiver frame 0, h
Sending ACK for frame 0
Received frame 1: e
Sending ACK for frame 1
```

Received frame 2 : 1
Sending ACK for frame 2
Received frame 3 : 1
Sending ACK for frame 3
Received frame 4 : 0
Sending ACK for frame 4
Received frame 5 : END
Sending ACK for frame 5
End of transmission received.

RESULT:
Thus the program for sliding window is successfully executed.

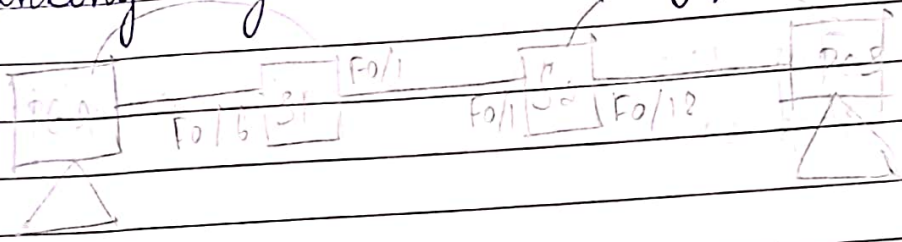
12/9/22

9/9/24

PRACTICAL-8

AIM:-
(a) Simulate Virtual LAN configuration using CISCO Packet Tracker Simulation

Packet Tracer - Configure VLANs and Trunking - Physical Mode Topology.



Addressing Table

Device	Interface	IP Address	Subnet Mask	Default Gateway
S1	VLAN-1	192.168.1.11	255.255.255.0	N/A
S2	VLAN 1	192.168.1.12	255.255.255.0	N/A
PC-A	NIC	192.168.10.3	255.255.255.0	192.168.10.1
PC-B	NIC	192.168.10.4	255.255.255.0	192.168.10.1

Part 1: Build the Network & Configure Basic Device Settings.
step 1: Build the Network as shown in the topology.
step 2: Configure basic settings for each switch
Step 3: Configure PC hosts.
step 4: Test connectivity.

Part 2: Create VLANs & Assign Switch Ports.
Step 1: Create VLANs on the Switches.
step 2: Assign VLANs to correct switch interfaces.

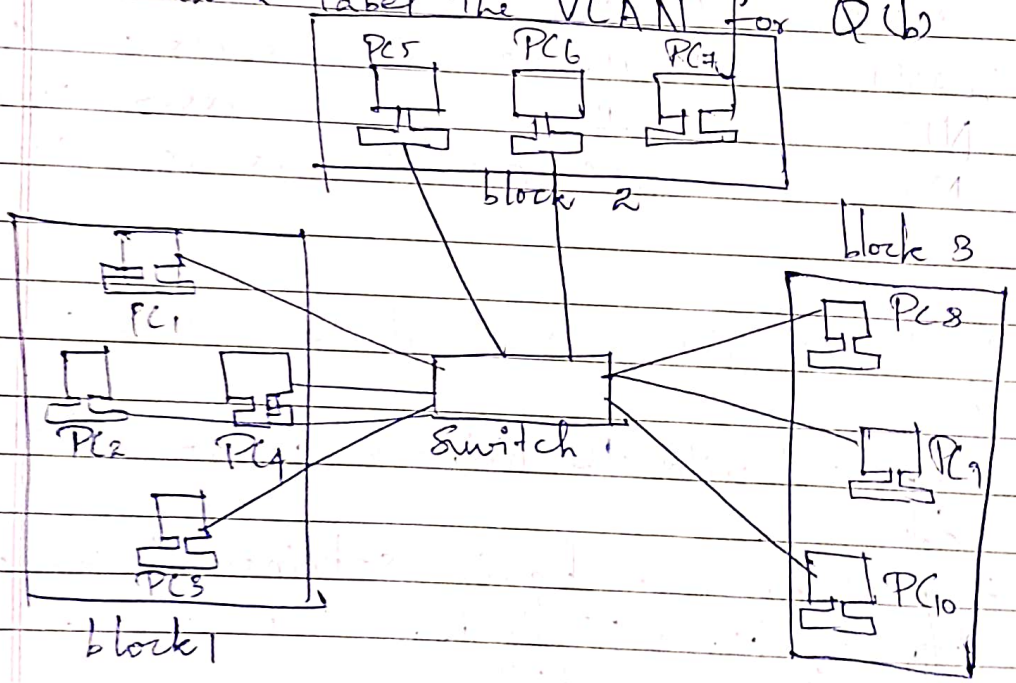
Part 3: Maintain VLAN Port Assignments and the VLAN Database:

Step 1: Assign a VLAN to multiple interfaces.
Step 2: Remove a VLAN Assignment from an interface.
Step 3: Remove a VLAN ID from VLAN database.

Part 4: Configure an 802.1Q Trunk between the Switches.
Step 1: Use DTP to initiate trunking on Fo/1.
Step 2: Manually configure trunk interface Fo/10.

Student Observation Questions:

(a) Draw & label the VLAN for Q(b)



(b) Show IP configuration for each device:

Device	IP Address	Subnet Mask	Default
PC1	192.168.20.2	255.255.255.0	192.168.20.1
PC2	192.168.20.3	255.255.255.0	192.168.20.1
PC3	192.168.20.4	255.255.255.0	192.168.20.1
PC4	192.168.20.5	255.255.255.0	192.168.20.1
PC5	192.168.20.6	255.255.255.0	192.168.20.1
PC6	192.168.20.7	255.255.255.0	192.168.20.1
PC7	192.168.20.8	255.255.255.0	192.168.20.1
PC8	192.168.20.9	255.255.255.0	192.168.20.1
PC9	192.168.20.10	255.255.255.0	192.168.20.1
PC10	192.168.20.11	255.255.255.0	192.168.20.1

(c) Write the commands used for VLAN configuration in switch.

```
switch > enable
switch # configure terminal
Switch (config) # vlan 10
Switch (config-vlan) # name Robotics
Switch (config-vlan) # exit

Switch (config) # interface range f 0/1-10
Switch (config-if-range) # switchport mode access
switch (config-if-range) # switchport access vlan 10

switch (config-if-range) # exit
```

RESULT:

Thus, the simulation of virtual LAN configuration using Cisco packet Tracer has been performed and the output is verified.

AIM:

(b) Configuration of wireless LAN using CISCO Packet Tracker.

Student Observation Questions:

(a) What is SSID of a wireless router.

SSID (Service Set Identifier) is the name of a wireless network. It is used to uniquely identify a wireless LAN.

(b) What is security key in a wireless router?

A security key in a wireless router is a password / encryption key used to secure wireless network.

RESULT:

Thus, a wireless LAN has been configured using CISCO Packet Tracer and output is verified.

21/9/24

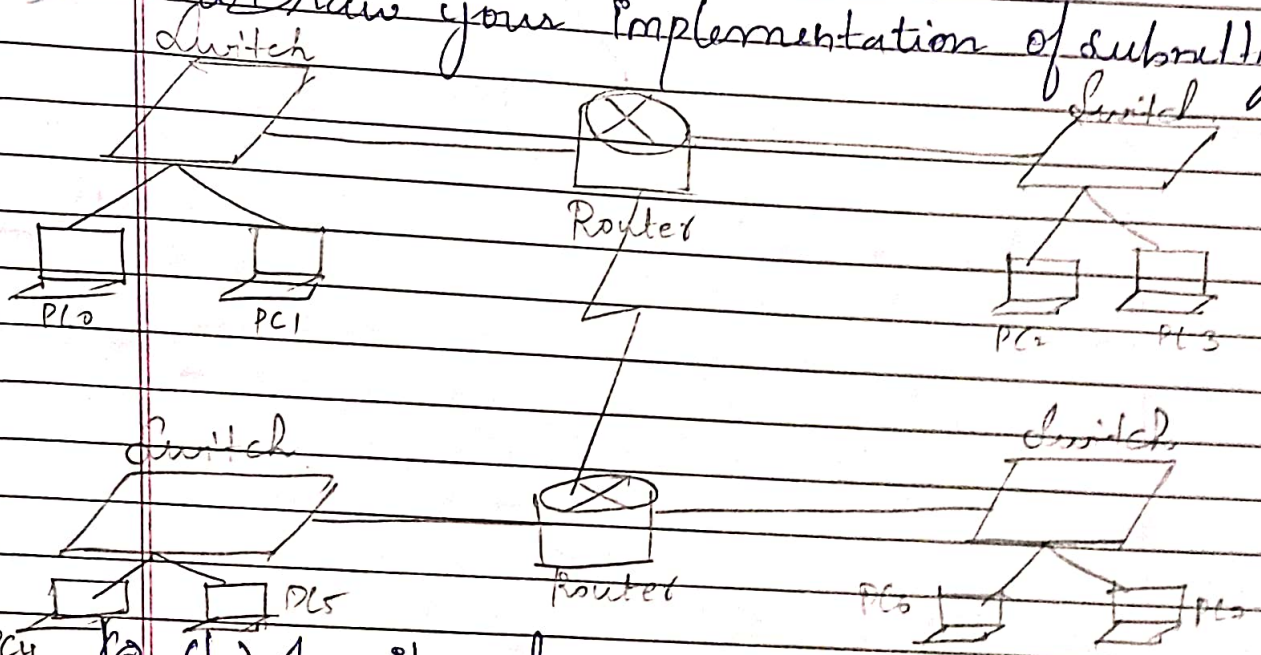
PRACTICAL-9

AIM:

Implementation of subnetting in Cisco Packet Tracer simulator.

Student Observation Questions:

(a) Draw your implementation of subnetting



(b) Write down your understanding of subnetting.

Classless IP subnetting is a technique that allows for more efficient use of IP address by allowing for subnet masks that are not just the default masks for each IP class. This means that we can divide our IP address space into smaller subnets, which can be useful when we have a limited number of IP address but need to create multiple networks.

(c) What is the advantage of implementing within a network?

Subnetting is useful when we have a limited no of. IP address but need to create multiple networks. This also enhances security.

RESULT:

Thus, subnetting has been implemented in cisco packet tracer & output verified.

25/9/24

PRACTICAL 10

AIM

(a) Internetworking with routers in CISCO PACKET TRACER Simulator

(b) Design & configure an internetwork using wireless router, DHCP server & internet cloud

Student Observations:

(1) Write down key features of configuring Wireless router and DHCP server

→ Access the Router's Configuration Page
→ Enter admin username & password
→ Set Basic Wireless Settings.
→ Enable DHCP Server.
→ Configure DNS Servers
→ MAC Address Filtering - Allow devices by MAC address
→ Apply changes & restart routers.

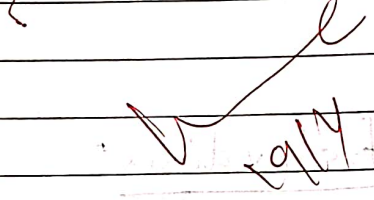
(2) Significance of DHCP Server in internetworking:

* Automated IP Address allocation.
* No duplicate IP Addresses
* Supports large-scale networks.
* Provides a centralized point for managing IP address.

(3) Design & Configure inter-network using switch, router, cables. Also show ip address conf. of each device.

RESULT:

Internetworking with routers is done successfully by CISCO.



1/10/24

PRACTICAL-11

AIM:

(a) Simulate Static Routing Configuration using CISCO Packet Tracer.

(b) Simulate RIP using CISCO packet tracer.

(a) 1. Adding Static Routes: each router knows only the networks directly connected to it. add static route to reach a network not directly connected.

Eg: Router0, networks 10.0.0.0/8, 20.0.0.0/8, & 40.0.0.0/8 are directly connected, but 30.0.0.0/8 & 50.0.0.0/8.

(2) Creating Main & Backup Routes

Administrative Distance decides preference of routes: the lower the AD, the higher the preference.

(3) Router Configurations

Configure static routes on each router for networks not directly connected.

(4) Verifying Router:

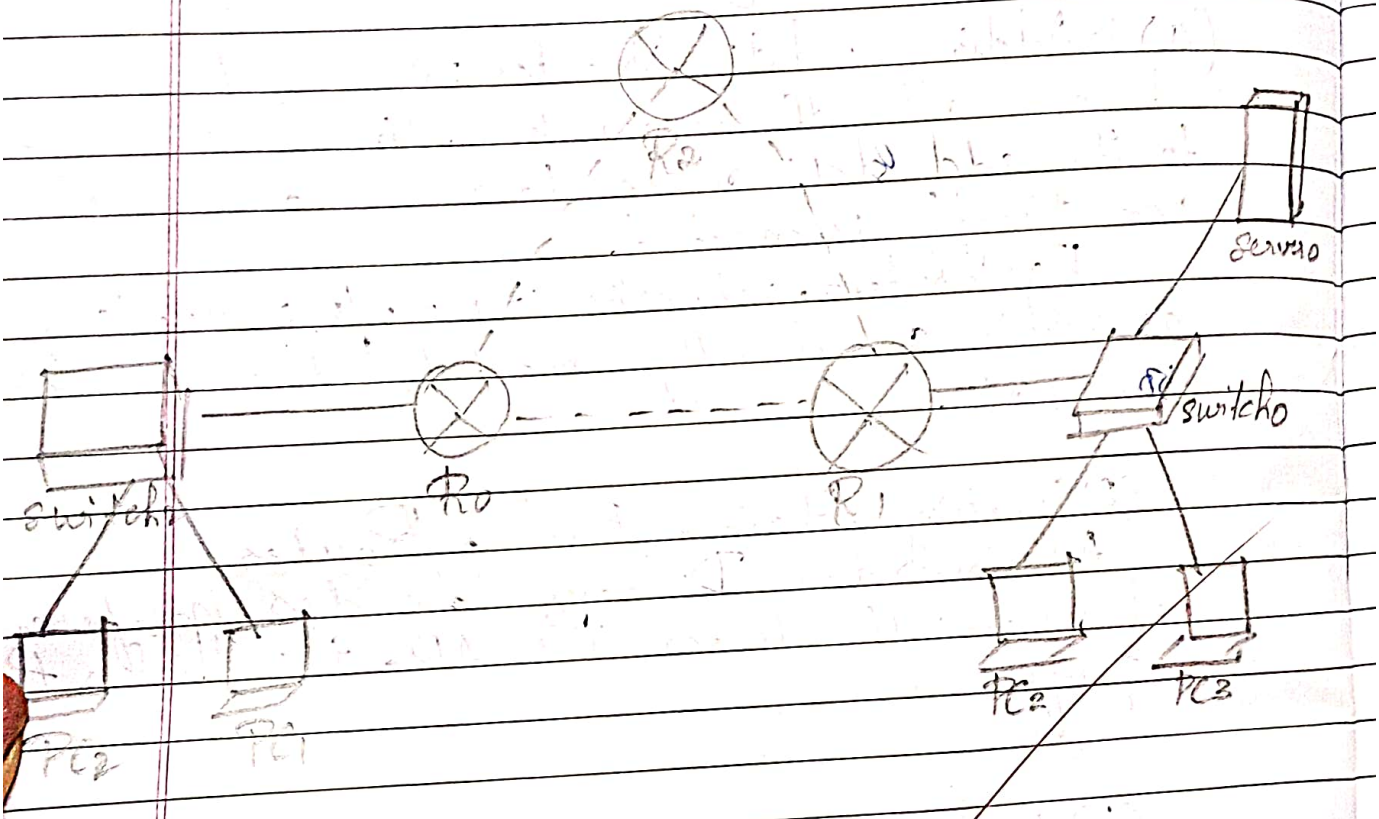
Verify routes by using commands: show ip route static.

(5) Testing Route Fail over:

→ Test connectivity using tracert or ping from a device on a connected network.

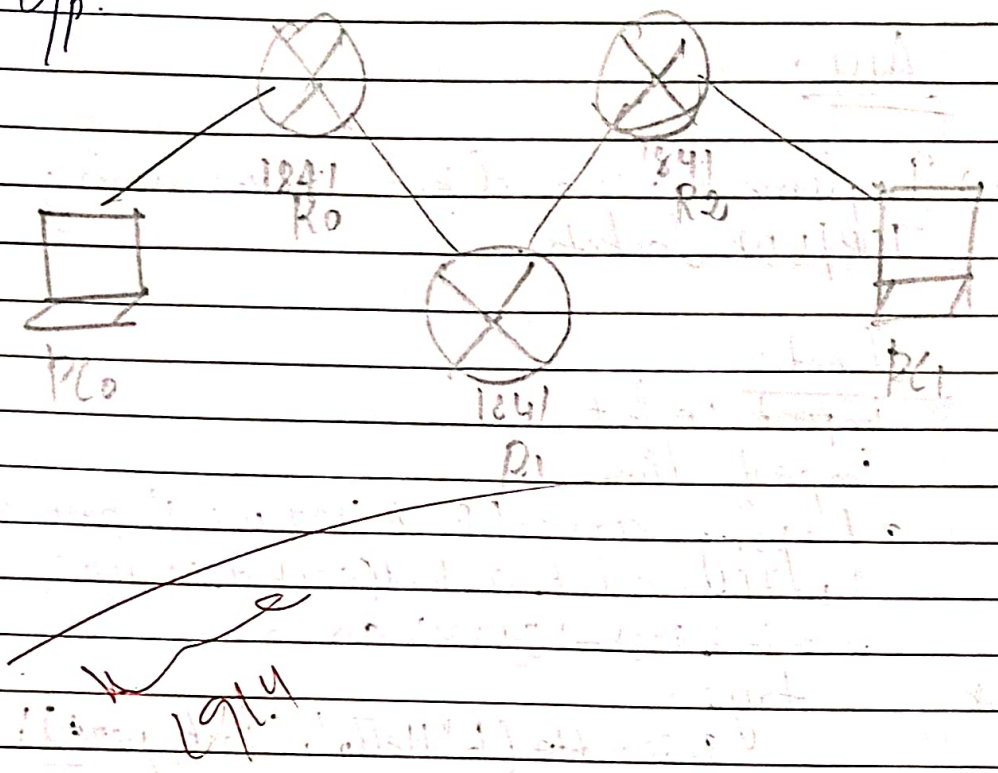
→ Disconnect or "break" the link on the main route.

(6) Deleting a Static Route:
show ip route static



(b)
(1.) Initial IP Configuration for Devices
(2) Assign IP Addresses to Devices - for PCs and Routers.
(3) Enable & Configure Interfaces on Routers.
(4) Configure RIP on Routers.
(5) Verify and Test Redundancy.
- use ping command on PC1
- use tracert to see RIP redirecting traffic through an alternate route.

o/p:



16/10/24

PRACTICAL-12

AIM:

(a) Implement echo client server using TCP/UDP Sockets.

Client:

```
import socket
import time
def ping_server(host='127.0.0.1', port=12345):
    with socket.socket(socket.AF_INET,
    socket.SOCK_DGRAM) as s:
        try:
            s.sendto(b"Hello", (host, port))
        except s.timeout:
            print("Request timed out")

if __name__ == "main":
        ping_server()
```

Server:

```
import socket
def start_server(host='127.0.0.1', port=12345):
with socket.socket(socket.AF_INET, socket.SOCK_
                    DGRAM) as s:
    s.bind((host, port))
    print(f"UDP Server running on {host}:{port}")

while True:
    data, addr = s.recvfrom(1024)
    print(f"Received message from {addr}:
            {data.decode()}")
if __name__ == "__main__":
        start_server()
```

O/P : python server.py
UDP server running on 127.0.0.1:12345
Received message from ('127.0.0.1', 59220):
Hello

python client.py
Received reply from server: Hello, client

(b) Implement chat client server using TCP/UDP sockets:

Chat serv.py

```
import socket
def recv1():
    port=12345
    host='127.0.0.1'
    with socket.socket(socket.AF_INET, socket.
                       SOCK_DGRAM) as s:
        s.bind((host,port))
        while (True):
            d, add = s.recvfrom(1024)
            print("Client", d.decode())
            a=input("Enter Reply")
            s.sendto(a.encode(), add)
            if(a=="end"):
                break
        exit
recv1()
```

recvn2.py

```
import socket
import time
def recvn2(a):
    host='127.0.0.1'
    port=12345
    with socket.socket(socket.AF_INET, socket.
    SOCK_DGRAM) as s:
        s.sendto(a.encode(), (host, port))
        d, addr = s.recvfrom(1024)
        print(f'{d.decode()}')
while(True):
    a=input("Enter Message")
    if (a=="end"):
        recvn2(a)
        break
    else:
        recvn2(a)
```

sendto
recvfrom

O/P:

```
python .\chat_serv.py
Client f'hi'
Enter Reply hello
Client f"How are you"
Enter Reply Im fine
```

```
python .\recvn.py
Enter Message hi
f'hello'
Enter Messages How are you
f'Im fine'
Enter Message
```

19/10/24

PRACTICAL 13

AIM:

Implement your own ping program.

Server.py

```
import socket
def start_server(host='127.0.0.1', port=12345):
    with socket.socket(socket.AF_INET,
        socket.SOCK_DGRAM) as s:
            s.bind((host, port))
            print(f"UDP Server running on {host}:
                {port}")
        while True:
            data, addr = s.recvfrom(1024)
            print(f"Received message from {addr}:
                {data.decode()}")
            s.sendto(b'Pong', addr)

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

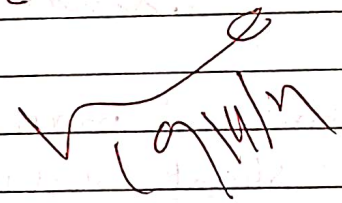
client.py

```
Import socket
import time
def ping_server(host='127.0.0.1', port=12345):
    with socket.socket(socket.AF_INET, socket
        SOCK_DGRAM) as s:
        try:
            s.settimeout(2)
            start = time.time()
            s.sendto(b'Ping', (host, port))
```

```
except socket.timeout:
    print("Request timed out").
if __name__ == "__main__":
    ping_server()
```

Output

```
python server.py
UDP server running on 127.0.0.1:12345
Received message from ('127.0.0.1', 53009): ping

python client.py
Received Pong from ('127.0.0.1', 12345) in
0.00 seconds.
```



23/10/24

PRACTICAL 14:
PACKET SNIFFING

AIM:

To implement Packet Sniffing program.

SOURCE CODE:

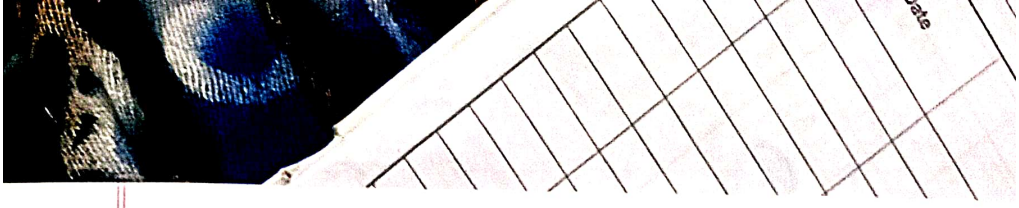
```
from scapy.all import sniff
from scapy.layers.inet import IP, TCP, UDP, ICMP

def packet_callback(packet):
    if IP in packet:
        ip_layer = packet[IP]
        protocol = ip_layer.proto
        src_ip = ip_layer.src
        dst_ip = ip_layer.dst

        protocol_name = ""
        if protocol == 1:
            protocol_name = "ICMP"
        elif protocol == 6:
            protocol_name = "TCP"
        elif protocol == 17:
            protocol_name = "UDP"
        else:
            protocol_name = "Unknown Protocol"

        print(f"Protocol: {protocol_name}")
        print(f"Source IP: {src_ip}")
        print("-" * 50)
```

OUTPUT :

Packet : TCP
SOURCE IP : 20.247.158.142
Destination IP : 172.20.10.2

RESULT :

Thus Packet sniffing Program is successfully executed & verified.



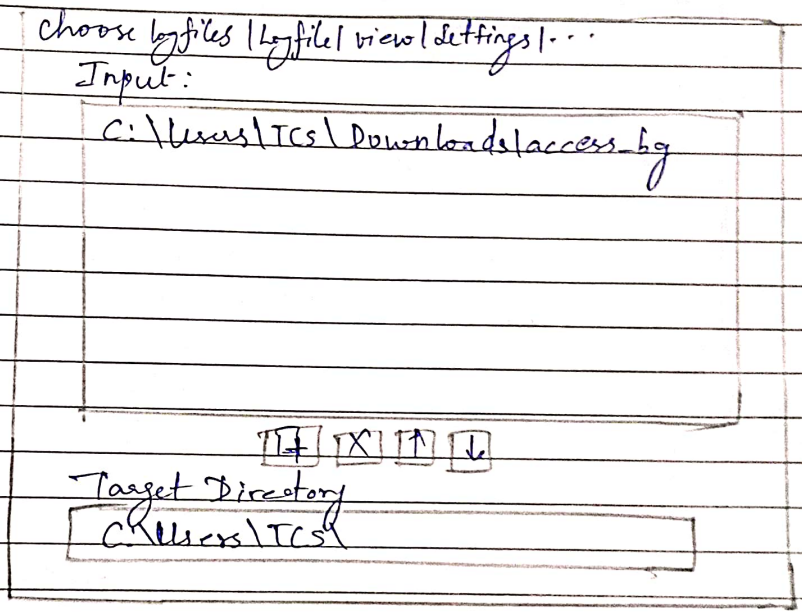
23/10/24

PRACTICAL 15

AIM: To analyze the different types of web logs using Webalizer tool.

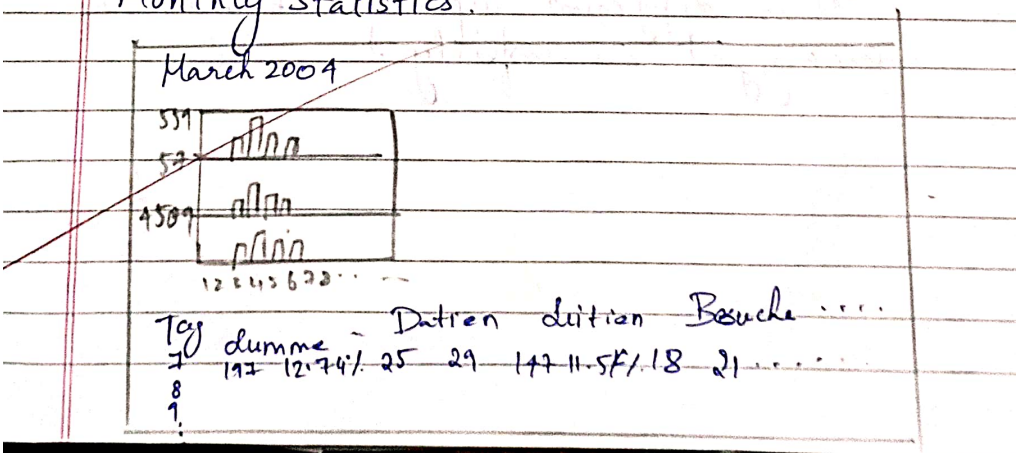
PROCEDURE:

(1) Run webalizer windows version
(2) Input web log file
(3) Press Run webalizer

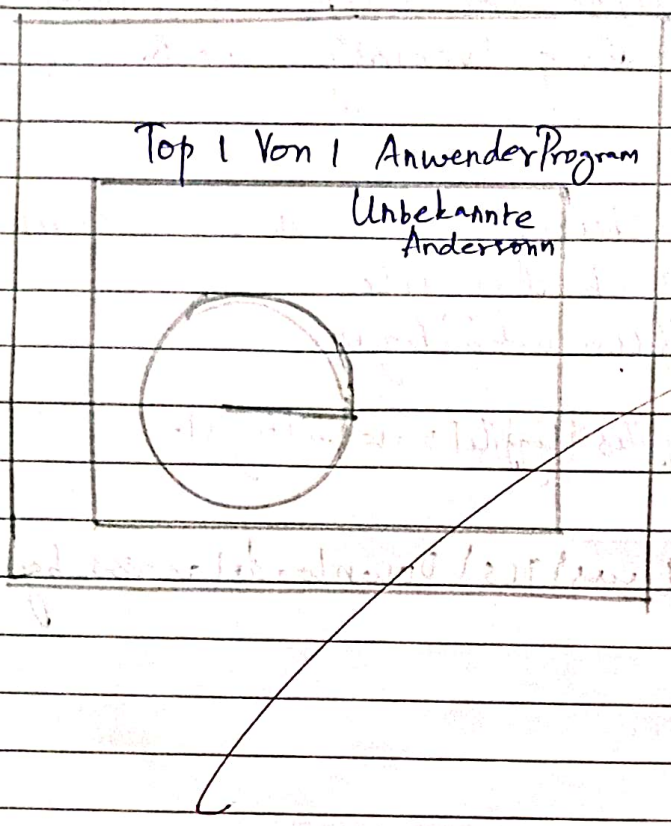


OUTPUT:

Monthly Statistics:



USER-AGENTS:



RESULT:

Thus different types of web logs are analyzed successfully.

INDEX

NAME: Divya Sundhari STD.: CSE SEC.: B ROLL NO.: 68 SUB.: _____

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	16/7/24	Basic Networking commands		9
2.	23/7/24	Different types of Network cables		3/8 7
3	30/7/24	Behaviour of network devices using CISCO PACKET Tracer.		3/8 9
4.	3/8/24	Set up a LAN		7/8 8
5.	5/8/24	Experiments on Packet Tracer tool: Wireshark		21/8 9
6.	21/8/24	Hamming code		8/9/24 8
7.	11/9/24	Sliding window protocol		
8.	9/9/24	(a) Virtual LAN		
8.	9/9/24	(b) Wireless LAN		
9.	21/9/24	SUBNETTING		
10.	25/9/24	(a) Internetworking using DHCP Server		9/10/24
	25/9/24			
11.	9/10/24	(a) Static Routing Configuration		
	9/10/24	(b) RIP		
12.	16/10/24	(a) Echo client server		
	16/10/24	(b) Chat Client server		
13.	19/10/24	PING Program		
14.	23/10/24	Packet Sniffing		
15.	23/10/24	Webalizer Tool		

completed