Indeoc

1.	16.7.2024	Network commands	
2.	23.7.2024	Different types of network cables	
3.	30.7.2024	Packet Tracer Installation	
4.	6.8.2024	LAN Configuration	
5.	9.8.2024	Wireshark	
6.	20.8.2024	Error Correction	
7.	6.9.2024	Sliding window protocol	
8a	18.10.2024	Virtual LAN	
8b	18.10.2024	Wireless LAN	
9.	8.10.2024	Classfull Subnetting	
10.	18.10.2024	Internetworking	
11.a	18.10.2024	Stating Routing Config	
11.b	18.10.2024	RIP	

12 a TCP/UDP Socket

12 b Chat Client Server

13 Ping Program

14 RAW Socket Packet Sniffing

15. Webalizer Tool

Completed

16/7/24

Experiment – 1

1. arp – a

2. hostname

3. ipconfig/all [ifconfig]

4. nbstat -a [nmblookup -A 192.168.1.1]

5. netstat

6. nslookup [nslookup google.com]

7. pathping [Unique to windows]

8. ping

9. Route

O/P

1. _gateway (10.0.2.2) at 52:54:00:12:35:02 [ether] on enp0s3
2. localhost - live
3. enp0s3: flags=4163<UP, BROADCAST, RUNNING, MULTICAST> mtu 1500
   inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255

   lo: flags=73<UP, LOOPBACK, RUNNING> mtu 65536
   inet 127.0.0.1 netmask 255.0.0.0
4. Looking up status of 192.168.1.1
   No reply from 192.168.1.1
5. Active Internet Connections (w/o servers)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	localhost-live:58446	166.188.117.34.bc:https	Established
tcp	0	0	43492	104.18.32.115:https	Established
udp	0	0	Bootpc	_gateway:bootps	Established

6. Server: 127.0.0.53
   Address: 127.0.0.53#53

   Non-authoritative answer:
   Name: google.com
   Address: 216.58.203.46
   Name: google.com
   Address: 2404:6800:4009:80f::200e

8. Limited to Windows

9. PING google.com (216.58.203.46) 56(84) bytes of data
   64 bytes from ~~216.58.~~ hkg12s10-in-f46.1e100.net (216.58.203.46): icmp_seq=1 ttl= time=
   64 bytes from hkg12s10-in-f46.1e100.net (216.58.203.46): icmp_seq=2 ttl=59 time = 37.7 ms
   64 bytes from hkg12s10-in-f46.1e100.net (216.58.203.46): icmp_seq=3 ttl=59 time=36.6 ms

   --- google.com ping statistics ---
   3 packets ~~received~~ transmitted, 3 received, 0% packet loss, time 200 ms

   rtt min/avg/max/mdev = 36.582/43.447/56.041/8.917 ms

1. ip

ip <optio

a. # ip

b. # ip

c. # ip ad

d. # ip

e. # ip

f. # ip

g. # ip ro

h. # ip

i. # ip ro

j. # ip

k. # ip

[enps03 → enp0s3]

1. ip

ip <options> <object> <command>

a. # ip address show

b. # ip address add 192.168.1.254/24 dev enps03

c. # ip address del 192.168.1.254/24 dev enps03

d. # ip link set eth0 ~~down~~ up

e. # ip link set eth0 down

f. # ip link set eth0 promisc on

g. # ip route add default via 192.168.1.254 dev eth0

h. # ip route add 192.168.1.0/24 ~~via~~ via 192.168.1.254

i. # ip route add 192.168.1.0/24 dev eth0

j. # ip route delete 192.168.1.0/24 via 192.168.1.254

k. # ip route get 10.10.1.4

ip address show

1: lo: <LOOPBACK, UP, LOWER_UP> mtu 65536 qdisc noqueue
state UNKNOWN group default qlen 1000
link/loopback 00:00:00:00:00:00 brd
00:00:00:00:00:00

2: enp0s3: <BROADCAST, MULTICAST, UP, LOWER_UP> mtu 1500
qdisc fq_codel state UP group default qlen 1000
link/ether 08:00:27:1f:01:65 brd ff:ff:ff:ff:ff:ff

ip address add 192.168.1.254/24 dev enp0s3
RTNETLINK answers: File exists

ip address del 192.168.1.254/24 dev enp0s3

ip address add 192.168.1.254/24 dev enp0s3

ip address add 192.168.1.254/24 dev enp0s3

ip route get 10.10.1.4
10.10.1.4 via 10.0.2.2 dev enp0s3 src 10.0.2.15 uid 0
cache

ifconfig

enp0s3: flags = 4163 <UP, BROADCAST, RUNNING, MULTICAST>
mtu 1500
inet 10.0.2.15 netmask 255.255.255.0 broadcast
10.0.2.255

lo: flags = 73 <UP, LOOPBACK, RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0

ip link set enp0s3 up

ip link set enp0s3 down

ip link set enp0s3 promisc on

ip route add 192.168.1.0/24 dev enp0s3
Error: Device for nexthop is not up.

2. mtr

mtr <options> hostname/IP

a. # mtr google.com

b. # mtr -g google.com

c. # mtr -b google.com

d. # mtr -c 10 google.com

localhost-live (2401:4900:1cc9:2ab:1cle:6de:ac1b:a2fe)
—> google.com (2404:6800:4009:811::200e)

Scale: .: 984 ms 1: 937 ms 2: 859 ms 3: 749 ms a: 609 ms
b: 437 ms c: 234 ms

#mtr -l google.com

x 0 338000
x 1 338001
x 2 338002
x 3 338003
x 4 33004

3. tcpdump

a. # dnf install -y tcpdump

b. # tcpdump -D

c. # tcpdump -i eth0

d. # tcpdump -i eth0 -c 10

e. # tcpdump -i eth0 -c 10 host 8.8.8.8

f. # tcpdump -i eth0 src host 8.8.8.8

g. # tcpdump -i eth0 dst host 8.8.8.8

h. # tcpdump -i eth0 net 10.1.0.0 mask 255.255.255.0

i. # tcpdump -i eth0 net 10.0.0.0/24

j. # tcpdump -i eth0 port 53

k. # tcpdump -i eth0 host 8.8.8.8 and port 53

l. # tcpdump -i eth0 -c 10 host www.google.com and port 443

```
#dnf install -y tcpdump
Last metadata expiration check: 0:43:06 ago on
                    Thu 11 Jul 2024 03:48:46 PM EDT.
Package tcpdump-14:4.99.4-2.fc39.x86_64 is already
installed.
Dependencies resolved.
Nothing to do.
Complete!
#tcpdump -D
1. enp0s3 [Up, Running, Connected]
2. lo [Up, Running, Loopback]
3. bluetooth-monitor (Bluetooth Linux Monitor) [Wireless]
#tcpdump -i enp0s3
dropped privs to tcpdump
tcpdump: verbose outpumpt suppressed, use -v[v]... for
full protocol decode
listening on enp0s3, link-type EN10MB(Ethernet), snapshot
length 262144 bytes
16:32:48.655388 ARP, Request who-has 192.168.1.12 tell _gateway,
length 46
16:32:50.761108 IP 192.168.1.10.6537 > 255.255.255.255.6537: UDP,
length 201
^C
361 ~~packets~~ packets captured
361 packets received by filter
0 packets dropped by kernel

#tcpdump -i enp0s3 host 8.8.8.8
~~tcpdump:~~
dropped privs to tcp-dump
tcpdump: verbose output suppressed, use -v[v]... for full
protocol
listening on enp0s3, link-type EN10MB(Ethernet), snapshot
length 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
```

#tcpdump -i enp0s3 -c 3
dropped privs to tcpdump
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode

16:33:19.629767 IP localhost-live.49664 > mad05s23-in-f3.1e100.net.https: Flags [P.], seq 4162835473:4162835512, ack 26699339727, win 501, options [nop,nop,TS val 1893368936 ecr 3471518623], length 39
3 packets captured
3 packets received by filter
0 packets dropped by kernel

#tcpdump -i enp0s3 host google.com and port 443
dropped privs to tcpdump
tcpdump: verbose output suppressed, use -v[v]... for full pw-protocol decode
linker listening on enp0s3, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C
0 packets captured
4 packets received by filter
0 packets dropped by kernel

$ nmcli

NAME
wired conne

lo

$ nmcli
connect
success

$ nmcli
auto

$ nmcli
auto

$ ip a
2: ep
mtu

$ ip
default
192.168

$ cat
namese
option
search

$ nmcli connection show

NAME	UUID	TYPE	DEVICE
wired connection 1	59fbdda8-3af4-3001-8551-dafed13f2909e	ethernet	enp0s3
lo	01d740d2-42c4-4cle-9452-07fe9915ae60	loopback	lo

$ nmcli connection add con-name cenp0s2 type ethernet
connection 'cenp0s2' (6406 79c - 6702 - 4761 - afb3 - 04809 0aeb71d) successfully added.

$ nmcli connection modify "Wired connection 1" ipv4.method auto

$ nmcli connection modify "Wired connection 1" ipv6.method auto

$ ip address show enp0s3

2: enp0s3 : <BROADCAST, MULTICAST, PROMISC, UP, LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
link/ether 08:00:27:1f:01:65 brd ff:ff:ff:ff:ff:ff

$ ip route show default

default via 192.168.137.1 dev enp0s3 proto dhcp src 192.168.137.92 metric 1000

$ cat /etc/resolv.conf

nameserver 127.0.0.53
options edns0 trust-ad
search mshome.net

Student Observation

1. Which command is used to find the reachability of a host machine from your device?

Ping command

2. Which command will give the details of hops taken by a packet to reach its destination?

mtr (Matt's traceroute)

3. Which command displays the IP configuration of your machine?

IP <options> <object> <command>

4. Which command displays the TCP port status in your machine?

netstat

5. Write the modify ip configuration in a Linux machine

i. address add 192.168.1.254/24 dev enp0s3

ii. ip addres del 192.168.1.254/24 dev enp0s3

Result

Thus networking commands of both Linux & Windows are studied and executed successfully

16/7/24

23/7/24

Aim:

a) Un

are

UTP

i. C

Ap

23/7/24

Experiment 2

Aim:

Study of different types of network cables

a) Understand different types of network cable

Different types of cable used in networking are:

1. Unshielded Twisted Pair (UTP) Cable
2. Shielded Twisted Pair (STP) Cable
3. Coaxial cable
4. Fibre optic cable

UTP

i. Category 3

Max. Data Transmission (MDT): 10 bps

Adv:

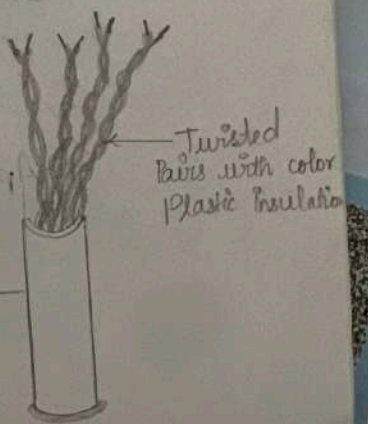
- Cheaper in cost
- Easy to install as they have a smaller overall diameter

Dis-adv:

- More prone to EMI

Application

- 10 base T Ethernet



a) UTP

ii. Category 5

MDT: Up to 100 Mbps

Application:
- Fast ethernet
- Gigabit ethernet

iii. Category 5c

MDT: 1Gbps

Application:
- Fast ethernet
- Gigabit ethernet

STP

Category 6, 6a

MDT: 10 GBps

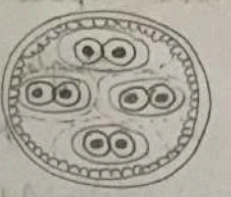
Advantages:
- Shielded
- Faster than UTP
- Less susceptible to noise and interference

Disadvantages
- Expensive
- Greater installation effort

Application
- Gigabit Ethernet
- 10G Ethernet (55 m)

SSTP

Category 7



SSTP

MDT: 10 Gbps

Application

- Gigabit Ethernet
- 10G Ethernet (100 m)

Coaxial Cable

RG-6, RG-59, RG-11

MDT: 10 – 100 Mbps

Adv

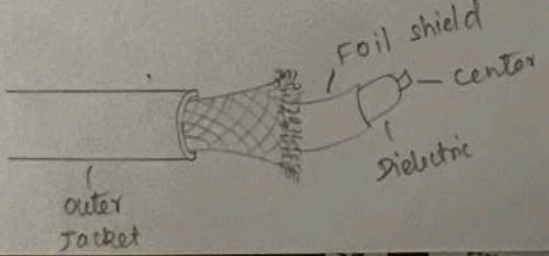
- High bandwidth
- Immune to interference
- Low loss bandwidth
- Versatile

Dis-adv

- Limited distance
- Cost
- Size is bulky

Application

- Speed of signal is 500 m
- Television network
- High speed internet connections



Fibre Optics Cable

Single mode, Multi mode

MDT: 100 Gbps

Adv:

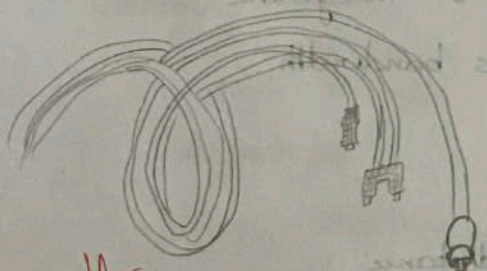
- High speed
- High bandwidth
- High security
- Long distance

Dis-adv

- Expensive
- Requires skilled installers

Application

- Max dist is around 100 m



23/7/24

Result

Different types of network cables have been studied

30/7/24

Aim:

30/7/24 Experiment - 3

Aim:

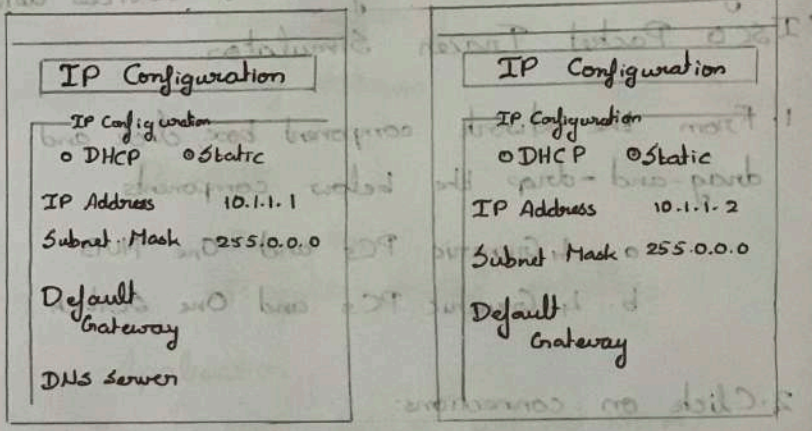
To study packet tracer tool installation and user interface

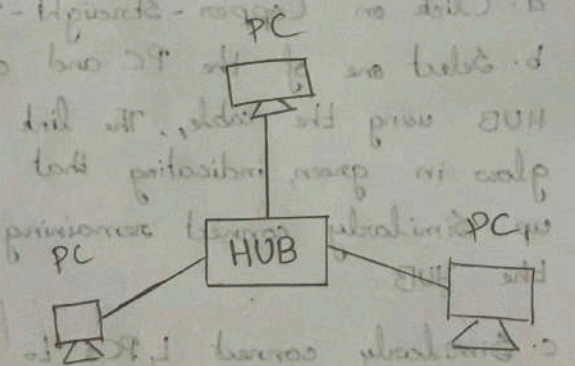
Analyze the behaviour of network devices using CISCO Packet Tracer Simulation

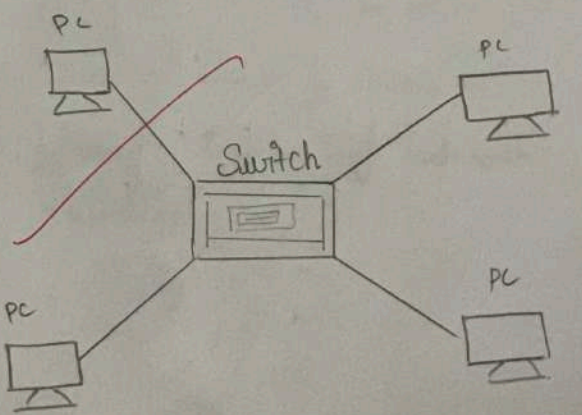
1. From the network component box, click and drag-and-drop the below components
   a. 4 Generic PCs and One HUB
   b. 4 Generic PCs and One Switch

2. Click on connections:
   a. Click on Copper-Straight-Through cable
   b. Select one of the PC and connect it to HUB using the cable,. The link LED should glow in green, indicating that the link is up. Similarly connect remaining 3 PCs to the HUB
   c. Similarly connect 4 PCs to the switch using copper straight-through cable.

3. Click on the PCs connected to HUB, go to the Desktop tab, click on the IP configuration and enter an IP address and subnet mask. Here the default gateway and DNS server information is not needed as there are only two end devices in the network.







From your observation write down the behaviour of Switch and Hub in terms of following the packets received by them

1-Hub

— Forwards data packets to each and every connected computer.

Switch

— Forwards data to specified destination

Find out the network topology implemented in your college & draw and label the topology:

Mesh Topology

Result:

The study of packet tracer tool ~~imp~~ installation & user interface overview has been successfully completed.

30/7/24

6/8/24

Practical 4

Aim

Setup and configure a LAN using a Switch and Ethernet cables

How to set-up a LAN?

Step 1: Plan and design an appropriate network topology taking into account network requirements and equipment location.

Step 2: You can take 4 computers, a switch with 8, 16 or 24 ports which is sufficient for networks of these sizes and 4 ethernet cables

Step 3: Connect your computers to network switch via an Ethernet cable.

Step 4: Assign IP Address to your PCs

1. Log on the client computer as Architecture or as owner.
2. Click Network and Internet Connections
3. Right click Local Area Connections / Ethernet → Go to properties → Select Internet Protocol (TCP/IPV4) → Click Properties → Select IP address option and assign IP address.

Step 5: Configure a network switch:

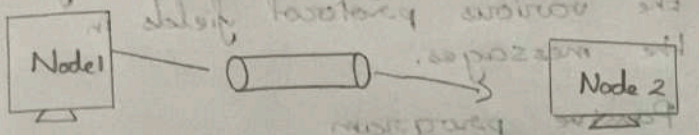
1. Connect your computer to the switch
2. Log in to web interface
3. Configure basic settings
4. Assign IP address

Step 6: Check the connectivity b/w switch and other machine by using ping command in the command prompt of the device.

Step 7: Select a folder → Go to properties → Click Sharing tab → Share with everyone on the same LAN

Step 8: Try to access the shared ~~file~~ folder from other computers in the network.

Observation



IP Configuration:

Node 1: IPV4 Address: 10.1.1.1

Node 2: IPV4 Address: 10.1.1.2

Outcome:

The file has been shared successfully

Result:

Thus the set up of LAN using switch and Ethernet cable is successful and file has been shared

6/8/24

5

capture tool

ng sent/received
ides

the contents of
col fields in

packets to itself
ts addressed to it
copy of all

Diagnostic Tools

plication

DS
all/ Ethernet

sent/received

Application
(Ex: web browser, ftp client)

Transport TCP/UDP
Network(IP)
Link (Ethernet)
Physical

Wireshark

A network analysis tool formerly as Ethereal, captures packets in real tim display them in human-readable for

What we can do?

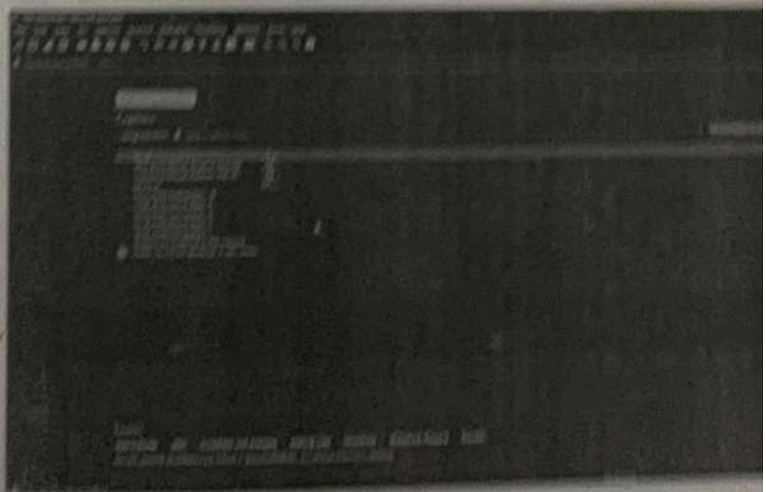
— Capture network traffic
— Analyze problems

Uses

— Network admin: Trouble sho problems
— Developers: Debug protocol im

Capturing Packets

Launch Wireshark and doub the name of a network interface Capture to start capturing pac the interface.



Wireshark

A network analysis tool formerly known as Ethereal, captures packets in real time and display them in human-readable format

What we can do?

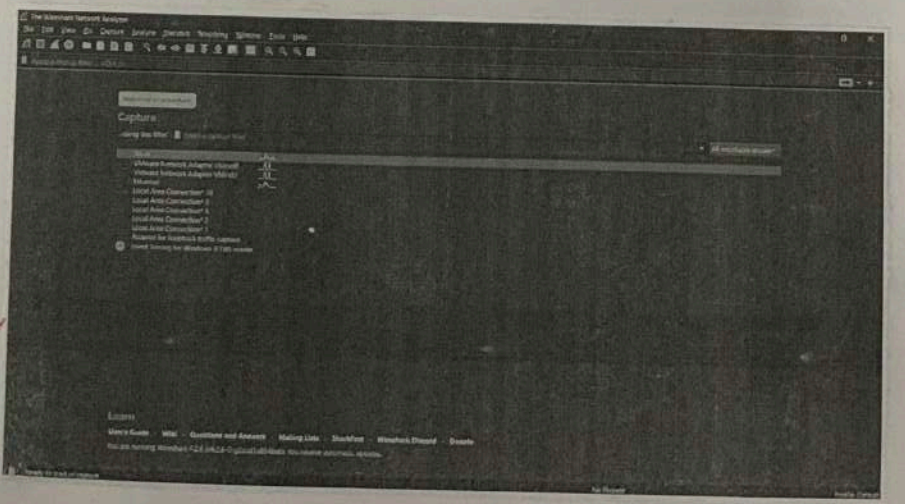
- Capture network traffic
- Analyze problems

Uses

- Network admin: Trouble shoot network problems
- Develops : Debug protocol implementations

Capturing Packets

Launch Wireshark and double-click the name of a network interface under Capture to start capturing packets on the interface.

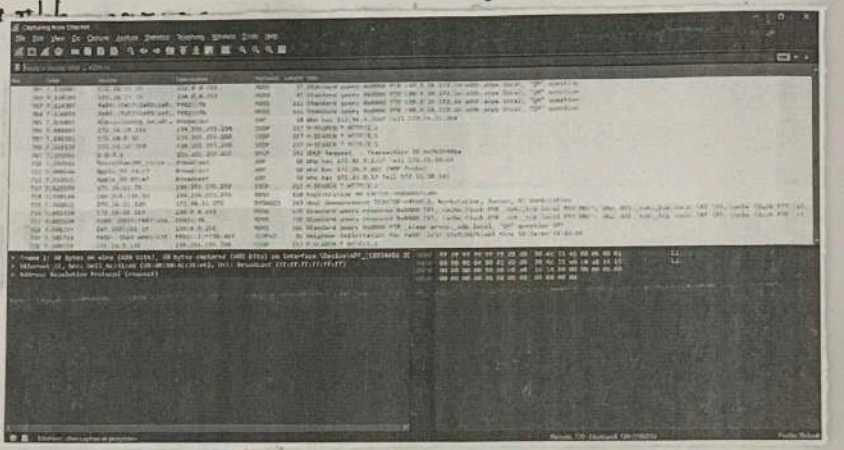


Application
(Ex: WWW browser, ftp client)
↑
Transport TCP/UDP
Network (IP)
Link (Ethernet)
Physical
↓
to/from network

Colour Coding

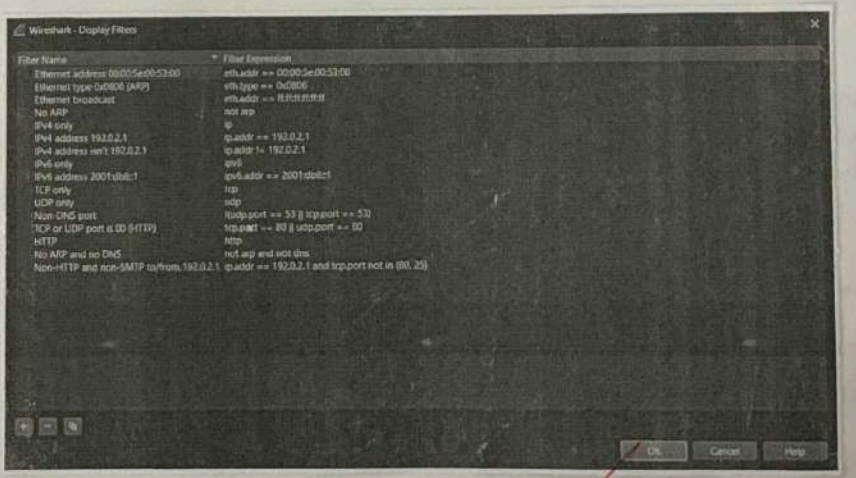
Wireshark uses colours to help you identify the types of traffic at a glance.

By default, light purple is TCP traffic, light blue is UDP traffic and black identifies packets



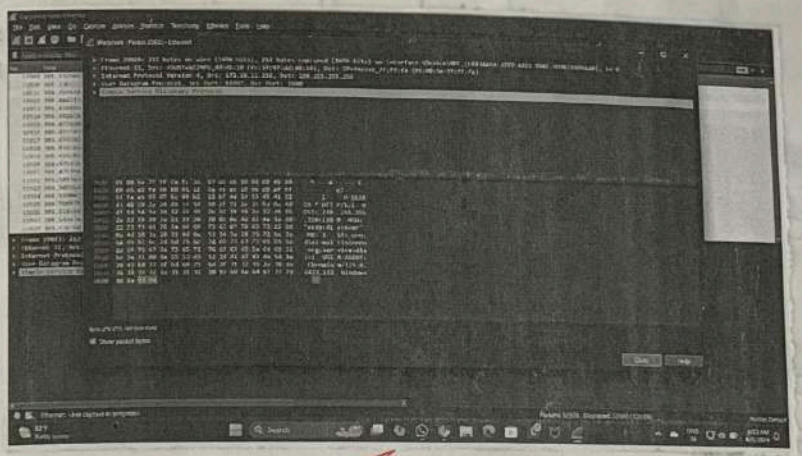
Filtering Packets

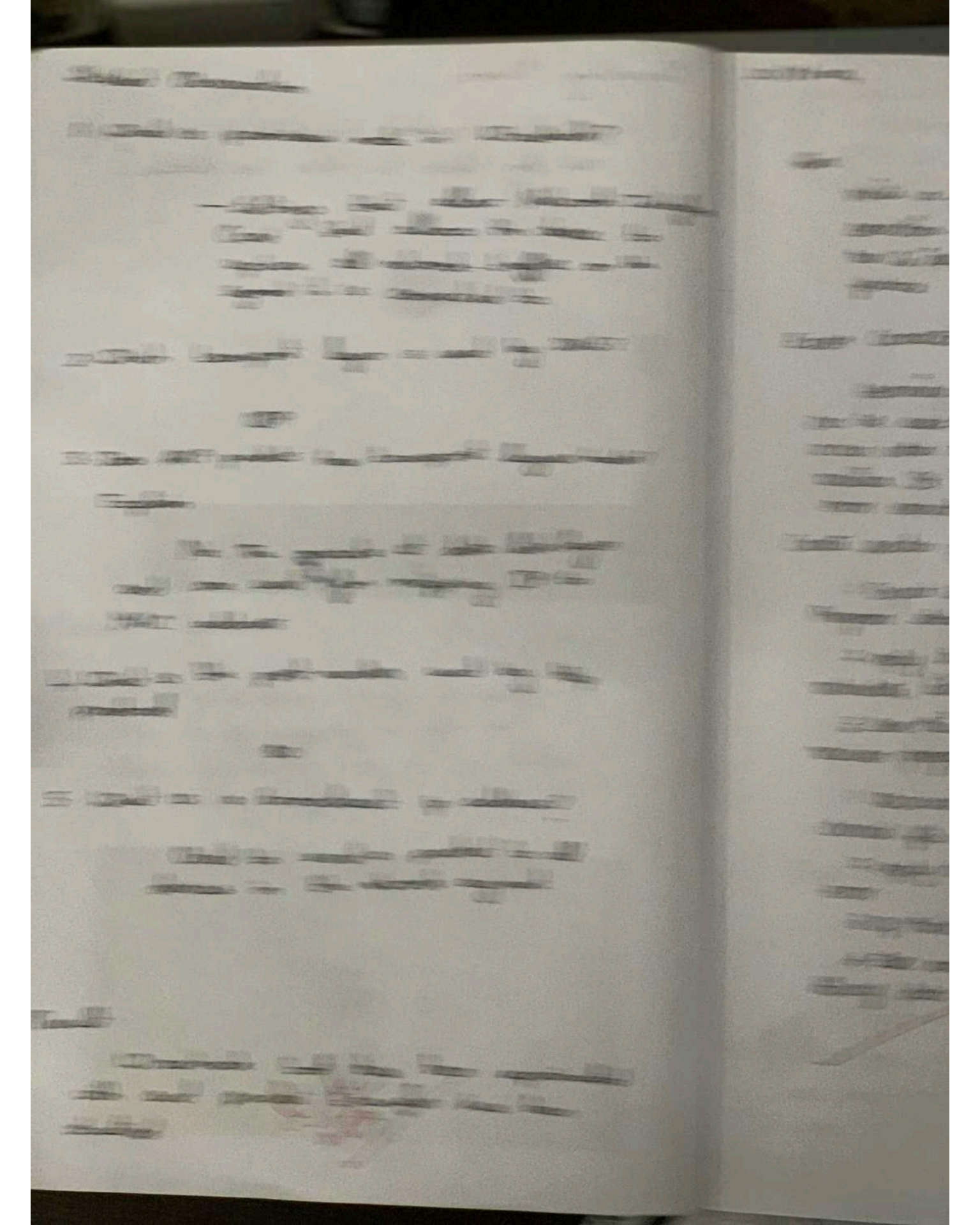
The most basic way to apply a filter is by typing it into the filter box at the top of the window and clicking apply.



Inspecting Packets

- Click a packet to select it and you can dig down to view its details.





20/8/2024

Practical - 6

Aim:

Write a program to implement error detection and correction using HAMMING CODE concept. Make a test run to input data stream and verify error correction feature.

Error Correction at Data Link layer

Hamming Code is a set of error-correction codes that can be used to detect and correct the error that can occur when the data is transmitted from the sender to the receiver. It is a technique developed by R.W Hamming for error correction.

Create sender program with below functions.

1. Input to sender file should be a text of any length. Program should convert the text to binary.
2. Apply hamming code concept on the binary data and add redundate bits to it.
3. Save this output in a field called channel. Create a receiver program with below features.

1. Receiver program should be read the input from channel file.
2. Apply hamming code on the binary data to check for error.
3. If there is an error, display the position of the error.
4. Else remove the redundant bits and convert the binary data to ascii and display the output.

Code :-

```
def to_binary(char):
    return format(ord(char), '08b')
def calculate_redundant_bits(m):
    r=0
    while (1<<r)< (m+r+1):
        r+=1
    return r
def insert_redundant_bits(data, m, r):
    encoded = ['0']*(m+r)
    j=0
    for i in range(1,m+r+1):
        if(i&(i-1))==0:
            continue
        encoded[i-1] = data[j]
        j+=1
    for i in range(r):
        pos = (1<<i)
        xor_sum = 0
        for j in range(pos-1, m+r, pos*2):
            for k in range(j, min(j+pos, m+r)):
                xor_sum ^= int(encoded[k])
        encoded[pos-1] = str(xor_sum)
    return ''.join(encoded)
def apply_hamming_code(text):
    binary_data = ' '.join([to_binary(char) for char in text])
    Print(f"Binary representation : {binary_data}')
    m = len(binary_data)
    r = calculate_redundate_bits(m)
    encoded_data = ''.join([to_binary(char) for char i text])
    encoded_data = insert_redundant_bits(binary_data, m, r)
    Print(f"Encoded data with redundant bits : {encoded_data}")
    Printf"Redundant bit positions: {[1<<i for i in range(r)]}"
    return encoded_data, r
```

```
def check_and_correct(encoded_data, r):
    m = len(encoded_data) - r
    error_position = 0
    for i in range(r):
        pos = (1<<i)
        xor_sum = 0
        for j in range(pos-1, len(encoded_data), pos*2):
            for k in range(j, min(j+pos, len(encoded_data))):
                xor_sum^= int(encoded_data)
        error_position |= (xor_sum<<i)
    if error_position:
        Print(f"Error found at position: {error_position}")
        corrected_data = list(encoded_data)
        corrected_data[error_position-1] = '0' if encoded_data
                                [error_position-1]=='1' else '1'
        Print(f"Corrected data: {''.join(corrected_data)}")
    else:
        Print("No error found")
        corrected_data = list(encoded_data)
    return ''.join(corrected_data), error_position
def extract_original_data(corrected_data, r):
    m = len(corrected_data) - r
    data_bits [ ]
    for i in range(1, m+r+1):
        if (i & (i-1)) != 0:
            data_bits.append(corrected_data[i-1]
    binary_str = ''.join(data_bits)
    return ''.join([chr(int(binary_str[i:i+8], 2)) for i in range
                                (0, len(binary_str), 8)]
def main():
    text = input("Enter the text:")
    encoded_data, r = apply_hamming_code(text)
    change = input("Do you want to change any bit in encoded
                                data:").strip().
                                lower()
    if change == 'yes':
        Position = int(input(f"Enter the bit position to change
                                (1-{len(encoded_data)}):"))
```

```
if 1<=position<=len(encoded_data):
    is_redundant = position in [1<<i for i range (r)]
    encoded_data = list (encoded_data)
    encoded_data [position-1]='0' if encoded_data[position-1]
                                  =='1' else '1'

    encoded_data = ''.join(encoded_data)
    Print(f"Data after bit change : {encoded_data}")
if is_redundant:
    Print (f"Note : you change a redundant bit at position
                                  {position}.")

else :
    Print ("Invalid Position, No changes made")
corrected_data, error_position = check_and_correct(encoded_data)
if change=='Yes' and is_redundant and error_position==
                                  Position :
    Print ("The change in the redundant bit was corrected
                                  successfully")
elif change=="Yes' and is_redundant and error_position!=
                                  Position :
    Print (f"The change in the redundant bit was not detected")
original_text = extract_original_data (corrected_data.r)
Printf ("Decoded text : {original_text}")
if __name__=="__main__":
    main()
```

Output

Enter
Binary
Redun
Redun
Do y
Enter
Error
corr
Dec

RESULT

Th
and

i range (n)]

oded_data[position-1]
=='1' else '1'

ta)

coded_data {")

t bit at position
{position}.")

made")

nd_correct(encoded_data,)

error_position==
Position:

bit was corrected
successfully)

and error_position!=
Position:
bit was not detected)

corrected_data, r)

)

Output

```
Enter the text : k
Binary representation : 01101011
Redundant bit of Encoded data : 100111011011
Redundant bit position: [1, 2, 4, 8]
Do you wanna to change bit in encoded data : yes
Enter the bit position (1-12) : 10
Error found at position : 11
corrected data : 100111011011
Decoded text : k.
```

RESULT:

The program has been executed successfully and the output has been verified.

20/8/24

Practical 7

Sliding Window

Aim :

To write programs to ~~achieve~~ implement flow control at data link layer using Sliding Window Protocol.

Sender.py

```
import time
def send_frames(frames, window_size):
    current_frame = 0
    acked_frames = [False] * len(frames)
    while current_frame < len(frames):
        sender_buffer = []
        for i in range(window_size):
            frame_index = current_frame + i
            if frame_index < len(frames) and
                not acked_frames[frame_index]:
                frame = f"Frame No: {frame_index},
                DATA: {frames[frame_index]}"
                sender_buffer.append(frame)
        if sender_buffer:
            with open("Sender_Buffer.txt","w")
                                    as f:
                for frame in sender_buffer:
                    f.write(f"{frame}\n")
        print("Frames sent:", sender_buffer)
    time.sleep(2)
```

```
with open("Received_Buffer.txt","r") as f:
    ack_data = f.read().strip().split("\n")
print("Received ACK/NACK:", ack_data)

for ack in ack_data:
    if "ACK" in ack:
        ack_no = int(ack.split()[1])
        acked_frames[ack_no] = True
    elif "NACK" in ack:
        nack_no = int(ack.split()[1])
        acked_frames[nack_no] = False
        current_frame = nack_no
    if all(acked_frames[current_frame : current_frame
                        + window_size]):
        current_frame += window_size

window_size = int(input("Enter window size: "))
message = input("Enter the message to send: ")

frames = [char for char in message]

send_frames(frames, window_size)
```

Receiver.py

```
import time
def receive_frames():
    while True:
        with open("Sender_Buffer.txt", "r") as f:
            sender_buffer = f.read().strip().split("\n")
        print("Received frames: sender_buffer)
        receiver_buffer = []
        for frame in sender_buffer:
            try:
                frame_no = int(frame.split(",")[0]
                               .split()[-1])
            except(ValueError, IndexError):
                print(f"Invalid frame format: {frame}")
                continue

            if frame_no %2 == 0:
                ack = f"ACK {frame_no}"
            else
                ack = f"NACK {frame_no}"
            receiver_buffer.append(ack)
        with open("Receiver_Buffer.txt", "w" as f:
            for ack in receiver_buffer:
                f.write(f"{ack}\n")
        print("ACK/NACK sent: " receiver_buffer
        time.sleep(2)
receive_frames()
```

Output:

Enter
Enter

Sent
Sent
Send
Sent
Sent

All fr
wait

Rece
Rece
Rec
Rec
Rece

All

Tran

RESU

the

Output:

```
Enter the window size : 5
Enter Text message : Hello

Sent : Frame No : 0 , Data : H
Sent : Frame No : 1 , Data : e
Sent : Frame No: 2 , Data : l
Sent : Frame No: 3 , Data : l
Sent : Frame No : 4 . Data : o

All frames sent
waiting for ACK...
Received Frame No : 0 . Data : H
Received Frame No : 1 . Data : e
Received Frame No : 2 . Data : l
Received Frame No : 3 . Data : l
Received Frame No : 4 . Data : o
All frames are received . Stopping receiver

Transmission completed.
```

RESULT :

6/9/24

The program has been successfully executed and the output is verified.

Practical 8a

Virtual LAN

Aim:

Simulate virtual configuration using Cisco packet tracer simulation

Procedure:

1. Create the network using 1 switch & 4 PCs as shown in figure
2. Check the network by sending packets.
3. Assign IP addresses to the PC.

PC0	-	10.0.0.1
PC1	-	10.0.0.2
PC2	-	10.0.0.3
PC3	-	10.0.0.4

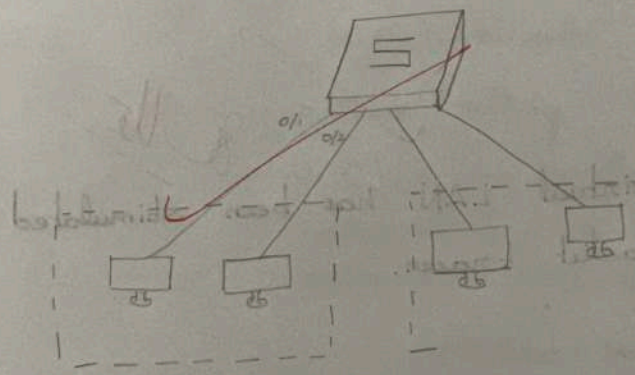
4. Right click on switch and then run the following commands in CLI

```
>enable
# conf t
#vlan 2
# name office
# exit
#vlan 3
# name home
# exit
```

```
# interface fastEthernet 0/1
# switchport access vlan 2
# exit

# interface fastEthernet 0/2
# switchport access vlan 2
# exit

# interface fastEthernet 0/3
# switchport access vlan 3
# exit

# interface fastEthernet 0/4
# switchport access vlan 3
# exit
```

5. Now try pinging packets from PCs to one another.

Diagramatic Representation



Observation

When packet transferred from PC0 to PC3, the packets are successfully transferred.

Similarly for PC, within same VLAN packets are transferred successfully.

Whereas for PCs cant transfer packets out of it's VLAN.

Output

Fire	Last Status	Source	Destination	Type	Color	Time	Periodic	Num
	Successful	PC0	PC1	ICMP		0.000	N	0
	Failed	PC1	PC2	ICMP		0.655	N	1

Result

Thus the virtual LAN has been stimulated using Cisco packet tracer.

Practical 8b

Wireless LAN

Aim

To configure wireless LAN using Cisco Packet tracer

Procedure

1. Create network 3 PCs & a wireless router.

2. Right click on the wireless router. Note the IP address. It would be 192.168.0.1

   → Disable DHCP server

   → Save setting

3. Click on wireless on the same page

   → Change Network name to "MyHomeNetwork"

   → Save setting

   → Click on wireless security under wireless & change security mode to WEP from disabled.

   ⇒ Add key1 : 0123456789

   ⇒ Save settings

   → Close the window

4. Now assign IP to the PC

   PC0 : 192.168.0.5

   → Gateway : 192.168.0.1

PC1 : 192.168.0.6

—> Gateway : 192.168.0.1

PC2 : 192.168.0.7

—> Gateway : 192.168.0.1

5. Now click on PC0

—> Click physical option

—> Turn off that

—> Drag the part in the diagram to the left menu.

—> Drag the 'WMP300N' to the port from the left.

—> Turn on that

—> Now click on desktop
PC Wireless ↵

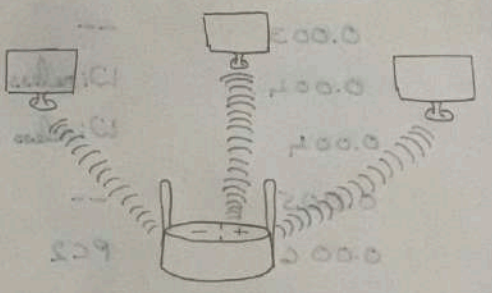
—> Click connect —> Site information
Connect ↵

—> Type '01234567' in the WEP key 1

~~6. Re~~ —> Connect

6. Repeat step 5 for the PC1 & PC2

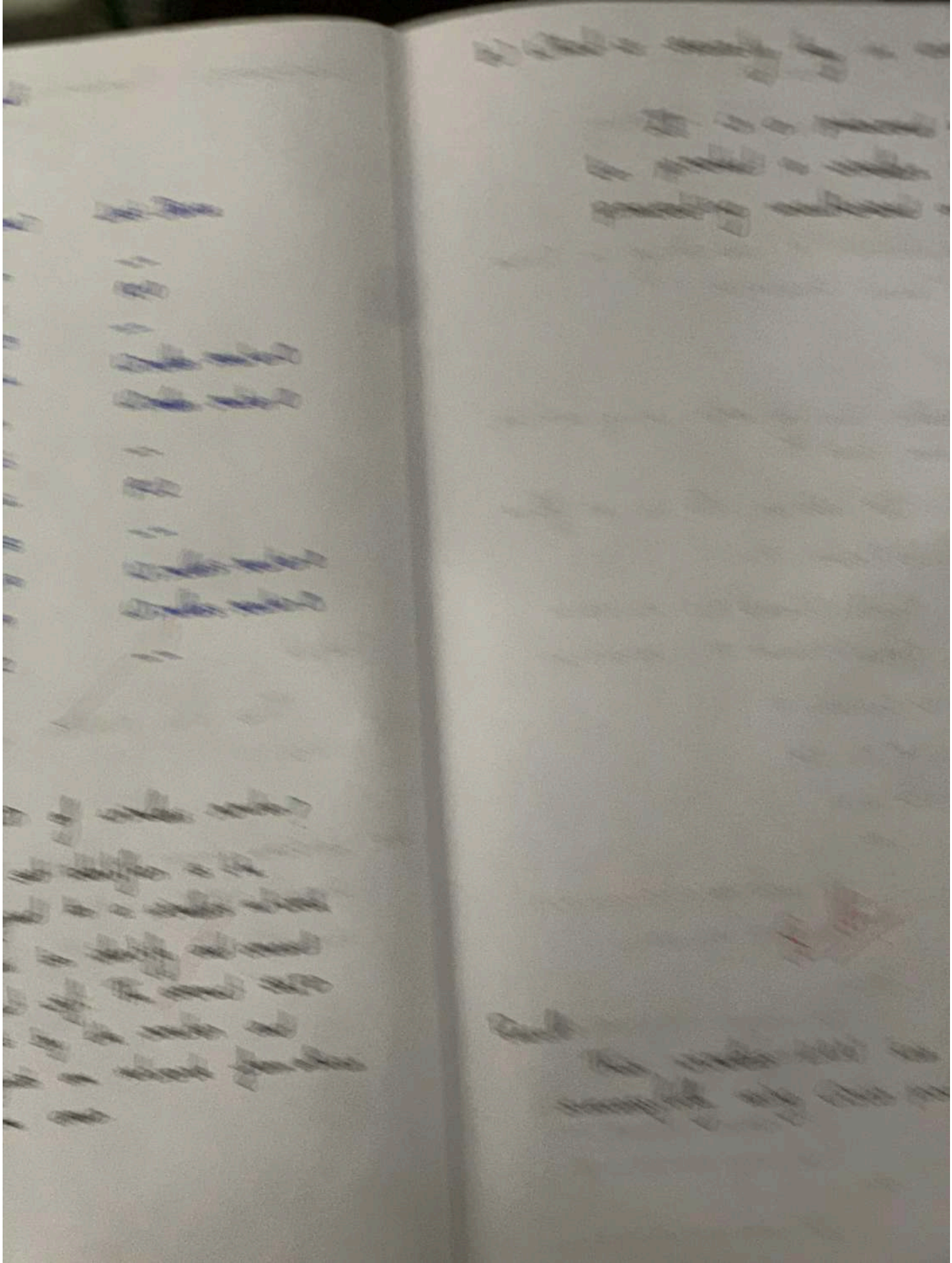
Diagrammatic Representation



Output

Thus the packets could be transmitted via

Fire	Last Status	Source	Destination	Type	time	period	num
	Successful	PC0	PC2	ICMP	0.000	N	0
	Successful	PC1	PC0	ICMP	0.566	N	1



b) What is security key in wireless router?

It is a password or code used to protect a wireless network preventing unauthorized access.

Result:

Thus wireless LAN has been configured successfully using Cisco packet tracer

Practical 9

Class full Subnetting

Aim:

Implentation of subnetting in Cisco Packet Tracer simulator

Procedure:

1. Create the network using switches, router and PC.
2. The IP address will be as follows

—> Router R1

GigabitEthernet 0/0 : 192.168.1.1

GigabitEthernet 0/1 : 192.168.2.1

—> Switch S1

* No IP

—> LAN1

. PC 0

IP address : 192.168.1.11

Gateway : 192.168.1.1

. PC 1

IP address : 192.168.1.12

Gateway : 192.168.1.12

. PC 2

IP address : 192.168.1.13

Gateway : 192.168.1.13

. PC 3

IP address : 192.168.1.14

—> Switch S2

* No IP

—> LAN 2

- PC 4

  IP: 192.168.2.11

  Gateway: 192.168.2.1

- PC 5

  IP: 192.168.2.12

  Gateway: 192.168.2.1

- PC 6

  IP: 192.168.2.13

  Gateway: 192.168.2.1

- PC 7

  IP: 192.168.2.14

  Gateway: 192.168.2.1

Practical 9

Output

Now lets assume the sender is PC1 and receiver is PC 7

Simulation Panel

Event List

vis	time	Last device
	0.000	--
	0.003	PC1
	0.005	Switch S1
	0.008	Router R1
	0.010	Switch S2
	0.013	PC 9
	0.015	Switch S2
	0.018	Router R1
	0.020	Switch S1
	0.023	--

Fire	Last Status	Source	Destination	type	time	num
	Successful	PC1	PC 9	ICMP	0.000	0

Student Observation

a) Write down your understanding of subnetting

Subnetting is the process of dividing a large IP address network and manageable section called subnets

Each subnet act as independent network & allow devices connected to it to communicate within the subnet & control the traffic btw subnets.

b) Advantages of implementing subnetting within a network

—> Efficient IP management based on requirement

—> Reduce network congestion - Limit broadcast traffic to individual subnets.

Result

The implementation of subnetting has been done successfully.

Practical 11b

RIP

Aim

To simulate RIP using Cisco Packet Tracer

Procedure

1. Create network using 3 PCs & 4 routers

2. Assign IP address

PC 0

IP 10.1.1.1

Gateway 10.1.1.2

PC 1

IP 200.1.1.1

Gateway 200.1.1.2

PC 2

IP 222.2.2.2

Gateway 222.2.2.12

Router 3

gig 0/0 20.1.1.1

0/1 192.168.1.1

0/2 10.1.1.1

Router 2

gig 0/0 20.1.1.2

0/1 172.1.1.1

0/2 200.1.1.2

Router 1

gig 0/0 192.168.1.3

0/1 172.1.1.2

0/2 217.1.1.1

Router 4

gig 0/0 217.1.1.2

0/1 222.2.2.12

3. Click on router 3

—> Click config —> RIP

—> Enter network 10.0.0.0 —> Add

—> Enter network 20.0.0.0 —> Add

—> Enter network 192.168.1.0 —> Add

This step is done inorder to add the neighbouring network addresses for router 3.

4. Do same for router 2, 1 & 4

5. Now to display the routing table click on router (say router 1)

—> then on CLI & type the command

#exit
#exit
#show ip route

Output

~~R 10.0.0.0/8~~

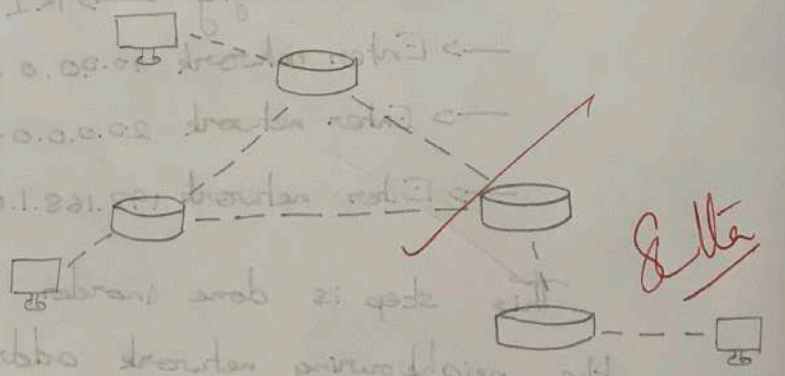
R. 10.0.0.0/8 via 192.168.1.1 gig 0/0

R. 20.0.0.0/8 via 192.168.1.1 gig 0/0

172.1.0.0/16 is variable connected 2 subnet 2 masks

C. 172.1.0.0/16 is directly connected gig 0/1

L. 172.1.1.2/32 is directly connected gig 0/1



Result

Thus rip is simulated using Cisco packet tracer successfully.

Practical 12a

Aim:

Implement echo client server using TCP/UDP sockets

Source Code

TCP Echo Server

```
import socket

def echo_server():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  as server:
        server.bind("localhost", 12345)
        server.listen(1)
        print("TCP Server Listening...")
        conn, addr = server.accept()
        with conn:
            print(f"Connected by {addr}")
            while data := conn.recv(1024):
                conn.sendall(data)

echo_server()
```

TCP Echo Client

```
import socket

def echo_client():
    with socket.socket(socket.AF_INET,
socket.SOCK_STREAM) as client:
        client.connect("localhost", 12345)
        client.sendall(b"Hello, TCP Server")
        print("Received:", client.recv(1024))

echo_client()
```

Practical 12b

Aim:

Implement chat client server using TCP/UDP sockets

Source Code

TCP Chat Server

```
import socket, threading

def chat_server():
    server = socket.socket(socket.AF_INET,
                           socket.SOCK_STREAM)
    server.bind("localhost", 12345)
    server.listen()
    clients = []

    def handle(client):
        while True:
            try:
                msg = client.recv(1024)
                for c in clients:
                    if c != client:
                        c.send(msg)
            except:
                clients.remove(client)
                break
    while True:
        client, _ = server.accept()
        clients.append(client)
        threading.Thread(target = handle, args=(client,)).start()
chat_server()
```

TCP Chat Client

```
import socket, threading

def chat_client()
    client = socket.socket(socket.AF_INET,
                           socket.SOCK_STREAM)
    client.connect("localhost", 12345)
    threading.Thread(target=lambda:
[print(client.recv(1024).decode() for _ in iter(int, 1)])
                    .start()

chat_client()
```

Practical 13

Aim

To implement ping program

Source code

```
import socket
import struct
import time
import os

def checksum(data):
    s = sum((data[i] << 8) + (data[i+1] if i+1 < len(data)
else 0) for i in range(0, len(data), 2))
    s = (s >> 16) + (s & 0xFFFF)
    return ~s & 0xFFFF

def ping(dest_addr):
    with socket.socket(socket.AF_INET,
    socket.SOCK_RAW, socket.IPPROTO_ICMP)
    as sock:
        sock.settimeout(1)
        pack_id = os.getpid() & 0xFFFF
        header = struct.pack("!BBHHH", 8, 0, 0,
                             pack_id, 1)
        data = struct.pack("d", time.time())
        packet = header[:2] + struct.pack("!H",
            checksum(header + data)) + header[4:] + data
        sock.sendto(packet, (dest_addr, 1))
```

```
        start = time.time()
        try:
            sock.recv(1024)
            end
            print(f"Reply from {dest_addr}:
                time={(time.time() - start)*1000:.2f}ms")
        except socket.timeout:
            print("Request timed out")

ping("8.8.8.8")
```

Practical 14

Aim:

Write a code using RAW sockets

```
import socket
def packet_sniffer():
    with socket.socket(socket.AF_PACKET,
        socket.SOCK_RAW, socket.ntohs(0x0003))
        as sniffer:
        print("Sniffer started. Capturing
            packets...")
        while True:
            packet, _ = sniffer.recvfrom(65565)
            print(f"Packet: {packet[:64].hex()}")

packet_sniffer()
```

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 (download from web)
3. ~~Process~~ Press ~~run~~ run webalizer.