



Name: Dhruv Hatkar

Reg No.-21BAI1218

Subject – Computer Networks

Slot – L43 + L44

Index :

Exp. No.	Date	Pg.No
1.	26/4/ 2023	3
2.	6/5/2023	11
3.	10/5/2023	16
4.	13/5/2023	29
5.	20/5/2023	36
6.	25/5/2023	40
7.	7/5/2023	63
8.	14/6/2023	70
9.	17/6/2023	75
10.	21/6/2023	77
11.	28/6/2023	81
12.	12/7/2023	85

EX 1(A)

ifconfig

- Network Management tool
- Used for configuration of network interfaces.
- Syntax -
ifconfig [-v] [-a] [-s] [interface]
- Command -
ifconfig -a
ifconfig wlan0
- Output-

```
Δ > ~
ifconfig -a
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 1176 bytes 93390 (91.2 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1176 bytes 93390 (91.2 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.136.84 netmask 255.255.255.0 broadcast 192.168.136.255
    inet6 fe80::f8ec:a1fe:9d8b:b4d6 prefixlen 64 scopeid 0x20<link>
    inet6 2409:40f4:101d:d52e:2350:390a:f9c5:cefl prefixlen 64 scopeid 0x0<global>
    ether 48:e7:da:5c:80:6f txqueuelen 1000 (Ethernet)
    RX packets 464080 bytes 623844977 (594.9 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 206619 bytes 18988497 (18.1 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

ping

- Packet Internet Groper is the most commonly used tool for troubleshooting a network.
- It is invoked using the ping command.

```
ping www.google.com
PING www.google.com (del11s20-in-x04.1e100.net (2404:6800:4002:82b::2004)) 56 data bytes
64 bytes from del11s20-in-x04.1e100.net (2404:6800:4002:82b::2004): icmp_seq=1 ttl=57 time=198 ms
64 bytes from del11s20-in-x04.1e100.net (2404:6800:4002:82b::2004): icmp_seq=2 ttl=57 time=99.0 ms
64 bytes from del11s20-in-x04.1e100.net (2404:6800:4002:82b::2004): icmp_seq=3 ttl=57 time=218 ms
64 bytes from del11s20-in-x04.1e100.net (2404:6800:4002:82b::2004): icmp_seq=4 ttl=57 time=143 ms
64 bytes from del11s20-in-x04.1e100.net (2404:6800:4002:82b::2004): icmp_seq=5 ttl=57 time=141 ms
^C
--- www.google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 98.969/159.796/218.180/42.866 ms
```

Traceroute

- Shows how a packet travelled from a local machine to a remote one
- Show the route taken, the IP and hostnames of routers on the network.
- Useful for understanding latency or diagnosing network issues
- Syntax : traceroute [options] host_Address[pathlength]

```
traceroute www.google.com
traceroute to www.google.com (142.250.193.132), 30 hops max, 60 byte packets
 1 _gateway (192.168.136.250) 2.653 ms 3.061 ms 3.305 ms
 2 255.0.0.0 (255.0.0.0) 37.285 ms 38.417 ms 39.253 ms
 3 255.0.0.2 (255.0.0.2) 39.334 ms 42.481 ms 42.850 ms
 4 255.0.0.3 (255.0.0.3) 37.883 ms 48.258 ms 47.896 ms
 5 192.168.225.99 (192.168.225.99) 54.113 ms 192.168.225.98 (192.168.225.98) 52.868 ms 192.168.225.99 (192.168.225.99) 53.712 ms
 6 192.168.83.22 (192.168.83.22) 60.480 ms 192.168.83.28 (192.168.83.28) 58.888 ms 192.168.83.22 (192.168.83.22) 58.907 ms
 7 * * *
 8 * * *
 9 72.14.196.126 (72.14.196.126) 101.899 ms 97.556 ms 98.328 ms
10 * * *
11 108.170.253.97 (108.170.253.97) 87.592 ms 142.251.55.64 (142.251.55.64) 78.461 ms 108.170.253.97 (108.170.253.97) 78.766 ms
12 108.170.253.122 (108.170.253.122) 79.101 ms 55.206 ms 108.170.253.104 (108.170.253.104) 55.533 ms
13 74.125.242.145 (74.125.242.145) 55.471 ms naa05s25-in-f4.1e100.net (142.250.193.132) 55.479 ms 55.104 ms
```

Nslookup

- Name server Look Up
- Finds IP address that corresponds to a host
- Finds the domain name that corresponds to an IP address (“Reverse DNS Lookup”)
- Find the mail servers for the domain
- It provides name server information for the DNS (Domain Name System), i.e. the default DNS server’s name and IP Address.

```
nslookup
> www.google.com
Server:          192.168.136.250
Address:         192.168.136.250#53

Non-authoritative answer:
Name:   www.google.com
Address: 142.250.192.100
Name:   www.google.com
Address: 2404:6800:4007:825::2004
```

Host

host is a simple utility for performing DNS lookups. It is normally used to convert names to IP addresses and vice versa. When no arguments or options are given, host prints a short summary of its command-line arguments and options.

```
host google.com
google.com has address 142.250.77.174
google.com has IPv6 address 2404:6800:4007:818::200e
google.com mail is handled by 10 smtp.google.com.
```

Netstat

- Display routing information by kernel
- The netstat provides the statistics and information in the use of the current TCP-IP connection network about the protocol.

```
netstat
```

Active Internet connections (w/o servers)

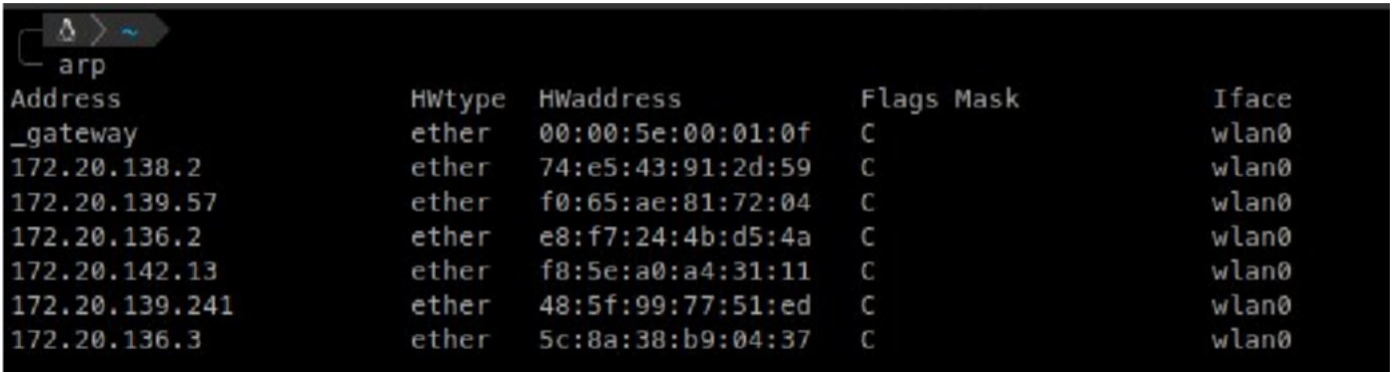
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	Zephyrus-614:6660	88.88.186.16:www-http	ESTABLISHED
tcp	0	0	Zephyrus-614:47064	184.21.89.197:https	ESTABLISHED
tcp	0	0	Zephyrus-614:40648	52.114.133.184:https	ESTABLISHED
tcp	0	0	Zephyrus-614:50984	184.76.94.5:https	ESTABLISHED
tcp	0	0	Zephyrus-614:50734	37.197.208.35:bc:https	ESTABLISHED
tcp	0	0	Zephyrus-614:45000	52.123.253.188:https	ESTABLISHED
tcp	0	0	Zephyrus-614:50938	184.76.94.5:https	ESTABLISHED
tcp	1	1	Zephyrus-614:48832	78.198.146.143:https	LAST_ACK
tcp	0	0	Zephyrus-614:35594	cdn-105-199-186-1:https	ESTABLISHED
tcp	0	0	Zephyrus-614:40698	104-85-139-19:www-http	ESTABLISHED
tcp	0	0	Zephyrus-614:54778	78.198.146.38:https	ESTABLISHED
tcp	1	1	Zephyrus-614:39930	52.160.117.178:https	LAST_ACK
tcp	0	0	Zephyrus-614:53654	cdn-105-199-111-1:https	ESTABLISHED
tcp	0	0	Zephyrus-614:50948	184.76.94.5:https	ESTABLISHED
tcp	0	0	Zephyrus-614:50302	78.198.118.188:https	ESTABLISHED
tcp	0	0	Zephyrus-614:57090	52.114.44.04:https	ESTABLISHED
tcp	0	0	Zephyrus-614:47882	172.66.43.12:https	ESTABLISHED
tcp	1	1	Zephyrus-614:48818	78.198.146.143:https	LAST_ACK
tcp	0	0	Zephyrus-614:54268	78.199.146.38:https	ESTABLISHED
tcp	0	0	Zephyrus-614:43668	48.99.8.226:https	ESTABLISHED
tcp	0	0	Zephyrus-614:47488	184.76.94.176:https	ESTABLISHED
tcp	0	0	Zephyrus-614:44300	52.114.36.207:https	ESTABLISHED
tcp	31	0	Zephyrus-614:50918	184.25.94.5:https	ESTABLISHED
tcp	0	0	Zephyrus-614:50978	184.76.94.5:https	ESTABLISHED
tcp	0	0	Zephyrus-614:55348	172.66.43.43:https	ESTABLISHED
tcp	0	0	Zephyrus-614:40702	104-85-139-19:www-http	ESTABLISHED
tcp	0	0	Zephyrus-614:60138	184.17.33.62:https	ESTABLISHED
udp	0	0	Zephyrus-614:bootps	172.78.72.18:bootps	ESTABLISHED

Active UNIX domain sockets (w/o servers)

Proto	RefCnt	Flags	Type	State	I Node	Path
unix	1	[]	STREAM	CONNECTED	13972	
unix	1	[]	STREAM	CONNECTED	28367	
unix	3	[]	STREAM	CONNECTED	28170	
unix	3	[]	STREAM	CONNECTED	14755	
unix	1	[]	STREAM	CONNECTED	17959	/run/dbus/system_bus_socket
unix	3	[]	STREAM	CONNECTED	27500	
unix	3	[]	STREAM	CONNECTED	27489	/run/user/1000/dolphin1kksd.2.klowner_socket
unix	1	[]	STREAM	CONNECTED	14878	
unix	3	[]	STREAM	CONNECTED	15326	
unix	3	[]	STREAM	CONNECTED	32476	
unix	1	[]	STREAM	CONNECTED	16733	
unix	1	[]	STREAM	CONNECTED	27265	/run/user/1000/bus
unix	3	[]	STREAM	CONNECTED	32632	
unix	3	[]	STREAM	CONNECTED	33724	
unix	1	[]	STREAM	CONNECTED	18878	
unix	3	[]	STREAM	CONNECTED	33540	

arp

- Address Resolution Protocol
- Resolve IP address of the machine to its MAC address



Address	HWtype	HWaddress	Flags	Mask	Iface
_gateway	ether	00:00:5e:00:01:0f	C		wlan0
172.20.138.2	ether	74:e5:43:91:2d:59	C		wlan0
172.20.139.57	ether	f0:65:ae:81:72:04	C		wlan0
172.20.136.2	ether	e8:f7:24:4b:d5:4a	C		wlan0
172.20.142.13	ether	f8:5e:a0:a4:31:11	C		wlan0
172.20.139.241	ether	48:5f:99:77:51:ed	C		wlan0
172.20.136.3	ether	5c:8a:38:b9:04:37	C		wlan0

systeminfo

```
PS C:\Users\Dhruv Hatkar> systeminfo
```

```
Host Name:                ASUS-ZEPHYRUS-G
OS Name:                  Microsoft Windows 11 Home Single Language
OS Version:              10.0.22621 N/A Build 22621
OS Manufacturer:        Microsoft Corporation
OS Configuration:       Standalone Workstation
OS Build Type:            Multiprocessor Free
Registered Owner:        Dhruv Hatkar
Registered Organization:  N/A
Product ID:              00327-35936-68522-AA0EM
Original Install Date:    01-Oct-22, 1:22:12 PM
System Boot Time:        26-Apr-23, 7:45:43 PM
System Manufacturer:     ASUSTeK COMPUTER INC.
System Model:             ROG Zephyrus G14 GA401QH_GA401QH
System Type:              x64-based PC
Processor(s):             1 Processor(s) Installed.
                          [01]: AMD64 Family 25 Model 80 Stepping 0 AuthenticAMD ~3201 Mhz
BIOS Version:            American Megatrends International, LLC. GA401QH.408, 13-Dec-21
Windows Directory:       C:\WINDOWS
System Directory:         C:\WINDOWS\system32
Boot Device:              \Device\HarddiskVolume1
System Locale:             en-us;English (United States)
Input Locale:             en-us;English (United States)
Time Zone:                (UTC+05:30) Chennai, Kolkata, Mumbai, New Delhi
Total Physical Memory:    15,776 MB
Available Physical Memory: 9,418 MB
Virtual Memory: Max Size: 29,088 MB
Virtual Memory: Available: 19,829 MB
Virtual Memory: In Use:   9,259 MB
Page File Location(s):    D:\pagefile.sys
Domain:                   WORKGROUP
Logon Server:             \\ASUS-ZEPHYRUS-G
Hotfix(s):                4 Hotfix(s) Installed.
                          [01]: KB5020880
                          [02]: KB5012170
                          [03]: KB5023706
                          [04]: KB5022948
Network Card(s):          3 NIC(s) Installed.
                          [01]: MediaTek Wi-Fi 6 MT7921 Wireless LAN Card
                              Connection Name: Wi-Fi
                              DHCP Enabled:   Yes
                              DHCP Server:    172.20.72.10
                              IP address(es)
                              [01]: 172.20.139.239
                              [02]: fe80::ca9c:193f:966:ff24
                          [02]: Bluetooth Device (Personal Area Network)
                              Connection Name: Bluetooth Network Connection
                              Status:         Media disconnected
                          [03]: VirtualBox Host-Only Ethernet Adapter
                              Connection Name: Ethernet 5
                              DHCP Enabled:   No
                              IP address(es)
                              [01]: 192.168.56.1
                              [02]: fe80::8c2e:9d48:b07f:7e77
Hyper-V Requirements:     A hypervisor has been detected. Features required for Hyper-V will not be displayed.
```


EX 1(B)

1. Network cables :

- a. They are physical cables used to connect devices to a network.
- b. This allows for data transfer between those devices.

2. Routers:

- a. They are networking devices that forward data between computer networks.
- b. They are typically used to connect devices to the Internet.

3. Repeaters, Hubs, and Switches:

- a. Repeaters regenerate digital signals to extend the range of a network.
- b. Hubs connect devices in a network.
- c. Switches filter and forward data packets between devices in a network.

4. Bridges:

- a. They are networking devices.
- b. They connect two separate networks to form a single network, filtering and forwarding data packets between them.

5. Gateways:

- a. Gateways are networking devices that connect two different types of networks, such as a LAN and a WAN.
- b. This allows for communication between them.

6. Network Interface Cards:

- a. Network Interface Cards (NICs) are hardware components that enable devices to connect to a network.
- b. This is done by providing a physical interface between the device and the network cable.

Name - Dhruv Hatkar

Reg No - 21BAI1218

Lab No - 2

Server :

```

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <unistd.h>
#define CONNECTION_PORT 8080
int main()
{
    int socket_descriptor;
    int client_socket;
    char storage_buffer[80];
    int length_of_address;
    int option_value=1;
    struct sockaddr_in server_address;
    struct sockaddr_in connection_address;
    char* message = "This is a message from the server";
    socket_descriptor = socket(AF_INET, SOCK_STREAM, 0);
    if(socket_descriptor<0)
    {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
    int status=setsockopt(socket_descriptor, SOL_SOCKET, SO_REUSEADDR ,
    &option_value, sizeof(option_value));
    if(status<0){
        perror("Couldn't set options");
        exit(EXIT_FAILURE);
    }
    server_address.sin_family = AF_INET;
    server_address.sin_port    htons(CONNECTION_PORT);
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_zero[8]='\0';
    status=bind(socket_descriptor, (struct sockaddr*)&server_address, sizeof(struct
    sockaddr));
    if(status<0){
        perror("Couldn't bind socket");
        exit(EXIT_FAILURE);
    }
    status=listen(socket_descriptor,4);
    if(status<0){
        perror("Couldn't listen for connections");
        exit(EXIT_FAILURE);
    }

```

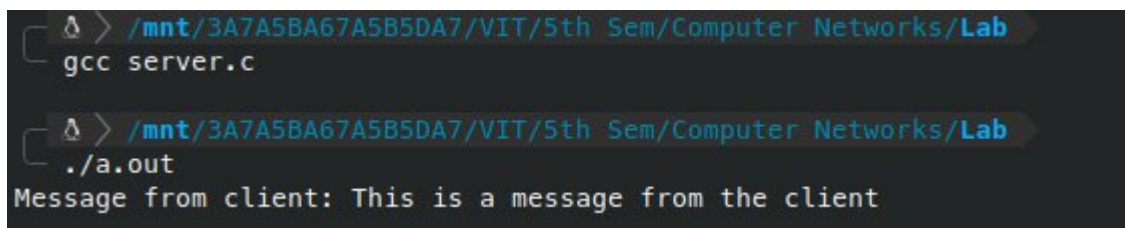
}

```

length_of_address = sizeof(connection_address);
client_socket = accept(socket_descriptor, (struct
sockaddr*)&connection_address,
&length_of_address);
if(client_socket<0){
perror("Couldn't establish connection with client");
exit(EXIT_FAILURE);
}
read(client_socket, storage_buffer, 80);
printf("Message from client: %s \n",storage_buffer);
send(client_socket, message, strlen(message), 0);
close(socket_descriptor);
close(client_socket);
return 0;
}

```

Output -



```

$ gcc server.c
$ ./a.out
Message from client: This is a message from the client

```

Client Side :

```

#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<fcntl.h>
#include <stdio.h>
#include <string.h>
#include<stdlib.h>
#include<arpa/inet.h>
#include<unistd.h>
#define CONNECTION_PORT 8080
int main()
{
    struct sockaddr_in server_address;
    char* message="This is a message from the client";
    char recieve_buffer[100];
    int socket_descriptor = socket(AF_INET, SOCK_STREAM, 0);
    if(socket_descriptor<0)
    {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
    int status=0;
    server_address.sin_family = AF_INET;
    server_address.sin_port    htons(CONNECTION_PORT);
    server_address.sin_addr.s_addr =INADDR_ANY;
    server_address.sin_zero[8]='\0';
    status=connect(socket_descriptor, (struct sockaddr*)&server_address,
    sizeof(server_address));
    if(status<0)
    {
        perror("Couldn't connect with the server");
        exit(EXIT_FAILURE);
    }
    write(socket_descriptor, message, strlen(message));
    read(socket_descriptor, recieve_buffer, 100);
    printf("Message from server: %s\n", recieve_buffer);
    close(socket_descriptor);
    return 0;
}

```

Output -

```
> /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab  
gcc client.c  
  
> /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab  
./a.out  
Message from server: This is a message from the server({}f0
```


Name - Dhruv Hatkar

Reg No - 21BAI1218

Lab No - 3

Q1 :

Server :

```

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MAX_MSG_SIZE 100

void reverseCase(char *str)
{
    int i;
    for (i = 0; str[i] != '\0'; i++) {
        if (islower(str[i]))
            str[i] = toupper(str[i]);
        else if (isupper(str[i]))
            str[i] = tolower(str[i]);
    }
}

int main() {
    int sockfd, clientfd;
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t clientLen;
    char buffer[MAX_MSG_SIZE];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        perror("Socket creation failed");
        exit(1);
    }

    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(8080);
    serverAddr.sin_addr.s_addr = INADDR_ANY;

    if (bind(sockfd, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) == -1)
    {
        perror("Binding failed");
        exit(1);
    }

    if (listen(sockfd, 5) == -1)
    {
        perror("Listening failed");
        exit(1);
    }
}

```

```
printf("Server is listening for incoming connections...\n");
```

```

while (1) {
    clientLen = sizeof(clientAddr);
    clientfd = accept(sockfd, (struct sockaddr *)&clientAddr, &clientLen);
    if (clientfd == -1) {
        perror("Accepting connection failed");
        exit(1);
    }

    char clientIP[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &(clientAddr.sin_addr), clientIP, INET_ADDRSTRLEN);
    printf("Client connected from IP: %s\n", clientIP);

    while (1) {
        memset(buffer, 0, sizeof(buffer));
        if (recv(clientfd, buffer, sizeof(buffer), 0) == -1)
            {perror("Receiving message failed");
            exit(1);
            }

        printf("\nReceived message from client: %s\n", buffer);

        if (strcmp(buffer, "0") == 0)
            break;

        reverseCase(buffer);

        printf("Transmitted message to client: %s\n", buffer);

        if (send(clientfd, buffer, strlen(buffer), 0) == -1) {
            perror("Sending response failed");
            exit(1);
        }
    }

    close(clientfd);
}

close(sockfd);

return 0;
}

```

Output -

```
Δ > /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Lab3  
gcc ServerProgram.c -o server  
  
Δ > /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Lab3  
./server  
Server is listening for incoming connections...  
Client connected from IP: 127.0.0.1  
  
Received message from client: Hello There  
Transmitted message to client: hELLO tHERE  
  
Received message from client: 0
```

Client Side :

```

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define SERVER_IP "127.0.0.1"
#define SERVER_PORT 8080
#define MAX_MSG_SIZE 100

int main() {
    int sockfd;
    struct sockaddr_in serverAddr;
    char message[MAX_MSG_SIZE], response[MAX_MSG_SIZE];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        perror("Socket creation failed");
        exit(1);
    }

    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(SERVER_PORT);
    if (inet_pton(AF_INET, SERVER_IP, &serverAddr.sin_addr) <= 0)
        {perror("Invalid server address");
        exit(1);
    }

    if (connect(sockfd, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) ==
-1) {
        perror("Connection failed");
        exit(1);
    }

    printf("Connected to the server.\n");

    while (1) {
        printf("\nEnter a message (enter '0' to exit): ");
        fgets(message, MAX_MSG_SIZE, stdin);
        message[strcspn(message, "\n")] = '\0';
        if (send(sockfd, message, strlen(message), 0) == -1)
            {perror("Sending message failed");
            exit(1);
        }
    }
}

```

```
if (strcmp(message, "0") == 0)
```



```

        break;

memset(response, 0, sizeof(response));
if (recv(sockfd, response, sizeof(response), 0) == -1)
{
    perror("Receiving response failed");
    exit(1);
}

printf("Response from server: %s\n", response);
}

close(sockfd);

return 0;
}

```

Output -

```

Δ > /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Lab3
gcc ClientProgram.c -o client

Δ > /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Lab3
./client
Connected to the server.

Enter a message (enter '0' to exit): Hello There
Response from server: hELLO tHERE

Enter a message (enter '0' to exit): 0

```

Q2.

Server :

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>

#define PORT 8080

int main() {
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    char response[1024] = {0};

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
        &opt, sizeof(opt))) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr *)&address,
        sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
        (socklen_t *)&addrlen)) < 0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }
}

```

```
valread = read(new_socket, buffer, 1024);
```

```

int marks[5];
int i = 0;
char* token = strtok(buffer, " ");
while (token != NULL) {
    marks[i++] = atoi(token);
    token = strtok(NULL, " ");
}

int sum = 0;
for (int i = 0; i < 5; i++)
    {sum += marks[i];
}
float percentage = (sum * 100) / 500.0;
char grade;
if (percentage >= 90) {
    grade = 'S';
} else if (percentage >= 80 && percentage < 90) {
    grade = 'A';
} else if (percentage >= 70 && percentage < 80) {
    grade = 'B';
} else if (percentage >= 60 && percentage < 70) {
    grade = 'C';
} else if (percentage >= 50 && percentage < 60) {
    grade = 'D';
} else {
    grade = 'E';
}

sprintf(response, "Sum : %d\nGrade: %c\n", sum, grade);
send(new_socket, response, strlen(response), 0);

return 0;
}

```

```

$ /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Lab3
gcc serverq2.c -o server
$ /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Lab3
./server

```

Client :

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 8080

int main(int argc, char const *argv[]) {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};
    char marks[20];

    printf("Enter the marks for 5 subjects separated by spaces: ");
    fgets(marks, 20, stdin);

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {printf("\n Socket creation error \n");
        return -1;
    }

    memset(&serv_addr, '0', sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
        {printf("\nConnection Failed \n");
        return -1;
    }

    send(sock, marks, strlen(marks), 0);

    valread = read(sock, buffer, 1024);
    printf("%s", buffer);

    return 0;
}

```

```
Δ > /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Lab3
./client
Enter the marks for 5 subjects separated by spaces: 9 7 8 9 3 2 5 6
Sum : 36
Grade: E
```

Name - Dhruv Hatkar

Reg No - 21BAI1218

Lab No - 4

Q1 :

Algorithm:

Server:

Define the buffer size and create a socket using the `socket()` function.

Configure the server address using the `sockaddr_in` structure.

Bind the socket to the server address using the `bind()` function.

Start listening for incoming data using the `recvfrom()` function.

Reverse the received data using a for loop.

Send the reversed data back to the client using the `sendto()` function.

Repeat steps 4-6 until the server is stopped.

Close the socket.

Client :

1. Include the necessary header files (`stdio.h`, `stdlib.h`, `string.h`, `sys/socket.h`, `arpa/inet.h`, and `unistd.h`).
2. Define the IP address and port number for the server.
3. Define the buffer size for the messages being sent and received.
4. Create a UDP socket using the `socket()` function.
5. Configure the server address using the `sockaddr_in` structure.
6. Prompt the user to enter a message to send to the server.
7. Use `fgets()` to read the user's input and store it in the buffer.
8. Remove the newline character from the end of the buffer.
9. Use `sendto()` to send the message to the server.
10. Print a message indicating that the message was sent.

11. Use `recvfrom()` to receive a message from the server.
12. Print the received message.

13. Close the socket.
14. Exit the program.

Program

Server :

```

#include <netinet/in.h>
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>
#define BUF_SIZE 1024

int main(int argc, char *argv[]){
    int sock;
    struct sockaddr_in serverAddr, clientAddr;
    char buffer[BUF_SIZE];
    socklen_t addrLen = sizeof(clientAddr);
    if((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1){
        perror("socket");
        exit(1);
    }
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family=AF_INET;
    serverAddr.sin_port=htons(8888);
    serverAddr.sin_addr.s_addr=INADDR_ANY;
    if(bind(sock, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) == -1){perror("bind");
        exit(1);
    }
    printf("Server started. Waiting for data.....\n");
    while(1){ if(recvfrom(sock, buffer, BUF_SIZE, 0,
        (struct
sockaddr*)&clientAddr, &addrLen) == -1){perror("recvfrom");
        exit(1);
    }
        printf("Recieved data from
%s:%d\n", inet_ntoa(clientAddr.sin_addr), ntohs(clientAddr.sin_port));
        int len=strlen(buffer);
        for(int
            i=0; i<len/2; i++){ char
            tmp=buffer[i];
            buffer[i]=buffer[len-i-1];
            buffer[len-i-1]=tmp;
        }
        if(sendto(sock, buffer, strlen(buffer), 0, (struct
sockaddr*)&clientAddr, addrLen) == -1){
            perror("sendto");
            exit(1);
        }
    }
}

```

}

}

```
    return 0;  
}
```

Output -

```
> /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Lab 4  
gcc serverq1.c -o server  
  
> /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Lab 4  
./server  
Server started. Waiting for data....  
Recieved data from 127.0.0.1:53410
```

Client :

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#define SERVER_IP "127.0.0.1"
#define SERVER_PORT 8888
#define BUF_SIZE 1024

int main(int argc, char *argv[]){
    int sock;
    struct sockaddr_in serverAddr;
    char buffer[BUF_SIZE];
    if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
        {perror("socket");
        exit(1);
    }
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(SERVER_PORT);
    serverAddr.sin_addr.s_addr = inet_addr(SERVER_IP);
    printf("Enter a message: ");
    fgets(buffer, BUF_SIZE, stdin);
    buffer[strlen(buffer)-1] = '\0';
    if (sendto(sock, buffer, strlen(buffer), 0, (struct sockaddr*)&serverAddr,
        sizeof(serverAddr)) == -1) {
        perror("sendto");
        exit(1);
    }
    printf("Message sent to server.\n");

    if (recvfrom(sock, buffer, BUF_SIZE, 0, NULL, NULL) == -1)
        {perror("recvfrom");
        exit(1);
    }
    printf("Received message from server: %s\n", buffer);
    close(sock);
    return 0;
}

```

Output -

```
Δ > /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Lab 4
gcc clientq1.c -o client

Δ > /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Lab 4
./client
Enter a message: abcd
Message sent to server.
Received message from server: dcba
```

Q2.

Algorithm :

Create a socket using the `socket()` function.

Initialize the server address structure with the IP address and port number using the `memset()` function.

Bind the socket to the server address using the `bind()` function.

Set up a loop to receive messages from the client using the `recvfrom()` function.

Parse the message to extract the two operands and the operator using `sscanf()` function.

Switch on the operator to perform the arithmetic operation and store the result in a variable.

Convert the result to a string using `sprintf()` function.

Send the result back to the client using the `sendto()` function.

Close the socket using the `close()` function.

Code

♦


```

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>
#define BUF_SIZE 1024

int main(int argc, char*
    argv[]){int sock;
    struct sockaddr_in server_addr, client_addr;
    char buf[BUF_SIZE];
    int len;
    if((sock=socket(AF_INET, SOCK_DGRAM, 0))==-1){
        perror("socket");
        exit(1);
    }
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(1234);
    server_addr.sin_addr.s_addr = INADDR_ANY;
    if (bind(sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) ==
-1) {
        perror("bind");
        exit(1);
    }
    len = sizeof(client_addr);
    while (1) {
        if (recvfrom(sock, buf, BUF_SIZE, 0, (struct sockaddr*)&client_addr,
&len) == -1) {
            perror("recvfrom");
            exit(1);
        }
    }
    int num1, num2, result;
    char op;
    sscanf(buf, "%d %c %d", &num1, &op, &num2);
    switch (op) {
        case '+':
            result = num1 + num2;
            break;
        case '-':
            result = num1 - num2;
            break;
        case '*':
            result = num1 * num2;

```

```
break;
```

```

    case '/':
        result = num1 / num2;
        break;
    case '%':
        result = num1 % num2;
        break;
    default:
        result =

        0; break;
}
sprintf(buf, "%d", result);
if (sendto(sock, buf, strlen(buf), 0, (struct sockaddr*)&client_addr,
len) == -1) {
    perror("sendto");
    exit(1);
}

close(sock);
return 0;
}

```



Name: Dhruv Hatkar

Reg No.-21BAI1218

Subject – Computer Networks

Slot – L43 + L44

Q1.

Algorithm :

1. Initialize an array data of size 20 and an array hammingCode of size 30.
2. Read the number of data bits from the user and store it in a variable dataBits.
3. Initialize all bits in hammingCode to 0.
4. Store the data bits in reverse order in hammingCode.
5. Calculate the number of parity bits required using the formula $2^r \geq m + r + 1$, where r is the number of parity bits and m is the number of data bits.
6. Set the parity bits in hammingCode using the following steps:
 - a. For each parity bit position i, calculate the parity bit value by XORing the data bits at positions that have a 1 in the ith bit position.
 - b. Store the parity bit value in the ith bit position of hammingCode.
7. Print the Hamming code.

Code :

```
#include <stdio.h>
#include <math.h>

int main() {
    int data[20], hammingCode[30];
    int dataBits, hammingCodeLen, i, j, k;

    printf("Enter the number of data bits: ");
    scanf("%d", &dataBits);

    for (i = 0; i < dataBits; i++)
        {hammingCode[i] = 0;
    }
    hammingCodeLen = dataBits;

    printf("Enter the data bits (in binary): ");
    for (i = 0; i < dataBits; i++) {
        scanf("%d", &data[i]);
        hammingCode[dataBits - i - 1] = data[i];
    }

    for (i = 0; i < dataBits; i++) {
```

```
    if (pow(2, i) >= dataBits + i + 1)
        break;
}
```

```

int numParityBits = i;

for (i = 0; i < numParityBits; i++)
{
    int parityBitPos = pow(2, i) - 1;
    int parityBit = 0;

    for (j = parityBitPos; j < hammingCodeLen; j += (parityBitPos + 1) * 2)
    {
        for (k = 0; k <= parityBitPos && j + k < hammingCodeLen; k++) {
            parityBit ^= hammingCode[j + k];
        }
    }


    hammingCode[parityBitPos] = parityBit;
}

printf("Hamming code: ");
for (i = hammingCodeLen - 1; i >= 0; i--)
{
    printf("%d ", hammingCode[i]);
}
printf("\n");

return 0;
}

```

Output :



```

Δ > /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Exp no. 5
g++ hammingcode.c

Δ > /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Exp no. 5
./a.out
Enter the number of data bits: 5
Enter the data bits (in binary): 1
0
1
0
1
Hamming code: 1 1 1 1 1

```

Q2.

Algorithm-

1. Read the data bits and divisor from the user and store them in variables data and divisor.
2. Append zeros to the data equal to the divisor length minus 1.
3. Perform CRC division using the following steps:
 - a. For each bit in the data, if it is 1, XOR the divisor with the data starting from that bit position.
 - b. Print the intermediate steps.
4. Check if the remainder is zero or not.
5. If the remainder is zero, return 1 (no error detected).
6. If the remainder is not zero, return 0 (error detected).

Code-

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int performCRCCheck(char *data, char *divisor)
{
    int dataLen = strlen(data);
    int divisorLen = strlen(divisor);
    int i, j;

    for (i = 0; i < divisorLen - 1; i++)
    {
        data[dataLen + i] = '0';
    }
    data[dataLen + i] = '\0';

    for (i = 0; i < dataLen; i++)
    {
        if (data[i] == '1')
        {
            for (j = 0; j < divisorLen; j++)
            {
                if (data[i + j] == divisor[j])
                {
                    data[i + j] = '0';
                }
            }
            else
            {

```



```
        data[i + j] = '1';  
    }  
}
```

```

    }

    printf("Iteration %d: %s\n", i + 1, data);
}

for (i = 0; i < dataLen; i++)
{
    if (data[i] == '1')
    {
        return 0;
    }
}
return 1;
}

int main()
{
    char data[100];
    char divisor[100];

    printf("Enter the data bits: ");
    scanf("%s", data);

    printf("Enter the divisor: ");
    scanf("%s", divisor);

    printf("\nStep-by-Step Calculation:\n");

    if (performCRCCheck(data, divisor))
    {
        printf("\nNo error detected. Data is correct.\n");
    }
    else
    {
        printf("\nError detected. Data is corrupted.\n");
    }

    return 0;
}

```

Output -

```

> /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Exp no. 5
gcc crc.c

> /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Exp no. 5
./a.out
Enter the data bits: 101010
Enter the divisor: 10

Step-by-Step Calculation:
Iteration 1: 0010100
Iteration 2: 0010100
Iteration 3: 0000100
Iteration 4: 0000100
Iteration 5: 0000000
Iteration 6: 0000000

No error detected. Data is correct.

```

Q3.

Algorithm :

1. Read the data from the user and store it in a variable data.
2. Calculate the size of the data and store it in a variable dataSize.
3. Calculate the sum of all 16-bit chunks in the data using the following steps:
 - a. For each 16-bit chunk in the data, add it to the sum.
 - b. If the sum overflows 16 bits, add the carry to the sum.
4. If there is an odd number of bytes in the data, add the last byte to the sum.
5. Calculate the one's complement of the sum.
6. Return the one's complement as the checksum.

Code :

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

unsigned short calculateChecksum(char *data, int dataSize)
{
    unsigned long sum = 0;
    int i;

    for (i = 0; i < dataSize - 1; i += 2)
    {
        unsigned short chunk = (data[i] << 8) | data[i + 1];
        sum += chunk;
    }
}

```

```
if (sum & 0xFFFF0000)
{
    sum = (sum & 0xFFFF) + 1;
```

```

    }
}

if (dataSize % 2 == 1)
{
    unsigned short lastByte = data[dataSize - 1] << 8;
    sum += lastByte;

    if (sum & 0xFFFF0000)
    {
        sum = (sum & 0xFFFF) + 1;
    }
}

return (unsigned short)(~sum);
}

int main()
{
    char data[100];
    unsigned short checksum;
    int dataSize, i;

    printf("Enter the data: ");
    fgets(data, sizeof(data), stdin);
    dataSize = strlen(data) - 1;
    data[dataSize] = '\0';

    checksum = calculateChecksum(data, dataSize);

    printf("\nStep-by-Step Calculation:\n");
    printf("Data: %s\n", data);

    printf("16-bit Chunks:\n");
    for (i = 0; i < dataSize - 1; i += 2)
    {
        unsigned short chunk = (data[i] << 8) | data[i + 1];
        printf("%04X ", chunk);
    }
    if (dataSize % 2 == 1)
    {
        printf("%02X00 ", data[dataSize - 1]);
    }
    printf("\n");
}

```

```
unsigned long sum = 0;  
for (i = 0; i < dataSize - 1; i += 2)
```

```

{
    unsigned short chunk = (data[i] << 8) | data[i + 1];
    sum += chunk;

    if (sum & 0xFFFF0000)
    {
        sum = (sum & 0xFFFF) + 1;
    }

    printf("Sum: %08IX\n", sum);
}

if (dataSize % 2 == 1)
{
    unsigned short lastByte = data[dataSize - 1] << 8;
    sum += lastByte;

    if (sum & 0xFFFF0000)
    {
        sum = (sum & 0xFFFF) + 1;
    }

    printf("Sum: %08IX\n", sum);
}

unsigned short onesComplement = (unsigned short)(~sum);

printf("\nChecksum (One's Complement): %04X\n", onesComplement);
printf("Final Checksum: %04X\n", checksum);

return 0;
}

```

Output:

```
> /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Exp no. 5  
gcc checksum.c
```

```
> /mnt/3A7A5BA67A5B5DA7/VIT/5th Sem/Computer Networks/Lab/Exp no. 5  
./a.out
```

Enter the data: 101010

Step-by-Step Calculation:

Data: 101010

16-bit Chunks:

3130 3130 3130

Sum: 00003130

Sum: 00006260

Sum: 00009390

Checksum (One's Complement): 6C6F

Final Checksum: 6C6F

Name - Dhruv Hatkar

Reg No - 21BAI1218

Q1.

Code -

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define TIMEOUT 5
int main() {
    int packet, ack, count = 0;
    srand(time(NULL));
    while (count < 5)
    { packet = rand() %
    10;
    printf("Sending packet %d\n", packet);
    sleep(1);
    if (rand() % 5 == 0)
    { printf("Packet lost!\n");
    continue;
    }
    sleep(1);
    ack = rand()

    2;if (ack == 1) {
    printf("ACK received for packet %d\n", packet);
    count++;
    } else {
    printf("ACK not received for packet %d, re-transmitting\n", packet);
    }
    sleep(TIMEOUT);
    }
    return 0;
```

Algorithm -

Sender sends one packet and waits for an ACK from the receiver.

♦

If the sender doesn't receive ACK for the previous sent packet after a certain period of time, the sender times out and retransmits that packet again.

If the ACK is not received, it re-transmits the previous packet again.

To support this feature, the sender keeps a record of each packet it sends.

Result -

```
./a.out
Sending packet 3
ACK received for packet 3
Sending packet 5
ACK received for packet 5
Sending packet 5
Packet lost!
Sending packet 7
ACK received for packet 7
Sending packet 4
ACK not received for packet 4, re-transmitting
Sending packet 3
ACK received for packet 3
Sending packet 6
ACK received for packet 6
```

Q2.

Code -

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define TIMEOUT 5
#define MAX_RETRIES 3
int main() {
int packet, ack, retries = 0;
srand(time(NULL));
while (retries < MAX_RETRIES)
{packet = rand() % 10;
printf("Sending packet %d\n", packet);
sleep(1);
if (rand() % 5 == 0)
{ printf("Packet lost!\n");
retries++;
continue;
}
sleep(1);
ack = rand() %

2;if (ack == 1) {
printf("ACK received for packet %d\n", packet);
retries = 0;
} else {
printf("ACK not received for packet %d, re-transmitting\n", packet);
retries++;
}
sleep(TIMEOUT);
}
printf("Max retries exceeded, giving up\n");
return 0;
}

```

Algorithm -

Sender sends a data frame or packet with sequence number 0.

◆

Receiver receives the data frame and sends an acknowledgment with sequence number 1 (the sequence number of the next expected data frame or packet).

There is only a one-bit sequence number that implies that both sender and receiver have a buffer for one frame or packet only.

If the ACK is not received, the sender re-transmits the packet.

◆

Result -

```
Sending packet 7
Packet lost!
Sending packet 5
ACK received for packet 5
Sending packet 8
ACK received for packet 8
Sending packet 7
ACK not received for packet 7, re-transmitting
Sending packet 8
ACK received for packet 8
Sending packet 4
ACK received for packet 4
Sending packet 4
ACK not received for packet 4, re-transmitting
Sending packet 9
Packet lost!
Sending packet 9
ACK received for packet 9
Sending packet 4
ACK not received for packet 4, re-transmitting
Sending packet 5
ACK not received for packet 5, re-transmitting
Sending packet 4
Packet lost!
Max retries exceeded, giving up
```

Q3.

Code :

```

#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define TIMEOUT 5
#define WINDOW_SIZE 3
int main() {
int packet, ack, next_seq_num = 0, base = 0;
int window[WINDOW_SIZE] = {0};
srand(time(NULL));
while (1) {
// Send packets in the window
for (int i = base; i < base + WINDOW_SIZE; i++)
{if (i < next_seq_num) {
continue; // Packet already sent
}
packet = rand() % 10;
printf("Sending packet %d\n", packet);
window[i % WINDOW_SIZE] = packet;
// Simulate transmission delay
sleep(1);
// Simulate packet loss
if (rand() % 5 == 0)
{ printf("Packet
lost!\n");
} else
{ next_seq_num+
+;
}
}
// Simulate ACK delay
sleep(1);
// Receive ACKs
ack = rand() % 2;
if (ack == 1) {
printf("ACK received for packet %d\n", base);
base++;
} else {
printf("ACK not received, resending packets\n");
}
// Simulate timeout
sleep(TIMEOUT);
}
return 0;
}

```

Algorithm :

Sender sends packets of window size N and the receiver acknowledges all packets whether they were received in order or not.

The receiver maintains a buffer to contain out-of-order packets and sorts them.

The sender selectively re-transmits the lost packet and moves the window forward.

The window size should always be greater than one to implement pipelining.

Result :

```
gcc q3.c && ./a.out
q3.c: In function 'main':
q3.c:20:1: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
 20 | sleep(1);
    | ^~~~~
Sending packet 5
Sending packet 3
Sending packet 1
ACK not received, resending packets
ACK not received, resending packets
ACK not received, resending packets
ACK not received, resending packets
ACK received for packet 0
Sending packet 8
Packet lost!
ACK not received, resending packets
Sending packet 4
ACK not received, resending packets
ACK not received, resending packets
ACK not received, resending packets
ACK not received, resending packets
ACK received for packet 1
Sending packet 1
ACK received for packet 2
Sending packet 1
Packet lost!
ACK received for packet 3
Sending packet 6
Sending packet 0
```

Q4.

Code :


```

#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define TIMEOUT 5
#define WINDOW_SIZE 3
int main() {
int packet, ack, seq_num = 0;
int window[WINDOW_SIZE] = {0};
srand(time(NULL));
while (1) {
for (int i = 0; i < WINDOW_SIZE; i++)
{if (window[i] == 0) {
packet = rand() % 10;
printf("Sending packet %d\n", packet);
window[i] = packet;
seq_num++;
}
}
sleep(1);
ack = rand() % seq_num + 1;
printf("ACK received for packet %d\n", ack);
for (int i = 0; i < WINDOW_SIZE; i++) {
if (window[i] == ack)
{window[i] = 0;
}
}
sleep(TIMEOUT);
}
return 0;
}

```

Algorithm :

Sender sends packets of window size N and the receiver acknowledges all packets whether they were received in order or not.

The receiver maintains a buffer to contain out-of-order packets and sorts them.

The sender selectively re-transmits the lost packet and moves the window forward.

Result :

```
└─ ./a.out
Sending packet 5
Sending packet 0
Sending packet 7
ACK received for packet 1
Sending packet 6
ACK received for packet 3
ACK received for packet 4
ACK received for packet 2
ACK received for packet 2
ACK received for packet 3
ACK received for packet 1
ACK received for packet 4
ACK received for packet 1
ACK received for packet 4
```

Q1]

Write a C program to convert an IP address in classful notation to classless addressing.

Sample Input:

IP address in classful notation: 190.168.0.0

Sample Output:

Classless address: 190.168.0.0/16

ANS]

ALGORITHM:

- Input the IP address in classful notation.
- Determine the class of the IP address by looking at its first octet:
 - o If the first octet is between 1 and 126 (inclusive), it is a Class A address with a default subnet mask of 255.0.0.0. Add /8 to the end of ip
 - o If the first octet is between 128 and 191 (inclusive), it is a Class B address with a default subnet mask of 255.255.0.0. Add /16 to the end of ip
 - o If the first octet is between 192 and 223 (inclusive), it is a Class C address with a default subnet mask of 255.255.255.0. Add /24 to the end of ip
 - o If the first octet is between 224 and 239 (inclusive), it is a Class D address used for multicasting, and no subnetting is possible.
 - o If the first octet is between 240 and 255 (inclusive), it is a Class E address reserved for future use and cannot be assigned to hosts on a network.
- Convert the default subnet mask for the class of the IP address to binary form.
- Calculate the number of bits required to represent the desired subnet mask based on the number of subnets and hosts required.
- Convert the new subnet mask to binary form.
- Combine the IP address and subnet mask to create the classless address in CIDR notation (e.g., 190.168.0.0/16).
- Output the classless address.

CODE:

```
#include<stdio.h>

int main() {

    int first_octet, second_octet, third_octet, fourth_octet;

    printf("Enter IP address in classful notation (e.g. 190.168.0.0): ");

    scanf("%d.%d.%d.%d", &first_octet, &second_octet, &third_octet, &fourth_octet);

    if (first_octet >= 0 && first_octet <= 127) {

        printf("Classless address: %d.%d.%d.%d/8\n", first_octet, second_octet,>

    } else if (first_octet >= 128 && first_octet <= 191) {

        printf("Classless address: %d.%d.%d.%d/16\n", first_octet, second_octet>

    } else if (first_octet >= 192 && first_octet <= 223) {

        printf("Classless address: %d.%d.%d.%d/24\n", first_octet, second_octet>

    } else {

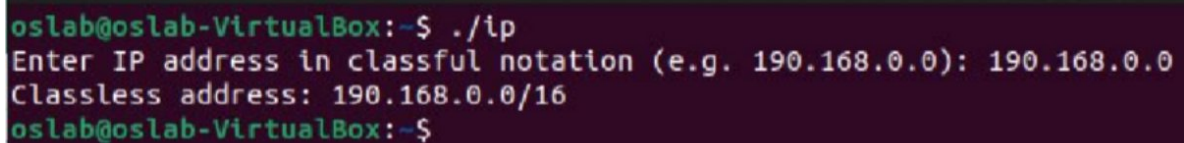
        printf("Invalid IP address.\n");

    }

    return 0;

}
```

OUTPUT:

A terminal window with a dark purple background. The prompt is 'oslab@oslab-VirtualBox:~\$'. The user enters './ip'. The program outputs 'Enter IP address in classful notation (e.g. 190.168.0.0): 190.168.0.0'. Then it outputs 'Classless address: 190.168.0.0/16'. The prompt returns to 'oslab@oslab-VirtualBox:~\$'.

```
oslab@oslab-VirtualBox:~$ ./ip
Enter IP address in classful notation (e.g. 190.168.0.0): 190.168.0.0
Classless address: 190.168.0.0/16
oslab@oslab-VirtualBox:~$
```

Q2]

Write a C program to get an IP address in dotted decimal notation and convert it to binary.

Sample Input:

IP address in dotted decimal notation: 128.11.3.31

Sample Output:

IP address in binary notation: 10000000 00001011 00000011 00011111

ANS]

ALGORITHM:

- Start the program.
- Declare an array of characters to store the IP address in dotted decimal notation.
- Declare an array of integers to store the four parts of the IP address.
- Prompt the user to enter an IP address in dotted decimal notation.
- Read the IP address entered by the user using the scanf() function and store it in the array of characters declared earlier.
- Use the strtok() function to split the IP address into its four parts using the dot (.) as the delimiter, and store each part in the array of integers declared earlier.
- For each part of the IP address:
 - Convert the part from string to integer using the atoi() function.
 - Convert the integer to binary using the strtol() function with a base of 2.
 - Store the resulting binary string in a character array.
 - Print the binary string using the printf() function, with a width specifier of 8 digits and leading zeros.
- End the program.

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int dToB(int n) {

    int binary = 0, base = 1;

    while (n > 0) {

        binary += (n % 2) * base;

        n /= 2;

        base *= 10;

    }

    return binary;

}

int main()

    { char ip[16];

      int parts[4], i;


      // Get IP address from user

      printf("Enter IP address in dotted decimal notation: ");

      scanf("%s", ip);


      // Split IP address into its four parts

      char *token = strtok(ip, ".");

      for (i = 0; i < 4; i++)

          { parts[i] = atoi(token);

            token = strtok(NULL, ".");

          }

    }
```

```
// Convert each part to binary and print
printf("IP address in binary notation: ");
for (i = 0; i < 4; i++) {
    parts[i]=dToB(parts[i]);
    printf("%08d ", parts[i]);
}
printf("\n");

return 0;
}
```

OUTPUT:

```
oslab@oslab-VirtualBox:~$ ./ip
Enter IP address in dotted decimal notation: 128.11.3.31
IP address in binary notation: 10000000 00001011 00000011 00011111
```

Q3]

A block of addresses is granted to a small organization. One of the addresses is known and given as input. Write a C program to compute the first address, last address, and number of addresses in the block.

Sample Input:

Known Address: 205.16.37.39/28

Sample Output:

First Address: 205.16.37.32

Last Address: 205.16.37.47

Number of Addresses: 16

ANS]

ALGORITHM:

1. Start
2. Declare variables ip, a, b, c, d, network_id, host_id, class
3. Read IP address from user
4. Extract octets from IP address
5. Check the value of first octet
6. If first octet is between 1 and 126, set class to A and extract network ID from first octet and host ID from remaining octets
7. If first octet is between 128 and 191, set class to B and extract network ID from first two octets and host ID from remaining octets
8. If first octet is between 192 and 223, set class to C and extract network ID from first three octets and host ID from remaining octet
9. Print the class, network ID, and host ID of the IP address
10. End

CODE:

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int main() {

char ip[18];

int prefix, a, b, c, d;

int mask, network, broadcast, num_addresses;

printf("Enter IP address in CIDR notation: ");

scanf("%s", ip);

sscanf(ip, "%d.%d.%d.%d/%d", &a, &b, &c, &d, &prefix);

mask = 0xffffffff << (32 - prefix);

network = (a << 24) | (b << 16) | (c << 8) | d;

network &= mask;

broadcast = network | ~mask;

num_addresses = broadcast - network + 1;

printf("First Address: %d.%d.%d.%d\n", (network >> 24) & 0xff, (network >> 16) &
0xff, (network >> 8) & 0xff, network & 0xff);

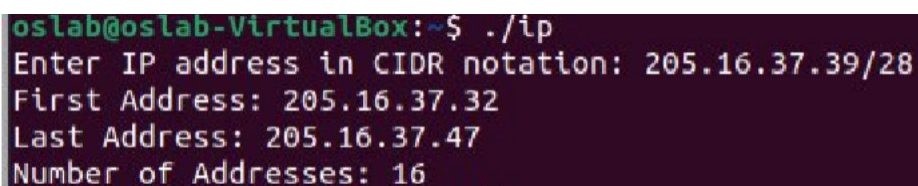
printf("Last Address: %d.%d.%d.%d\n", (broadcast >> 24) & 0xff, (broadcast >> 16)
& 0xff, (broadcast >> 8) & 0xff, broadcast & 0xff);

printf("Number of Addresses: %d\n", num_addresses);

return 0;

}
```

OUTPUT:

A terminal window with a dark background and light-colored text. The prompt is 'oslab@oslab-VirtualBox:~\$'. The user enters './ip'. The program outputs: 'Enter IP address in CIDR notation: 205.16.37.39/28', 'First Address: 205.16.37.32', 'Last Address: 205.16.37.47', and 'Number of Addresses: 16'.

```
oslab@oslab-VirtualBox:~$ ./ip
Enter IP address in CIDR notation: 205.16.37.39/28
First Address: 205.16.37.32
Last Address: 205.16.37.47
Number of Addresses: 16
```

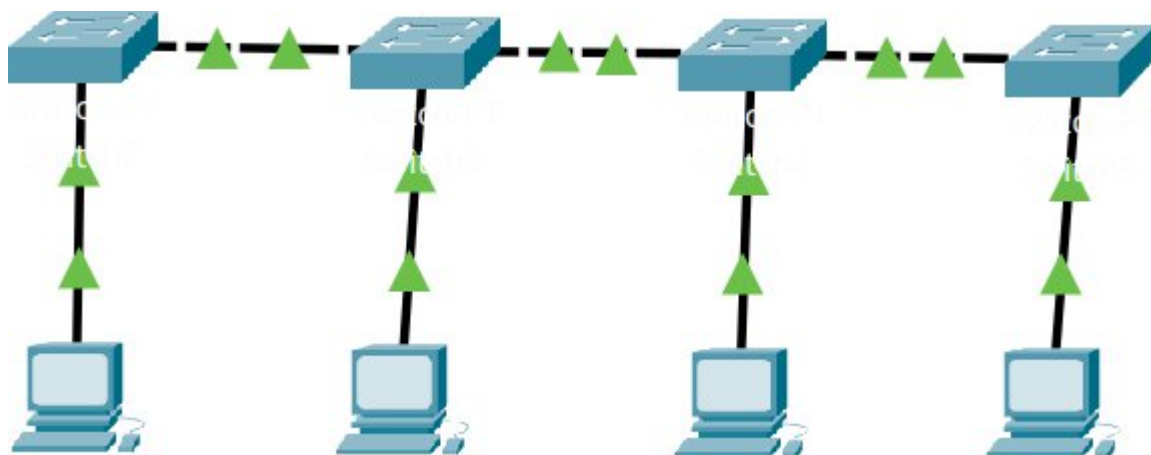
Name : Dhruv Hatkar

Reg No : 21BAI1218

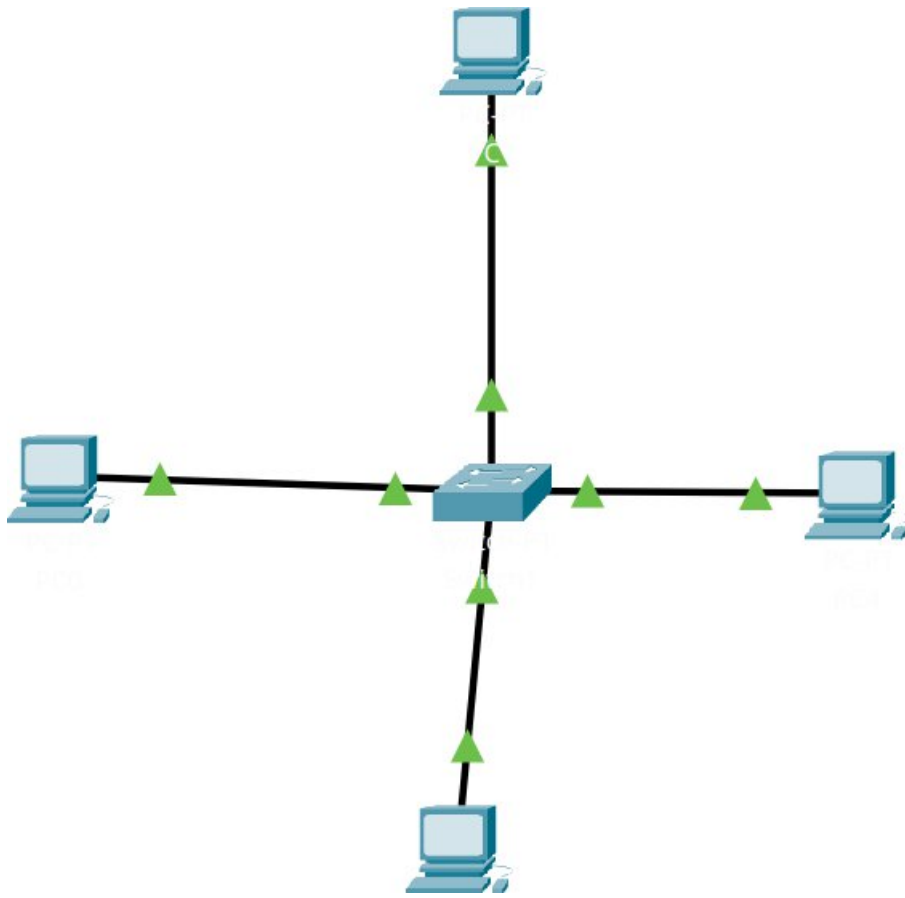
Exp No. : 8

Q1.

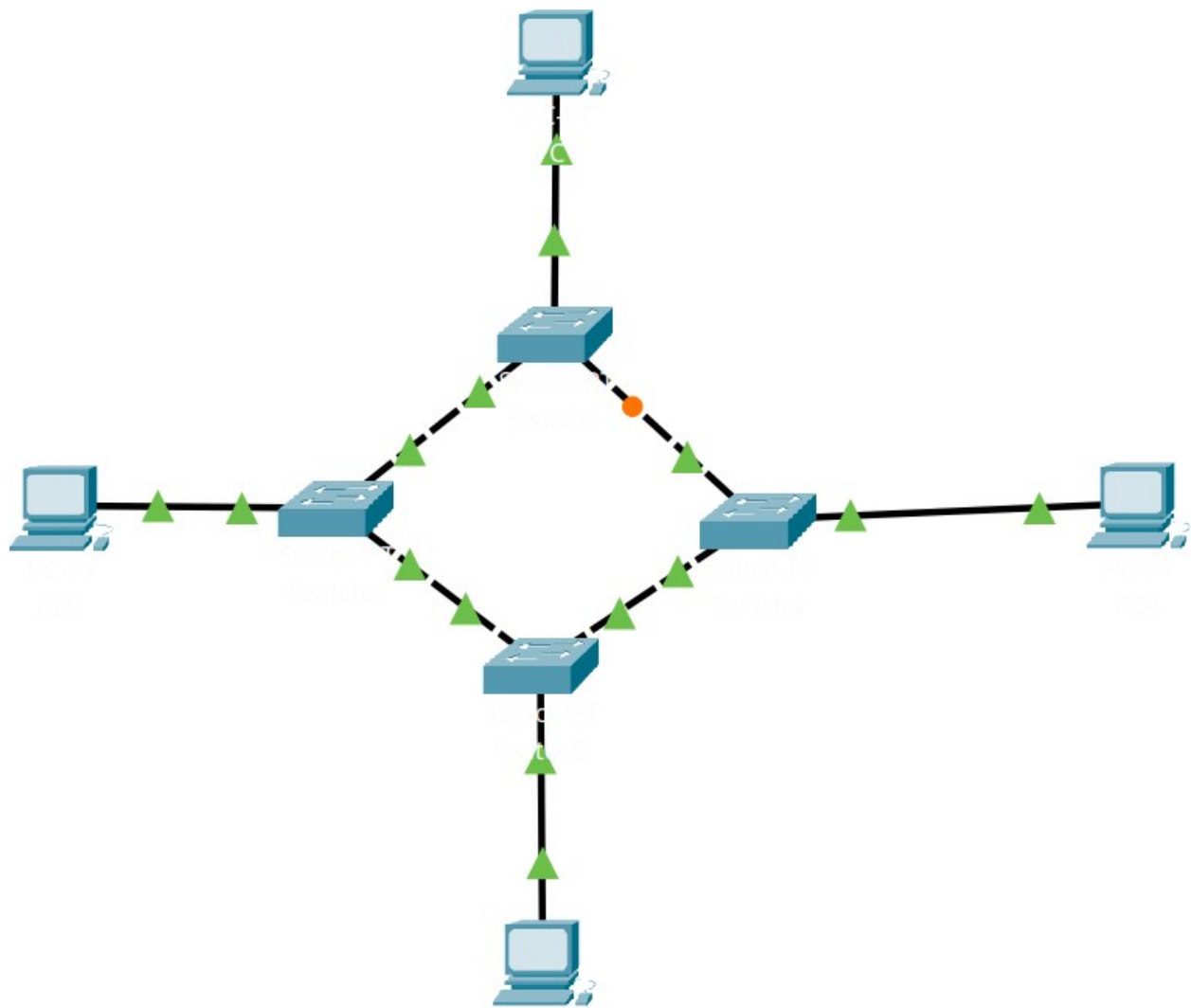
a) Bus



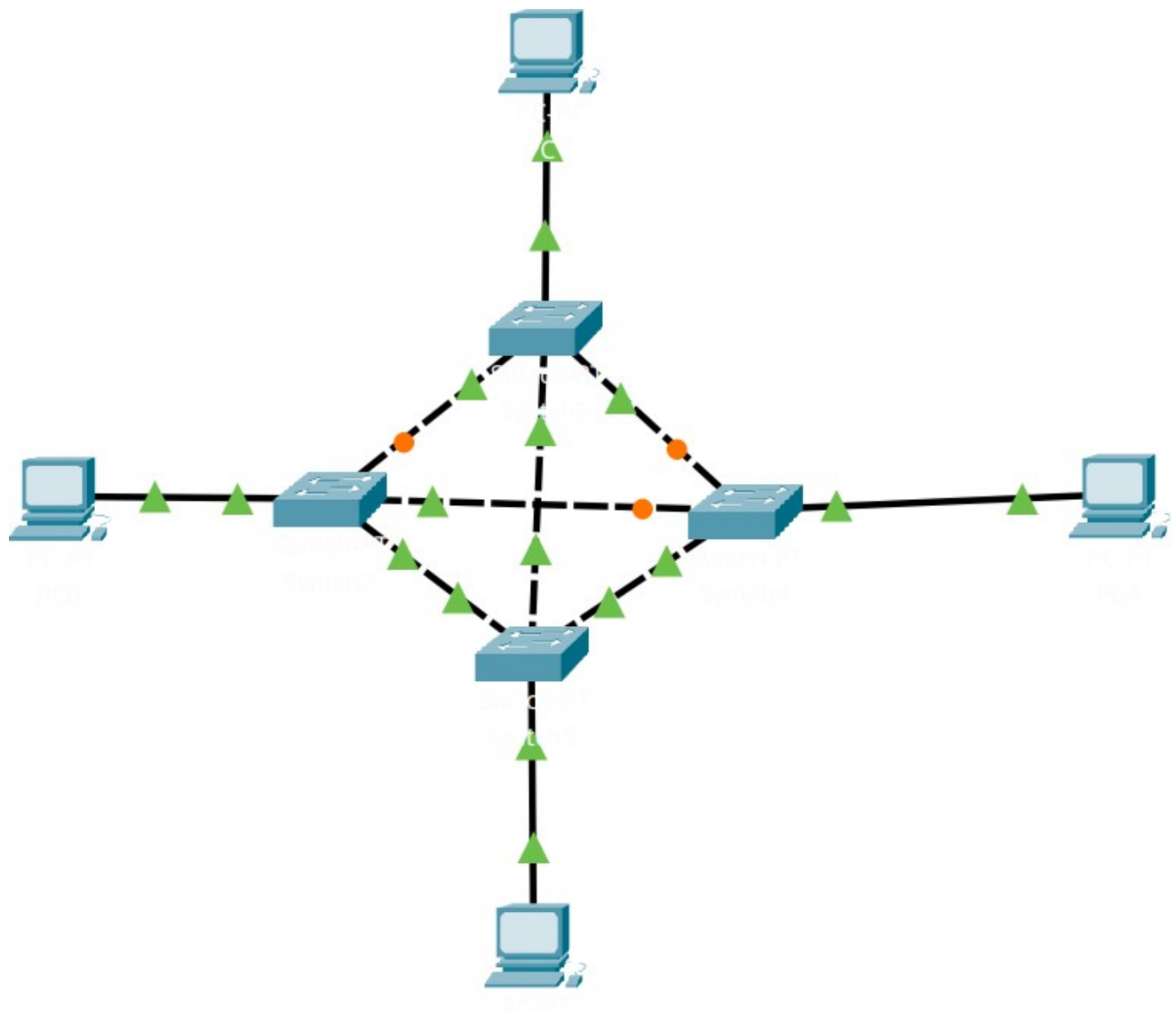
b) Star



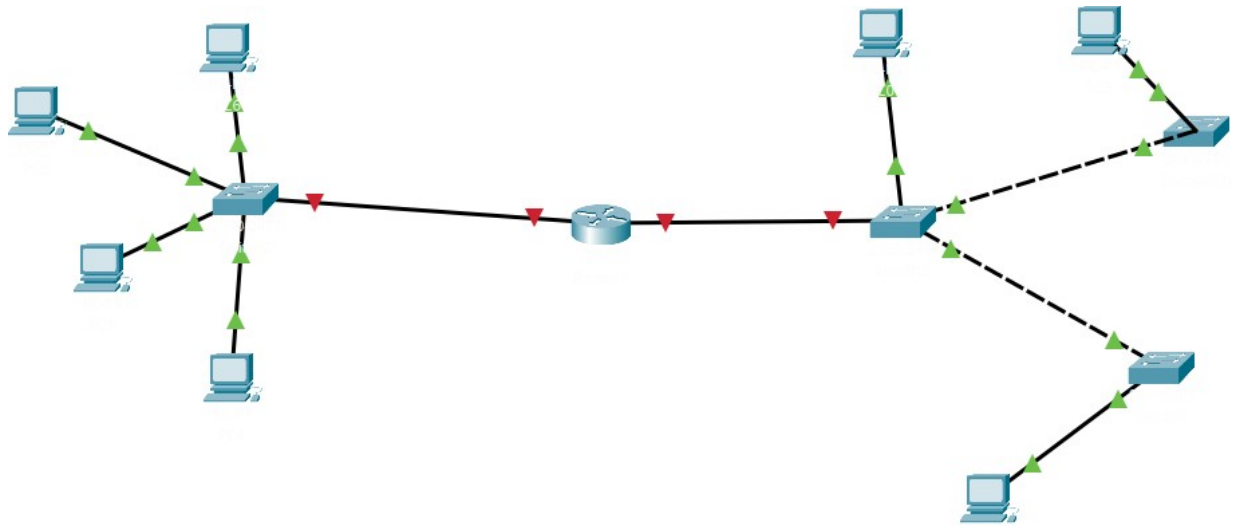
c) Ring



d) Mesh



Q2.





Name: Dhruv Hatkar

Reg No.-21BAI1218

Subject – Computer Networks

Exp. No. - 9

Algorithm:

Step 1: Draw the Topology diagram as mentioned in above diagram.

Step 2: Assigned IP address on Each PC with gateway IP.

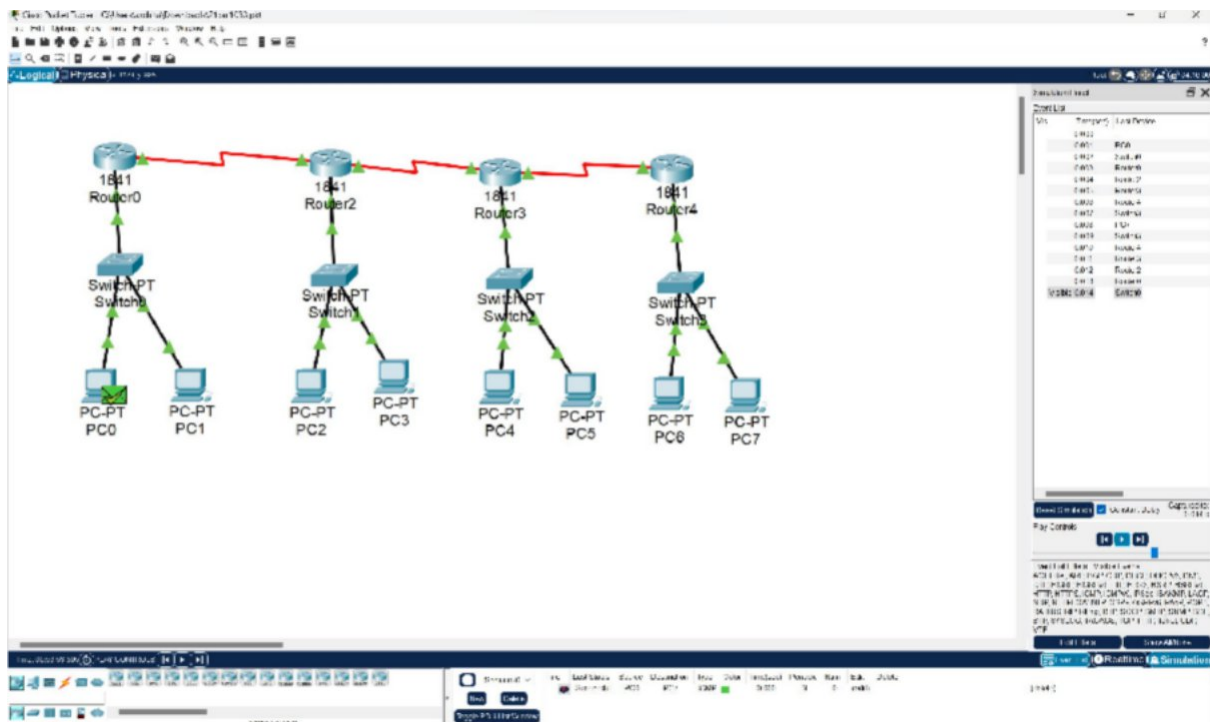
Step 3: Assign Ip Address on each Router One by One

Step 4: Now it's Time to configure Static Route over all those Four Routers R1,R2,R3 and R4

Step 5: At the End test the communication between two PC to test whether your Routing is properly configured or not

Step 6: First Check the IP Address of PC

Step 7: Then Ping with another PC





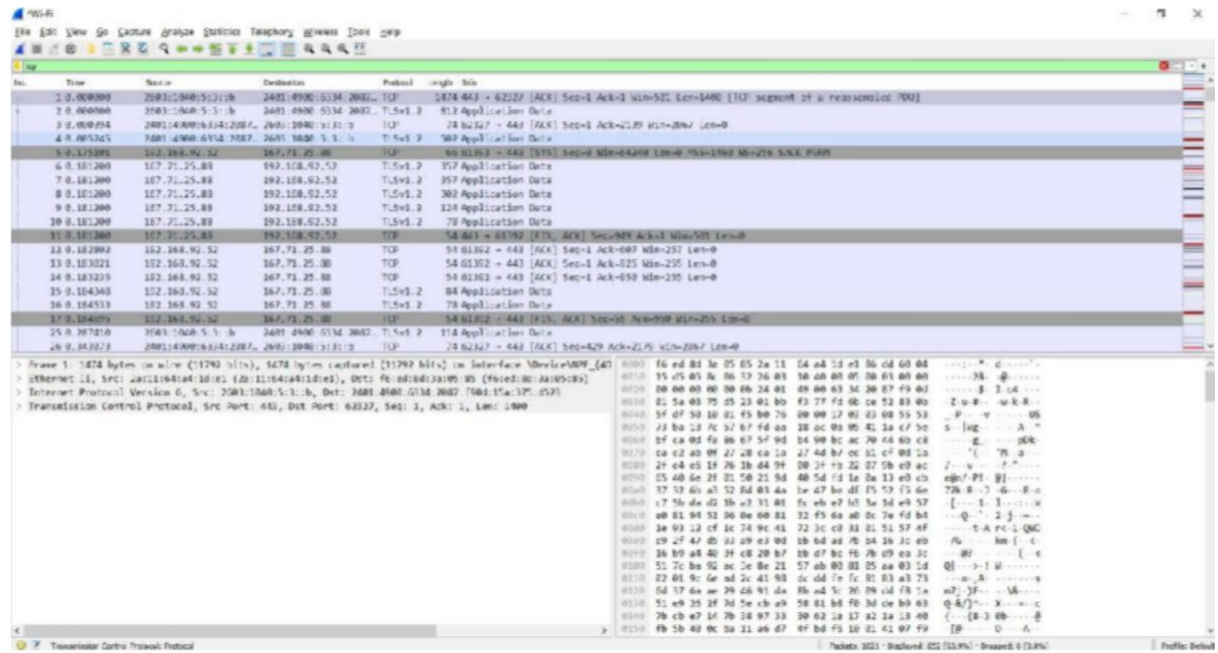
Name: Dhruv Hatkar

Reg No.-21BAI1218

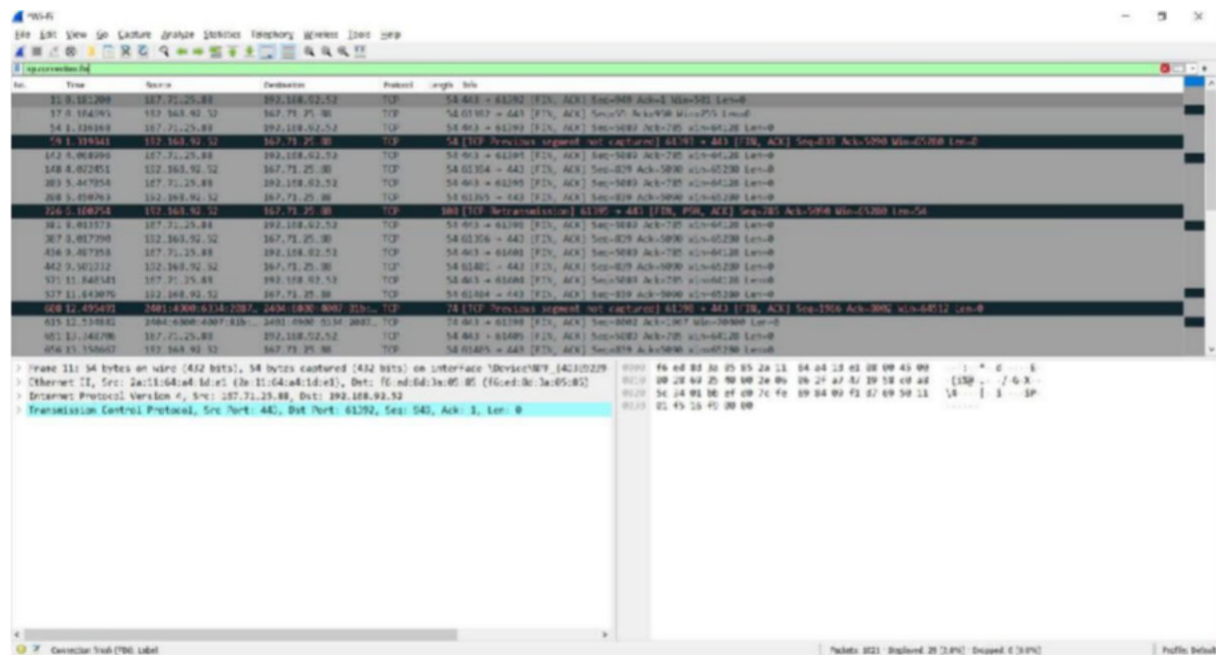
Subject – Computer Networks

Exp. No. - 10

1)

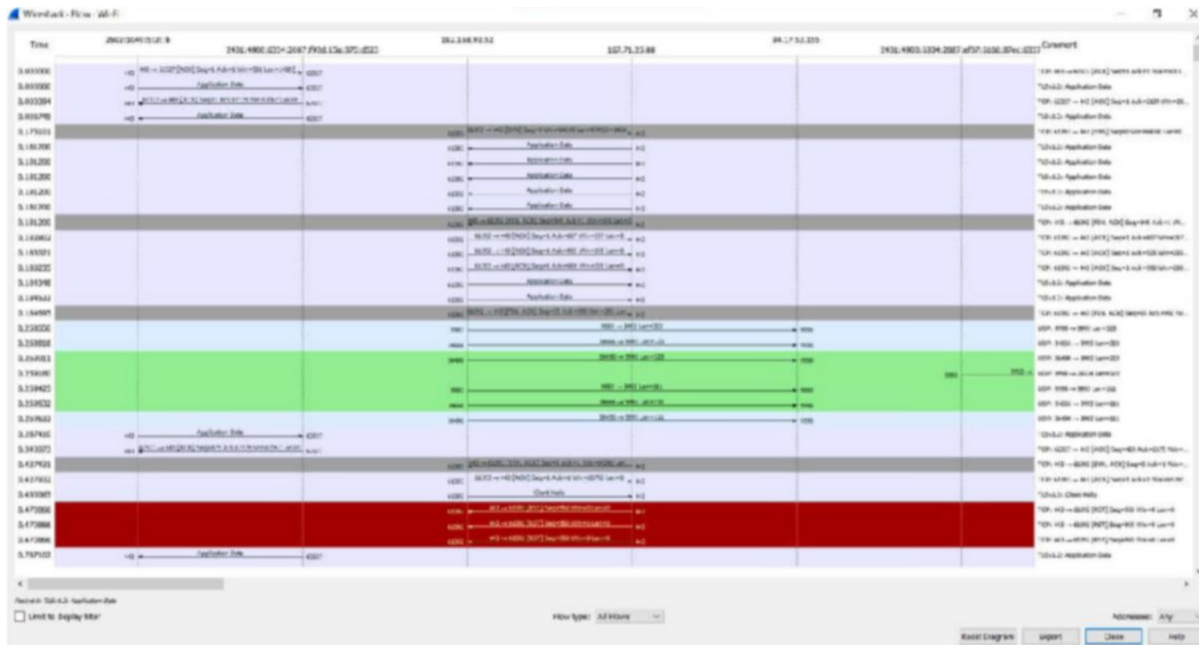


2)

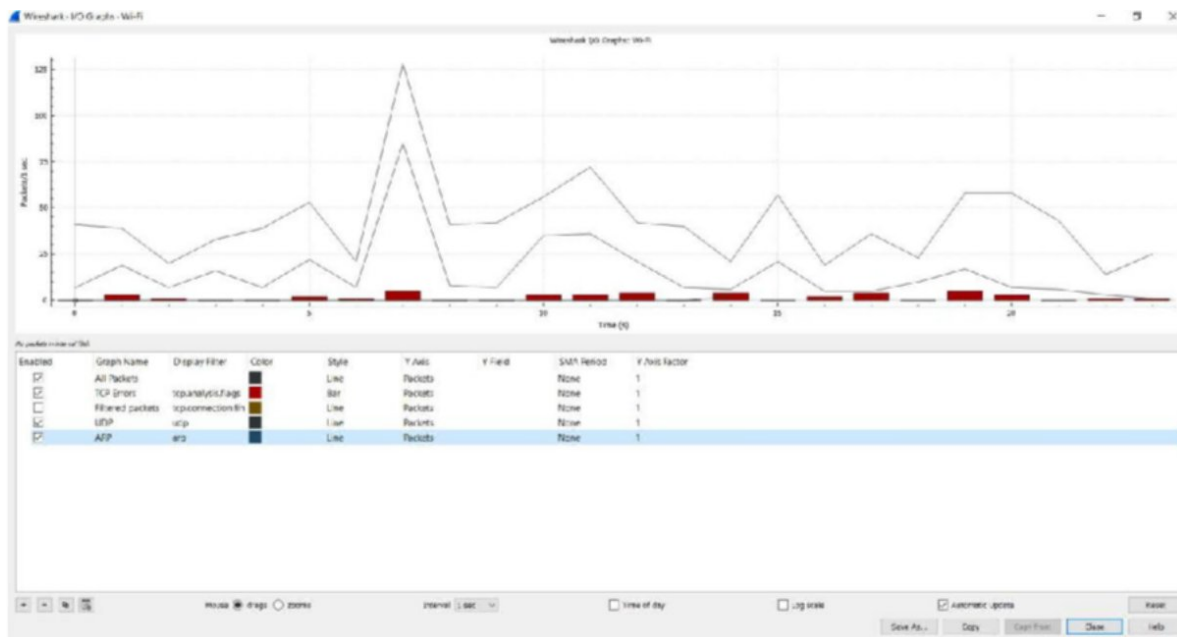


3)

Flow Graph



I/O Graph





Name: Dhruv Hatkar

Reg No.-21BAI1218

Subject – Computer Networks

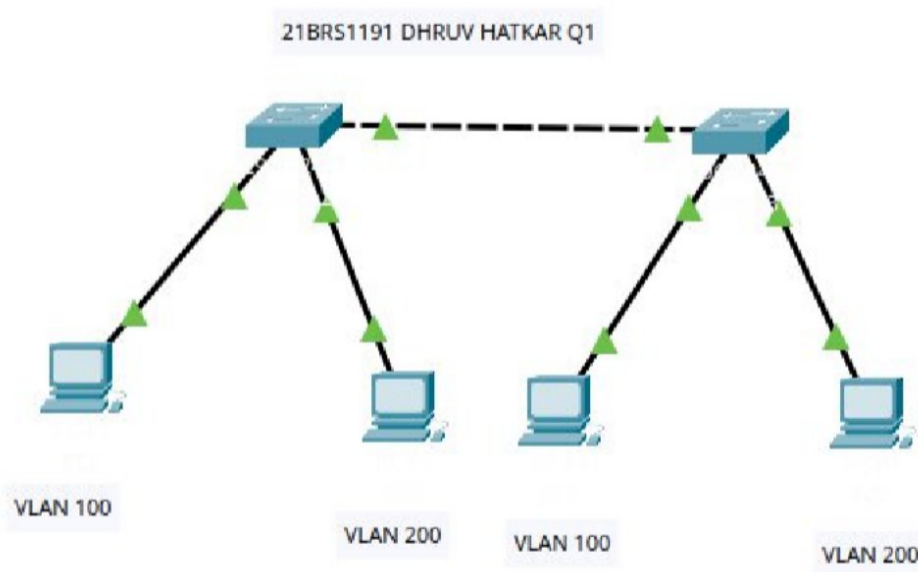
Slot – L43 + L44

Q1.

ALGORITHM

Open Cisco Packet Tracer and drag two switches from the bottom left corner of the window to the workspace.
Connect the switches using a copper straight-through cable.
Click on each switch and select "CLI" from the bottom left corner of the window.
Type "enable" and press enter.
Type "configure terminal" and press enter.
Type "vlan 100" and press enter.
Type "name faculty" and press enter.
Type "exit" and press enter.
Type "vlan 200" and press enter.
Type "name students" and press enter.
Type "exit" and press enter.
Assign ports to VLANs by typing "interface fastethernet 0/x" (where x is the port number) followed by "switchport mode access" and "switchport access vlan y" (where y is the VLAN number).

OUTPUT



Q2.

ALGORITHM

Open Cisco Packet Tracer and drag three switches from the bottom left corner of the window to the workspace.

Connect the switches using copper straight-through cables as shown in Figure 2 of the page.

Click on each switch and select “CLI” from the bottom left corner of the window.

Type “enable” and press enter.

Type “configure terminal” and press enter.

Type “vlan 10” and press enter.

Type “name VLAN1” and press enter.

Type “exit” and press enter.

Type “vlan 20” and press enter.

Type “name VLAN2” and press enter.

Type “exit” and press enter.

Type “vlan 30” and press enter.

Type “name VLAN3” and press enter.

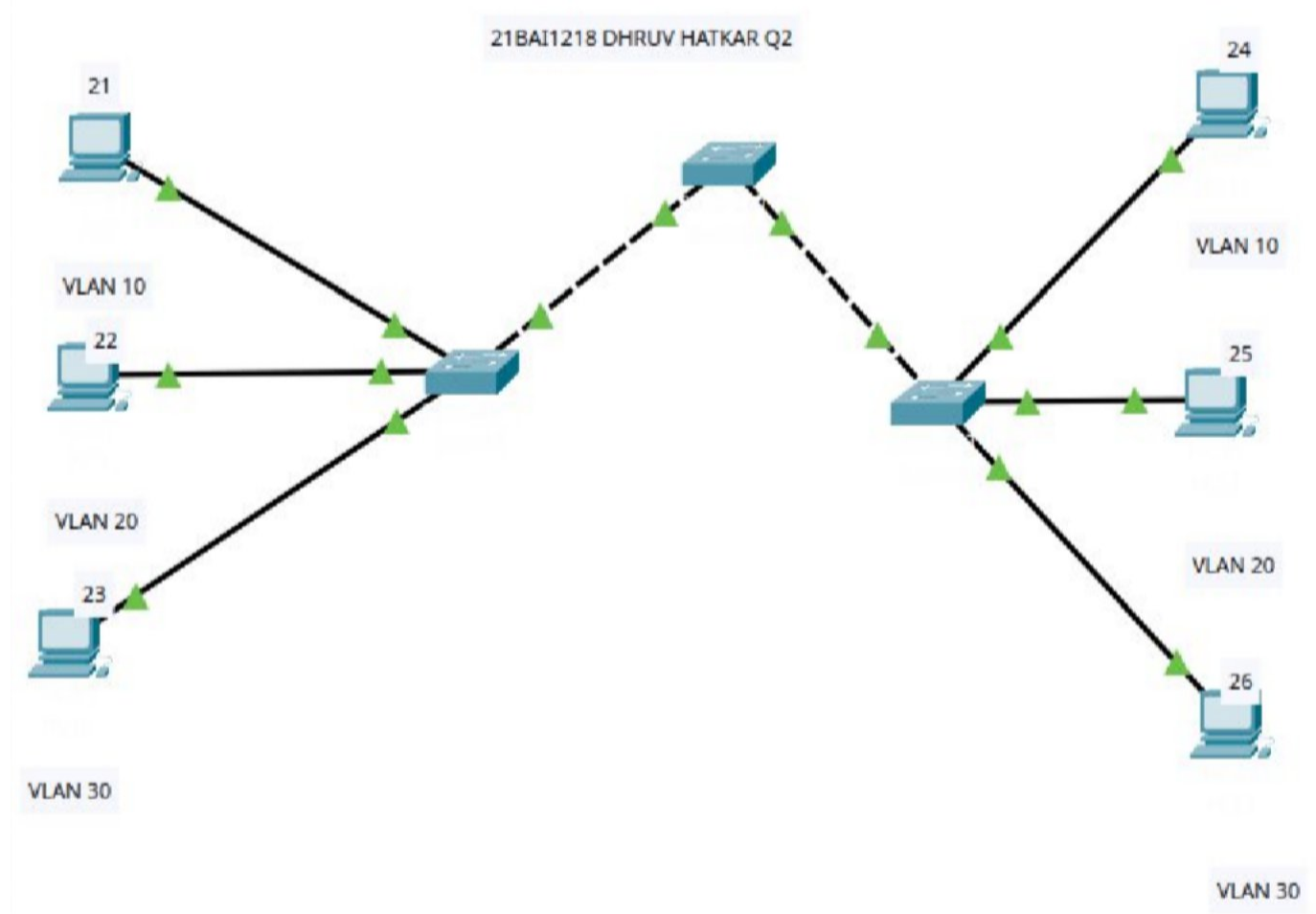
Assign ports to VLANs by typing “interface fastethernet 0/x” (where x is the port number) followed

by “switchport mode access” and “switchport access vlan y” (where y is the VLAN number).

Make sure that the connections between switches is set to trunk.

The pcs are connected as access points to separate vlans.

OUTPUT





Name: Dhruv Hatkar

Reg No.-21BAI1218

Subject – Computer Networks

Exp. No. - 12

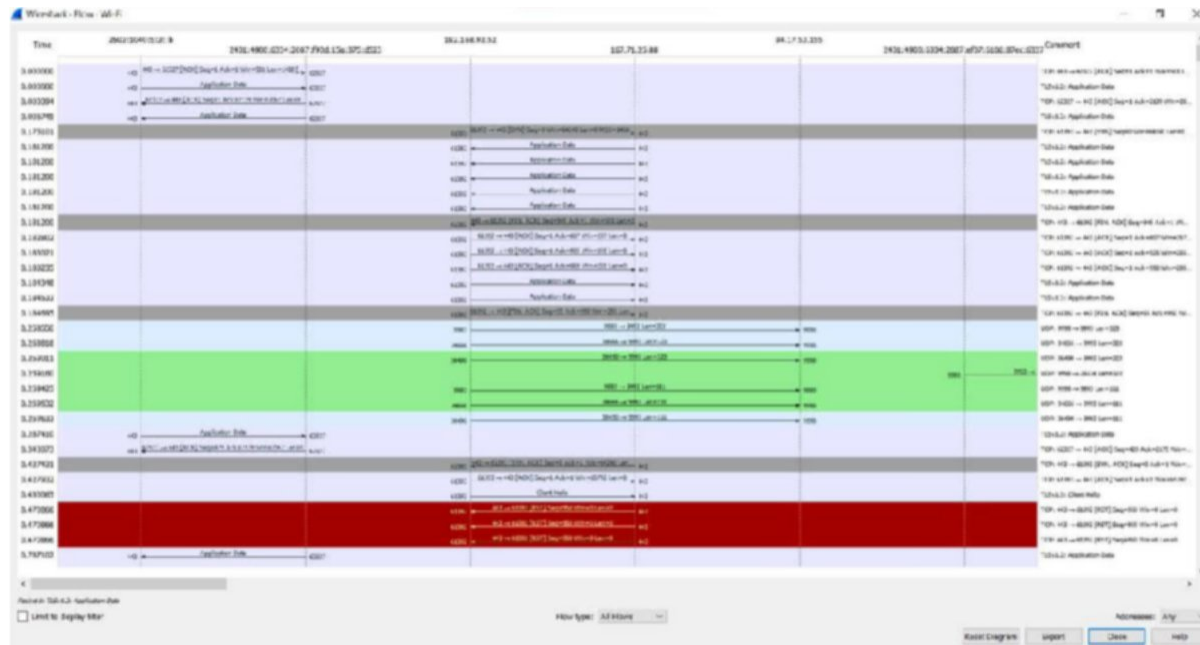
[illegible]

Wireshark packet capture showing a TCP Reset (RST) from 10.0.2.15 to 10.0.2.1. The packet is a response to a previous connection attempt. The 'Reset Sequence' field is highlighted in blue.

No.	Time	Source	Destination	Protocol	Length	Info
51	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
52	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
53	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
54	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
55	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
56	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
57	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
58	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
59	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
60	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
61	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
62	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
63	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
64	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
65	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
66	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
67	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
68	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
69	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
70	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
71	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
72	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
73	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
74	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
75	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
76	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
77	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
78	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
79	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
80	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
81	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
82	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
83	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
84	0.000000	10.0.2.1	10.0.2.15	TCP	54	40102 → 4010 [FIN, ACK] Seq=500 Ack=500 Win=0 Len=0
85	0.000000	10.0.2.15	10.0.2.1	TCP	54	4010 → 40102 [FIN, ACK] Seq=500 Ack=500 Win=0

3)

Flow Graph



I/O Graph

