A Lab Report
on

Computer Networks

Submitted By
Ashish Kumar Jha
Roll Number- 115CS0603

(Submitted to Prof. Suchismita Chinara)

Department of Computer Science and Engineering
ROURKELA - 769008
2018-2019

ASSIGNMENT 1

Date: 27/7/18

A1.1 Finding out what networking devices are installed in the department.

Ans.

Network Switch:

It is used to connect multiple network host and to transfer data packet. It filters the packet and sends only to the interface of the intended filter.

Modem:

It stands for (Modulator Demodulator). It modulates and demodulates the signal between the digital data of a computer and the analog signal of a telephone line.

Network Router:

A network router is responsible for routing traffic from one to another router.

Bridge:

A bridge connects two sub networks as a part of the same network.

Repeater:

It amplifies the signal it receives.

A1.2 Describe the network type and topology of the department.

Ans.

Our department most probably has a hybrid topology. It is two different types of topologies which is a mixture of two or more topologies. For example if in an office in one department ring topology is used and in another star topology is used, connecting these topologies will result in Hybrid Topology (ring topology and star topology).

A1.3 File and printer sharing in different OSs.

Ans.

File sharing is the public or private sharing of computer data or space in a network with various levels of access privilege.

File sharing in windows:

1. Start->control panel->
2. Network and home group->change advanced sharing settings
3. Turn on network discovery and file and printer sharing
4. Save changes

A1.4 Network address configuration in different OSs.

Ans.

1. Start->control panel
2. Change adapter settings
3. Click on the Internet Protocol Version 4 (TCP/IPv4) (you may need to scroll down to find it). Next, click on the Properties button.
4. Next, click the Use the following DNS server addresses: radio button. Next, in the Preferred DNS server:, and Alternate DNS server: num**ber fields, input the numbers that were assigned by OIT. Then click the OK button.**

A1.5 Finding the IP and MAC address in different OSs.

Ans.

1. In windows type command: ipconfig
2. In linux type command: ifconfig

A1.6 Work group and domain group configuration.
Ans.

A workgroup is a peer-to-peer network using Microsoft software. A workgroup allows all participating and connected systems to access shared resources such as files, system resources and printers.

Steps to configure workgroup in windows:

1. Navigate to Control Panel, System and Security and System to access your computer details.
2. Find Workgroup and select Change settings.
3. Select Change next to 'To rename this computer or change its domain…'.
4. Type in the name of the Workgroup you want to join and click OK.
5. Reboot your computer for the changes to take effect.
6. Navigate to Control Panel, Network and Internet and View network computers and devices to see other machines within that Workgroup.

Homegroup is a workgroup secured with password.

Steps to configure homegroup in windows:

1. To connect a second or third computer to the homegroup, go to the first computer's control panel, then click HomeGroup:
2. Click the View or print the homegroup password to view the password. The password will appear. You may opt to print it and distribute to other people connected to your homegroup.
3. One your other Windows computers, go to Control Panel > HomeGroup and then click Join Now.

A1.6 use of the utilities: arp, ipconfig, tracert, nslookup.
Ans.

Arp:

The Address Resolution Protocol (ARP) is a communication protocol used for discovering the link layer address, such as a MAC address, associated with a given network layer address, typically an IPv4 address.

Ipconfig:

In computing, ipconfig (Internet Protocol Configuration) in Microsoft Windows is a console application that displays all current TCP/IP network configuration values and can modify Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) settings.

Tracert:

In computing, traceroute is a computer network diagnostic tool for displaying the route (path) and measuring transit delays of packets across an Internet Protocol (IP) network.

Nslookup:

nslookup is a network administration command-line tool available for many computer operating systems for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or for any other specific DNS record.

ASSIGNMENT 2

DATE: 3/8/18

A2.1 Examine packet flow across a network segment and see the operation of various internet protocols across the different layers in TCP/IP stack.

Ans.

TCP/IP functionality is divided into four layers, each of which include specific protocols

The application layer provides applications with standardized data exchange. Its protocols include the Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), Post Office Protocol 3 (POP3), Simple Mail Transfer Protocol (SMTP) and Simple Network Management Protocol (SNMP).

The transport layer is responsible for maintaining end-to-end communications across the network. TCP handles communications between hosts and provides flow control, multiplexing and reliability. The transport protocols include TCP and User Datagram Protocol (UDP), which is sometimes used instead of TCP for special purposes.

·*The network layer*, also called the internet layer, deals with packets and connects independent networks to transport the packets across network boundaries. The network layer protocols are the IP and the Internet Control Message Protocol (ICMP), which is used for error reporting.

The physical layer consists of protocols that operate only on a link -- the network component that interconnects nodes or hosts in the network. The protocols in this layer include Ethernet for local area networks (LANs) and the Address Resolution Protocol (ARP).

ASSIGNMENT 3

DATE: 10/8/18

A3.1 Use unix sockets to implement a sample client and server communication over the network.

Ans.

<u>Client.c</u>

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdbool.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define KRED  "\x1B[31m"
#define KGRN  "\x1B[32m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT  "\x1B[37m"
#define RESET "\033[0m"

typedef enum
{
        CONNECT,
        DISCONNECT,
   GET_USERS,
        SET_USERNAME,
        PUBLIC_MESSAGE,
        PRIVATE_MESSAGE,
        TOO_FULL,
        USERNAME_ERROR,
        SUCCESS,
        ERROR
} message_type;

typedef struct
{
        message_type type;
        char username[21];
        char data[256];

} message;

typedef struct connection_info
{
        int socket;
```

```
	struct sockaddr_in address;
	char username[20];
} connection_info;

void
trim_newline (char *text)
{
	int len = strlen (text) - 1;
	if (text[len] == '\n')
	{
	text[len] = '\0';
	}
}

void
get_username (char *username)
{
	while (true)
	{
	printf ("Enter a username: ");
	fflush (stdout);
	memset (username, 0, 1000);
	fgets (username, 22, stdin);
	trim_newline (username);

	if (strlen (username) > 20)
	{

		puts ("Username must be 20 characters or less.");

	}
	else
	{
		break;
	}
	}
}

void
set_username (connection_info * connection)
{
	message msg;
	msg.type = SET_USERNAME;
	strncpy (msg.username, connection->username, 20);

	if (send (connection->socket, (void *) &msg, sizeof (msg), 0) < 0)
	{
	perror ("Send failed");
	exit (1);
```

```
        }
}

void
stop_client (connection_info * connection)
{
        close (connection->socket);
        exit (0);
}

void
connect_to_server (connection_info * connection, char *address, char *port)
{

        while (true)
        {
        get_username (connection->username);

        if ((connection->socket = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP)) <
0)
        {
                perror ("Could not create socket");
        }

        connection->address.sin_addr.s_addr = inet_addr (address);
        connection->address.sin_family = AF_INET;
        connection->address.sin_port = htons (atoi (port));

        if (connect (connection->socket, (struct sockaddr *) &connection->address, sizeof
(connection->address)) < 0)
        {
                perror ("Connect failed.");
                exit (1);
        }

        set_username (connection);

        message msg;
        ssize_t recv_val = recv (connection->socket, &msg, sizeof (message), 0);
        if (recv_val < 0)
        {
          perror ("recv failed");
                exit (1);

        }
        else if (recv_val == 0)
        {
                close (connection->socket);
```

```
                printf ("The username \"%s\" is taken, please try another name.\n",
connection->username);
                continue;
        }
        break;
        }

        puts ("Connected to server.");
        puts ("Type /help for usage.");
}

void
handle_user_input (connection_info * connection)
{
        char input[255];
        fgets (input, 255, stdin);
        trim_newline (input);

        if (strcmp (input, "/q") == 0 || strcmp (input, "/quit") == 0)
        {
        stop_client (connection);
        }
        else if (strcmp (input, "/l") == 0 || strcmp (input, "/list") == 0)
        {
        message msg;
        msg.type = GET_USERS;

        if (send (connection->socket, &msg, sizeof (message), 0) < 0)
        {
                perror ("Send failed");
                exit (1);
        }
        }
        else if (strcmp (input, "/h") == 0 || strcmp (input, "/help") == 0)
        {
        puts ("/quit or /q: Exit the program.");
        puts ("/help or /h: Displays help information.");
        puts ("/list or /l: Displays list of users in chatroom.");
        puts ("@<username> <message> Send a private message to given username.");
        }
        else if (strncmp (input, "@", 1) == 0)
        {
        message msg;
        msg.type = PRIVATE_MESSAGE;

        char *toUsername, *chatMsg;

        toUsername = strtok (input + 1, " ");
```

```
        if (toUsername == NULL)
        {
                puts (KRED "The format for private messages is: @<username>
<message>" RESET);
                return;
        }

        if (strlen (toUsername) == 0)
        {
                puts (KRED "You must enter a username for a private message." RESET);
                return;
        }

        if (strlen (toUsername) > 20)
        {
                puts (KRED "The username must be between 1 and 20 characters."
RESET);
                return;
        }

        chatMsg = strtok (NULL, "");

        if (chatMsg == NULL)
        {
                puts (KRED "You must enter a message to send to the specified user."
RESET);
                return;
        }

        strncpy (msg.username, toUsername, 20);
        strncpy (msg.data, chatMsg, 255);

        if (send (connection->socket, &msg, sizeof (message), 0) < 0)
         {
                perror ("Send failed");
                exit (1);
        }

        }
        else
        {
        message msg;
        msg.type = PUBLIC_MESSAGE;
        strncpy (msg.username, connection->username, 20);

        if (strlen (input) == 0)
        {
                return;
        }
```

```
        strncpy (msg.data, input, 255);

        if (send (connection->socket, &msg, sizeof (message), 0) < 0)
        {
                perror ("Send failed");
           exit (1);
        }
        }

}

void
handle_server_message (connection_info * connection)
{
        message msg;

        ssize_t recv_val = recv (connection->socket, &msg, sizeof (message), 0);
        if (recv_val < 0)
        {
        perror ("recv failed");
        exit (1);

        }
        else if (recv_val == 0)
        {
        close (connection->socket);
        puts ("Server disconnected.");
        exit (0);
        }

        switch (msg.type)
        {

        case CONNECT:
        printf (KYEL "%s has connected." RESET "\n", msg.username);
        break;

        case DISCONNECT:
        printf (KYEL "%s has disconnected." RESET "\n", msg.username);
        break;

        case GET_USERS:
        printf (KMAG "%s" RESET "\n", msg.data);
       break;

        case PUBLIC_MESSAGE:
        printf (KGRN "%s" RESET ": %s\n", msg.username, msg.data);
        break;
```

```
        case PRIVATE_MESSAGE:
        printf (KWHT "From %s:" KCYN " %s\n" RESET, msg.username, msg.data);
        break;

        case TOO_FULL:
        fprintf (stderr, KRED "Server chatroom is too full to accept new clients." RESET
"\n");
        exit (0);
        break;

        default:
        fprintf (stderr, KRED "Unknown message type received." RESET "\n");
           break;
        }
}

int
main (int argc, char *argv[])
{
        connection_info connection;
        fd_set file_descriptors;

        if (argc != 3)
        {
        fprintf (stderr, "Usage: %s <IP> <port>\n", argv[0]);
        exit (1);
        }

        connect_to_server (&connection, argv[1], argv[2]);

        while (true)
        {
        FD_ZERO (&file_descriptors);
        FD_SET (STDIN_FILENO, &file_descriptors);
        FD_SET (connection.socket, &file_descriptors);
        fflush (stdin);

             if (select (connection.socket + 1, &file_descriptors, NULL, NULL, NULL) < 0)
        {
                perror ("Select failed.");
                exit (1);
        }

        if (FD_ISSET (STDIN_FILENO, &file_descriptors))
        {
            handle_user_input (&connection);
        }
```

```
	if (FD_ISSET (connection.socket, &file_descriptors))
	{
		handle_server_message (&connection);
	}
	}

	close (connection.socket);
	return 0;
}
```

Server.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdbool.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>

#define MAX_CLIENTS 10

typedef enum
{
	CONNECT,
	DISCONNECT,
	GET_USERS,
	SET_USERNAME,
	PUBLIC_MESSAGE,
	PRIVATE_MESSAGE,
	TOO_FULL,
	USERNAME_ERROR,
	SUCCESS,
	ERROR
} message_type;

typedef struct
{
	message_type type;
	char username[21];
```

```
	char data[256];

} message;

typedef struct connection_info
{
	int socket;
	struct sockaddr_in address;
	char username[20];
} connection_info;

void
trim_newline (char *text)
{
	int len = strlen (text) - 1;
	if (text[len] == '\n')
	{
	text[len] = '\0';
	}
}

void
initialize_server (connection_info * server_info, int port)
{
	if ((server_info->socket = socket (AF_INET, SOCK_STREAM, 0)) < 0)
	{
	perror ("Failed to create socket");
	exit (1);
	}

	server_info->address.sin_family = AF_INET;
	server_info->address.sin_addr.s_addr = INADDR_ANY;
	server_info->address.sin_port = htons (port);

	if (bind (server_info->socket, (struct sockaddr *) &server_info->address,
sizeof (server_info->address)) < 0)
	{
	perror ("Binding failed");
	exit (1);
	}
```

```
	const int optVal = 1;
	const socklen_t optLen = sizeof (optVal);
	if (setsockopt (server_info->socket, SOL_SOCKET, SO_REUSEADDR,
(void *) &optVal, optLen) < 0)
	{
	perror ("Set socket option failed");
	exit (1);
	}

	if (listen (server_info->socket, 3) < 0)
	{
	perror ("Listen failed");
	exit (1);
	}

	printf ("Waiting for incoming connections...\n");
}

void
send_public_message (connection_info clients[], int sender, char
*message_text)
{
	message msg;
	msg.type = PUBLIC_MESSAGE;
	strncpy (msg.username, clients[sender].username, 20);
	strncpy (msg.data, message_text, 256);
	int i = 0;
	for (i = 0; i < MAX_CLIENTS; i++)
	{
	if (i != sender && clients[i].socket != 0)
	{
		if (send (clients[i].socket, &msg, sizeof (msg), 0) < 0)
		{
		    perror ("Send failed");
		exit (1);
		}
	}
	}
}

void
```

```
send_private_message (connection_info clients[], int sender, char
*username, char *message_text)
{
	message msg;
	msg.type = PRIVATE_MESSAGE;
	strncpy (msg.username, clients[sender].username, 20);
	strncpy (msg.data, message_text, 256);

	int i;
	for (i = 0; i < MAX_CLIENTS; i++)
	{
	if (i != sender && clients[i].socket != 0 && strcmp (clients[i].username,
username) == 0)
	{
		if (send (clients[i].socket, &msg, sizeof (msg), 0) < 0)
		{
		perror ("Send failed");
		exit (1);
		}
		return;
		}
	}

	msg.type = USERNAME_ERROR;
	sprintf (msg.data, "Username \"%s\" does not exist or is not logged in.",
username);

	if (send (clients[sender].socket, &msg, sizeof (msg), 0) < 0)
	{
	perror ("Send failed");
	exit (1);
	}

}

void
send_connect_message (connection_info * clients, int sender)
{
	message msg;
	msg.type = CONNECT;
	strncpy (msg.username, clients[sender].username, 21);
```

```
        int i = 0;
        for (i = 0; i < MAX_CLIENTS; i++)
          {
        if (clients[i].socket != 0)
        {
                if (i == sender)
                {
                msg.type = SUCCESS;
                if (send (clients[i].socket, &msg, sizeof (msg), 0) < 0)
                        {
                        perror ("Send failed");
                        exit (1);
                        }
                }
                else
                {
                if (send (clients[i].socket, &msg, sizeof (msg), 0) < 0)
                        {
                        perror ("Send failed");
                        exit (1);
                        }
                }
        }
        }
}

void
send_disconnect_message (connection_info * clients, char *username)
{
        message msg;
        msg.type = DISCONNECT;
        strncpy (msg.username, username, 21);
        int i = 0;
        for (i = 0; i < MAX_CLIENTS; i++)
        {
        if (clients[i].socket != 0)
        {
                if (send (clients[i].socket, &msg, sizeof (msg), 0) < 0)
                {
                perror ("Send failed");
                exit (1);
```

```
                }
        }
        }
}

void
send_user_list (connection_info * clients, int receiver)
{
        message msg;
        msg.type = GET_USERS;
        char *list = msg.data;

        int i;
        for (i = 0; i < MAX_CLIENTS; i++)
        {
        if (clients[i].socket != 0)
        {
                list = stpcpy (list, clients[i].username);
                list = stpcpy (list, "\n");
        }
        }

        if (send (clients[receiver].socket, &msg, sizeof (msg), 0) < 0)
        {
        perror ("Send failed");
        exit (1);
        }

}

void
send_too_full_message (int socket)
{
        message too_full_message;
        too_full_message.type = TOO_FULL;

        if (send (socket, &too_full_message, sizeof (too_full_message), 0) < 0)
        {
        perror ("Send failed");
        exit (1);
        }
```

```
        close (socket);
}

void
stop_server (connection_info connection[])
{
        int i;
        for (i = 0; i < MAX_CLIENTS; i++)
        {
        close (connection[i].socket);
        }
        exit (0);
}

void
handle_client_message (connection_info clients[], int sender)
{
        int read_size;
        message msg;

        if ((read_size = recv (clients[sender].socket, &msg, sizeof (message),
0)) == 0)
        {
        printf ("User disconnected: %s.\n", clients[sender].username);
        close (clients[sender].socket);
        clients[sender].socket = 0;
        send_disconnect_message (clients, clients[sender].username);

        }
        else
        {

        switch (msg.type)
        {
        case GET_USERS:
                send_user_list (clients, sender);
                break;

        case SET_USERNAME:;
                int i;
```

```
                for (i = 0; i < MAX_CLIENTS; i++)
                {
                if (clients[i].socket != 0 && strcmp (clients[i].username,
msg.username) == 0)
                        {
                        close (clients[sender].socket);
                        clients[sender].socket = 0;
                        return;
                        }
                }

                strcpy (clients[sender].username, msg.username);
                printf ("User connected: %s\n", clients[sender].username);
                send_connect_message (clients, sender);
                break;

        case PUBLIC_MESSAGE:
                send_public_message (clients, sender, msg.data);
                break;

        case PRIVATE_MESSAGE:
                send_private_message (clients, sender, msg.username,
msg.data);
                break;

        default:
                fprintf (stderr, "Unknown message type received.\n");
                break;
        }
        }
}

int
construct_fd_set (fd_set * set, connection_info * server_info, connection_info
clients[])
{
        FD_ZERO (set);
        FD_SET (STDIN_FILENO, set);
        FD_SET (server_info->socket, set);

        int max_fd = server_info->socket;
```

```
	int i;
	for (i = 0; i < MAX_CLIENTS; i++)
	{
	if (clients[i].socket > 0)
	{
		FD_SET (clients[i].socket, set);
		if (clients[i].socket > max_fd)
		{
		max_fd = clients[i].socket;
		}
	}
	}
	return max_fd;
}

void
handle_new_connection (connection_info * server_info, connection_info
clients[])
{
	int new_socket;
	int address_len;
	new_socket = accept (server_info->socket, (struct sockaddr *)
&server_info->address, (socklen_t *) & address_len);

	if (new_socket < 0)
	{
	perror ("Accept Failed");
	exit (1);
	}

	int i;
	for (i = 0; i < MAX_CLIENTS; i++)
	{
	if (clients[i].socket == 0)
	 {
		clients[i].socket = new_socket;
		break;

	}
	else if (i == MAX_CLIENTS - 1)
	{
```

```
                send_too_full_message (new_socket);
        }
        }
}

void
handle_user_input (connection_info clients[])
{
        char input[255];
        fgets (input, sizeof (input), stdin);
        trim_newline (input);

        if (input[0] == 'q')
        {
        stop_server (clients);
        }
}

int
main (int argc, char *argv[])
{
        puts ("Starting server.");

        fd_set file_descriptors;

        connection_info server_info;
        connection_info clients[MAX_CLIENTS];

        int i;
        for (i = 0; i < MAX_CLIENTS; i++)
        {
        clients[i].socket = 0;
        }

        if (argc != 2)
        {
        fprintf (stderr, "Usage: %s <port>\n", argv[0]);
        exit (1);
        }

        initialize_server (&server_info, atoi (argv[1]));
```

```
        while (true)
        {
        int max_fd = construct_fd_set (&file_descriptors, &server_info, clients);

        if (select (max_fd + 1, &file_descriptors, NULL, NULL, NULL) < 0)
        {
                perror ("Select Failed");
                stop_server (clients);
        }

        if (FD_ISSET (STDIN_FILENO, &file_descriptors))
        {
                handle_user_input (clients);
        }

        if (FD_ISSET (server_info.socket, &file_descriptors))
        {
                handle_new_connection (&server_info, clients);
        }

        for (i = 0; i < MAX_CLIENTS; i++)
        {
              if (clients[i].socket > 0 && FD_ISSET (clients[i].socket,
&file_descriptors))
                {
                handle_client_message (clients, i);
                }
        }
        }

        return 0;
}
```

A3.2 Write a program to implement daytime server that responds with day and time to request sent by client.

Ans.

```
#include "./unp.h"
#include<stdio.h>

int main(int argc,char **argv)
{
		//client part
		int sockfd,n;
		char recvline[MAXLINE+1];
		struct sockaddr_in servaddr;

		if(argc !=2){printf("Testing");}
			//err_quit("usage: a.out <IPaddress>");
		if(sockfd=socket(AF_INET,SOCK_STREAM,0))<0){}
			//err_sys("socket error")
		bzero(&servaddr,sizeof(servaddr))
		//same as memset set to zero

		servaddr.sin_family=AF_INET;
		servaddr.sin_port=htons(13); /*port no 13 is resereved for daytime server*/
		//converts port address into binary format
		if(inet_pton(AF_INET,argv[1],&servaddr.sin_addr)<=0){}
		//pton : presentation to network conversion of 127.0.0.1 passed as string to
numeric format
		//err_quit("inet_pton error for %s",argv[1]);

		if(connect(sockfd,(SA *) &servaddr,sizeof(servaddr))<0){}
			//err_sys("connect error");
		while((n=read(sockfd,recvline,MAXLINE))>0){
		recvline[n]=0;
		if(fputs(recvline,stdout)==EDF){}
			//err_sys("fputs error")
		}
		if(n<0)
		{
			//err_sys("read error");
		}
		exit(0);
}
```

A3.3 Implement a TCP client server application to transfer a file.

Ans.

filetransferserver.c

```
#include<stdio.h>
#include<sys/types.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>
#define SA struct sockaddr
#define LISTENQ 5
int main(int argc,char**argv)
{
int fd,sockfd,listenfd,connfd;
pid_t childpid;
socklen_t client;
struct sockaddr_in servaddr,cliaddr;
listenfd=socket(AF_INET,SOCK_STREAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(atoi(argv[1]));
bind(listenfd,(SA*)&servaddr,sizeof(servaddr));
listen(listenfd,LISTENQ);
client=sizeof(cliaddr);
connfd=accept(listenfd,(SA*)&cliaddr,&client);
char buffer[100];
FILE *fp;
read(connfd,buffer,100);
fp=fopen("add1.txt","w");
fprintf(fp,"%s",buffer);
printf("the file was received successfully");
printf("the new file created is add1.txt");
}
```

filetransferclient.c

```
#include<arpa/inet.h>
#include<unistd.h>
#define SA struct sockaddr
int main(int argc,char**argv)
{
int sockfd;
```

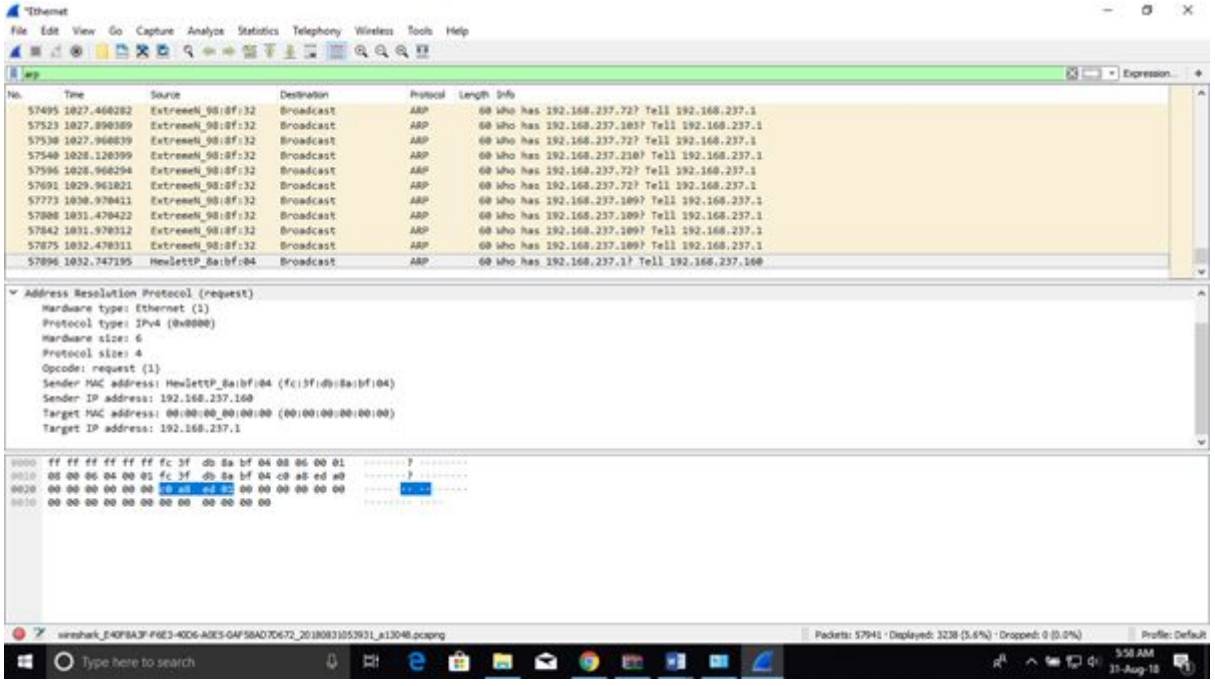
```
char fname[25];
int len;
struct sockaddr_in servaddr,cliaddr;
sockfd=socket(AF_INET,SOCK_STREAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(atoi(argv[1]));
inet_pton(AF_INET,argv[1],&servaddr.sin_addr);
connect(sockfd,(SA*)&servaddr,sizeof(servaddr));
char buffer[100];
FILE *f;
f=fopen("add.txt","r");
fscanf(f,"%s",buffer);
write(sockfd,buffer,100);
printf("the file was sent successfully");
}
```

ASSIGNMENT 4

Date: 31/8/18

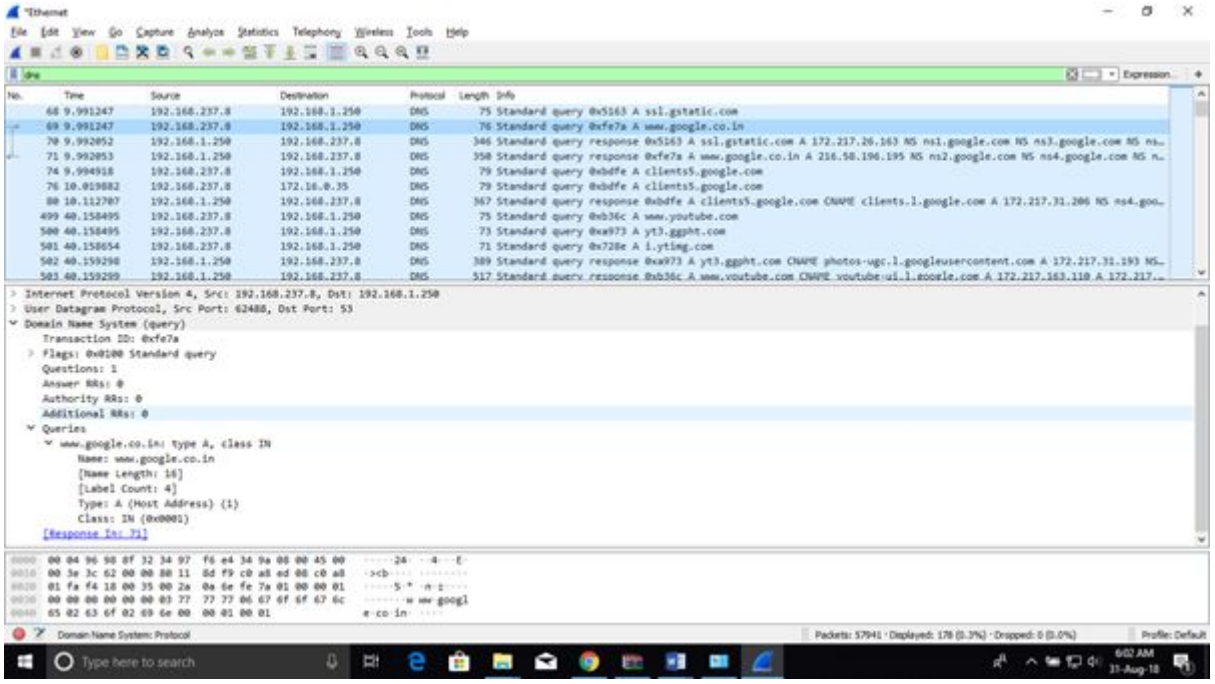
A1.1 Finding the last ARP Packet in the file. What IP Address was it asking about?

Ans. The selected packet is last ARP Packet and it was asking for IP Address 192.168.237.1. The below picture is for reference.



A1.2 Finding the first DNS Request. What is its Transaction ID in hexadecimal?

Ans. The selected packet is the first DNS Packet and Transaction ID in hexa-decimal is 0xfe7a.



A1.3 What Domain Name was it requesting?

Ans. It was requesting for "www.google.co.in".

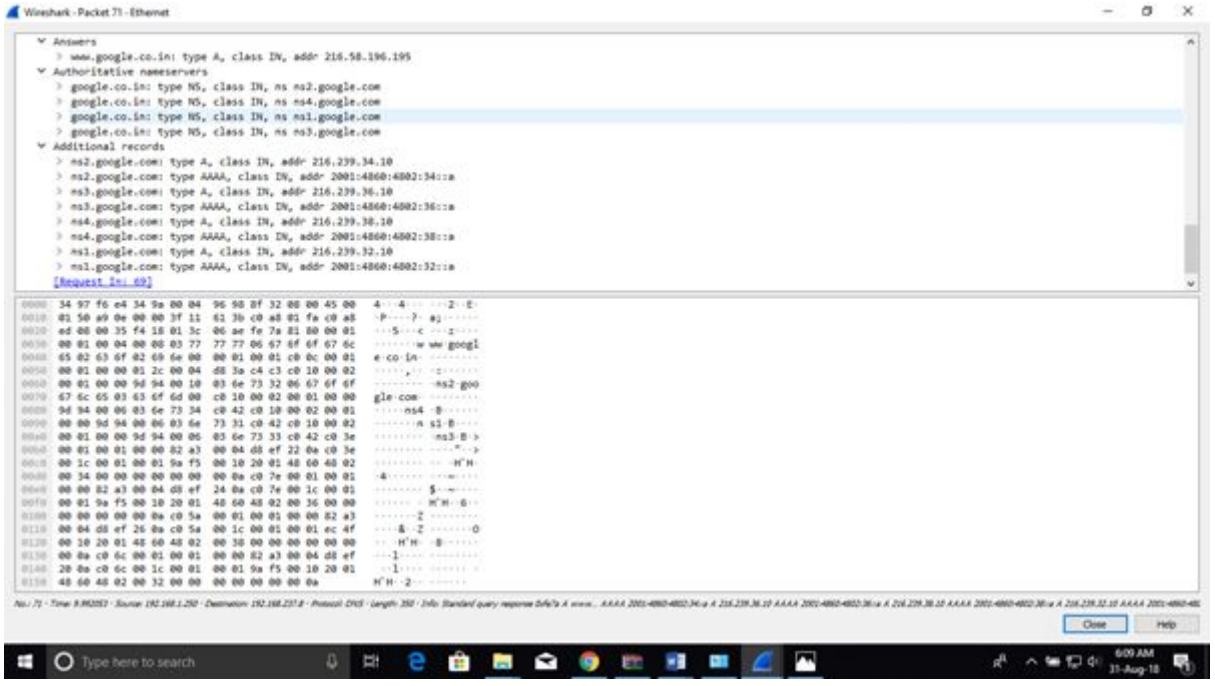
A1.4 What type of DNS Resource Record was requested?

Ans. DNS Resource Record of Type-A was requested. ‘A’ type of Record are used to point a domain or sub domain to an IP address.

A1.5 Finding the response to the DNS Request. What is the first IP address that it returned.

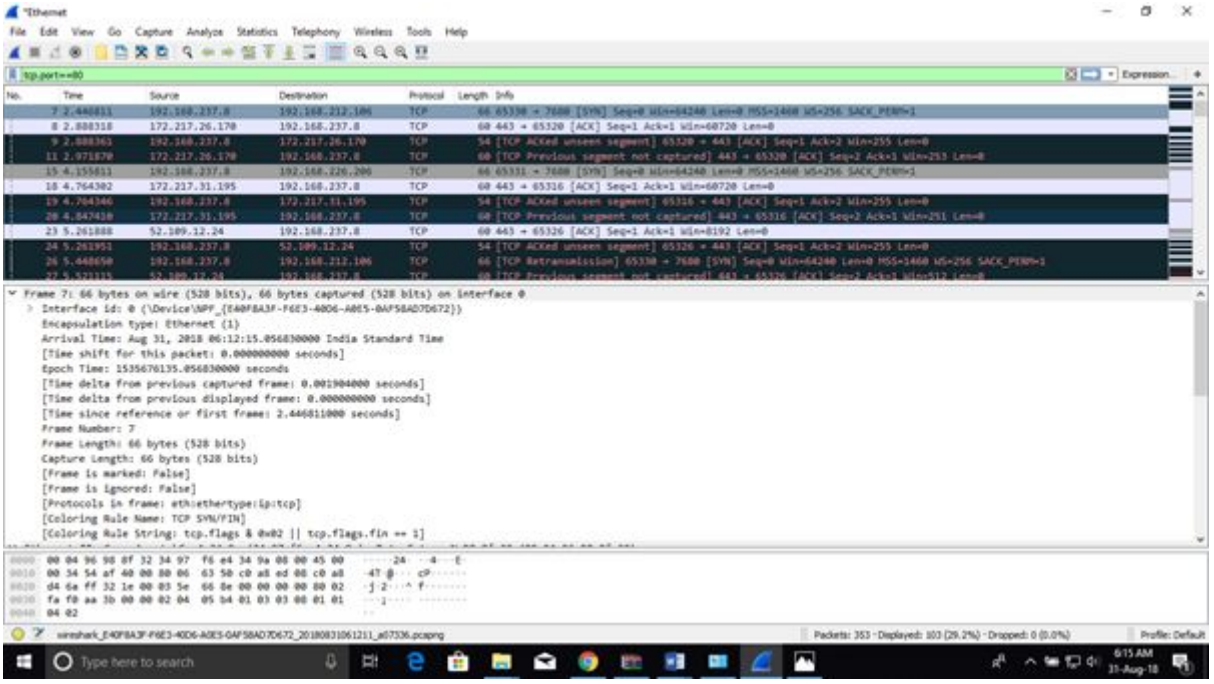
Ans. As in the picture in Ques.2 is written "Response in 71" meaning 71st packet is the response for the DNS Request.

The first IP Address it returned was 216.58.196.195.



A1.6 Finding the first TCP(IPv4 or IPv6) 3-Way Handshake.

Ans. To capture 3-Way Handshake. Open WireShark=>Open Web Browser=>Start Capture=>Stop Capture(After few seconds)=>Use filter as tcp.port==80, this is the port for web browsing traffic=> You can see the SYN request from your IP Address to Website's IP, then Website's IP sent acknowledgement for SYN as ACK, and then it synchronizes as SYN.



A1.7 Finding first IPv4 TCP Packet. What is its source IP Address?

Ans. The first IPv4 TCP Packet is shown in the picture in A1.6. Its a SYN Request made by the user. Its source IP Address is the user's IP Address i.e. 192.168.237.8.

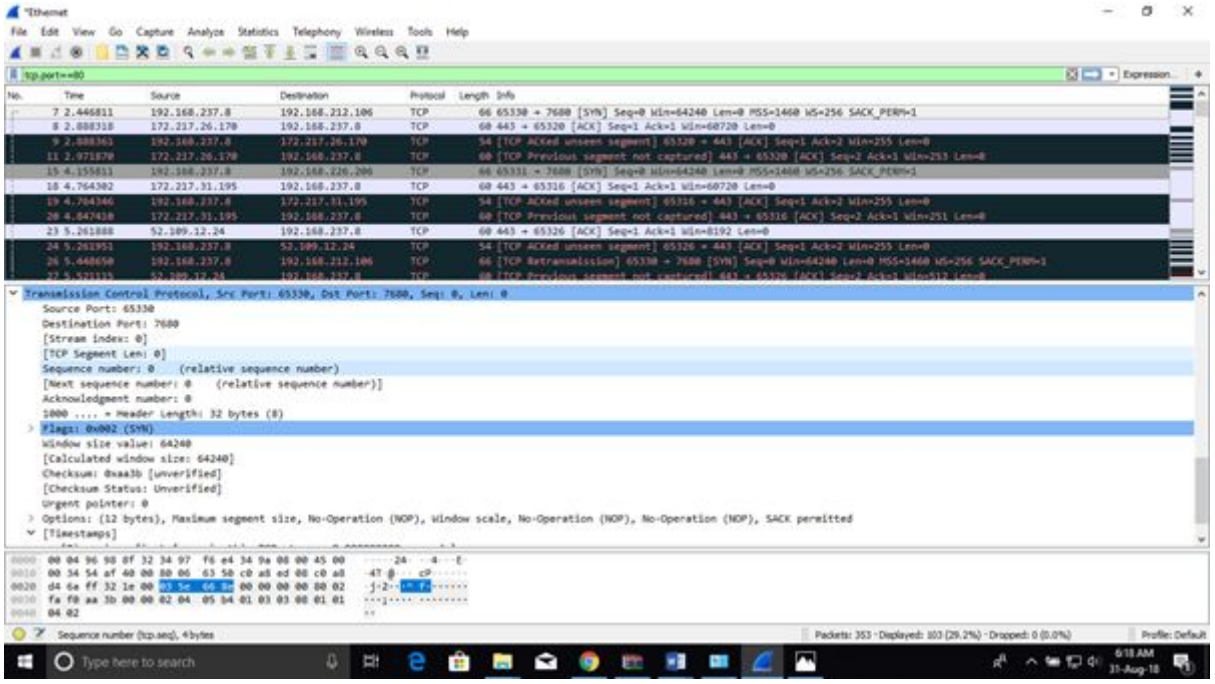
A1.8a What is the source IP Address in its Opening SYN Packet?

Ans. Source IP Address is 192.168.237.8.

A1.8b What is the destination IP Address in its Opening SYN Packet?

Ans. Destination IP Address is192.168.212.106.

A1.9 What is the (absolute)Sequence number sent in response to Question 8's opening SYN packet?



Ans. The Relative Sequence Number is shown as 0, since it is the first packet. Its Absolute Sequence Number is in Hexadecimal i.e. 035e668e which is 56518286 in Decimal.

A1.10 Finding the last TCP Packet with its FIN flag ON.

Ans.

A1.10a What is its source Port Number?

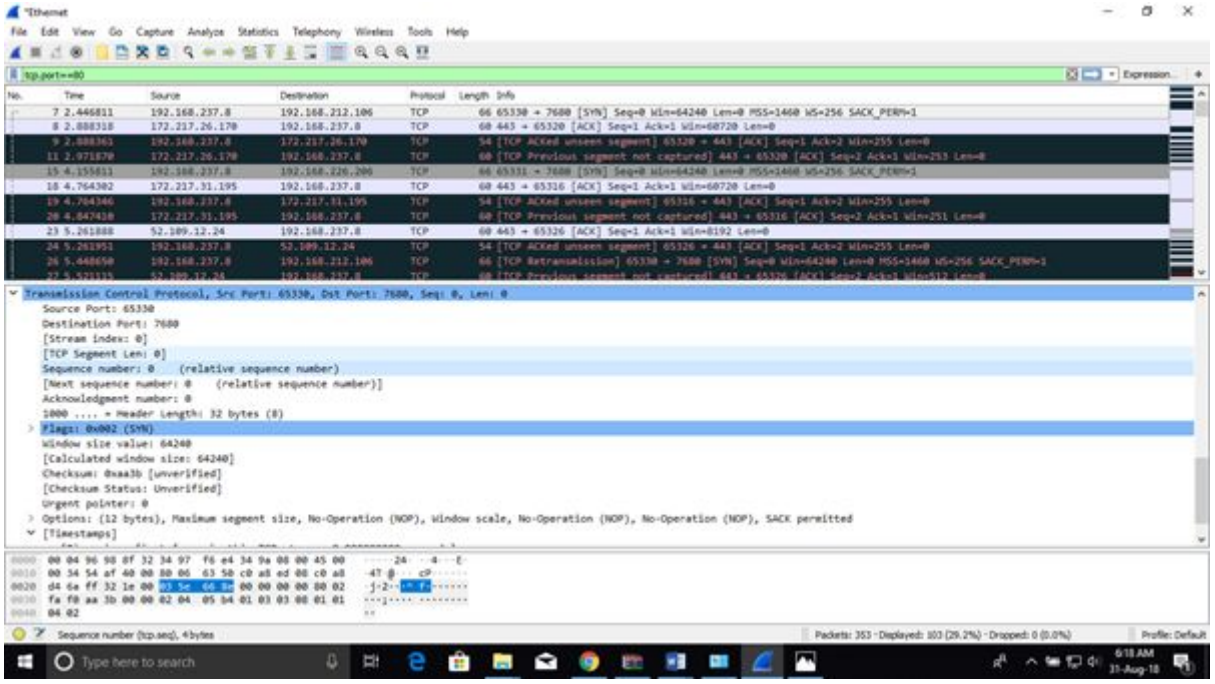
Ans. Source Port Number is 443.

A1.10a What is its Destination Port Number?

Ans. Destination Port Number is 6533

A1.11 Finding the Opening SYN+ACK packet that began the TCP session ending with that FIN packet. What is its sequence number?

Ans. The sequence number is 7bf90d9d.



A1.12 Look again at the FIN packet you found for Questions 10a and 10b. What was its "end byte" number? If source_port NOT(80 or 443):end_byte=ack_number else end_byte= seq_number.

Ans. End Byte Number = 2.

A1.13 How many bytes were sent from the responder to the sender in that tcp session?

Ans. Total bytes sent = 60. Header size = 20bytes.

Assignment 4 Date: 5th October 2018

1. Implement Go-back-n error control protocol for Data link layer.

Program:

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
#include <queue>
#include <algorithm>

using namespace std;

queue<int> frames;
int NACK = -1, ok = 0;
int error[35];

void *send(void *args) {
   int numFrames = (*(int *)args);

   for (int i = 0; !ok ;) {
      if (NACK != -1) {
         i = NACK;
         cout << "Sent frame number:\t" << i << "\n";
         frames = queue<int>();
         frames.push(i);
         i++;
         NACK = -1;
      } else if (i < numFrames) {
         cout << "Sent frame number:\t" << i << "\n";
         frames.push(i);
         i++;
      }

      usleep(1000000);
   }

   pthread_exit(NULL);
}

void *recv(void *args) {
   int numFrames = (*(int *)args);
   int processdFrames = 0;

   while (1) {
      if (!frames.empty()) {
         cout << "Received frame number:\t" << frames.front() << "\n";
         processdFrames++;
      }
                  if (!frames.empty() and !error[frames.front()])
                        cout<<endl;

      if (!frames.empty() and error[frames.front()]) {
         cout << "Erroneous frame number:\t" << frames.front() << "\n";
         cout<<"\nResending.....\n";
         processdFrames--;
```

```
            NACK = frames.front();
            while (NACK != -1);
            error[frames.front()] = 0;
            continue ;
        }

        if (!frames.empty()) frames.pop();

        if (processdFrames == numFrames) break ;

        usleep(1555555);
    }

    ok = 1;

    pthread_exit(NULL);
}

int main() {
    int numFrames;
    cout << "Enter number of frames to send: ";
    cin >> numFrames;

    int n = rand() % min(5, numFrames);
    for (int i = 0; i < n; i++) {
        int x = rand() % numFrames;
        error[x] = 1;
    }

    pthread_t sendThread, recvThread;

    int sendStatus = pthread_create(&sendThread, NULL, send, (void *)&numFrames);
    int recvStatus = pthread_create(&recvThread, NULL, recv, (void *)&numFrames);

    pthread_join(sendThread, NULL);
    pthread_join(recvThread, NULL);
    pthread_exit(NULL);

    return 0;
}
```

Output:

2. Implement selective repeat error control protocol in Data link layer

Program:

Client

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

int main(){
```

```
  int clientSocket,result;
  char buffer[1024];
  struct sockaddr_in serverAddr;
  socklen_t addr_size;

  clientSocket = socket(PF_INET, SOCK_STREAM, 0);
  serverAddr.sin_family = AF_INET;
  serverAddr.sin_port = htons(7891);
  serverAddr.sin_addr.s_addr = INADDR_ANY;
  memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);
  addr_size = sizeof serverAddr;
  connect(clientSocket, (struct sockaddr *) &serverAddr, addr_size);

int wt=9,winsiz,nr,i,sendf=0,x=1;
char arr[]={'S','O','U','M','Y','A','G','O','U','R','A','B','S','A','H','O','O'};
while(x)
{

 printf("Maximum Window size here is 9:\n");
 printf("What is the number of frames to be sent?:\n");
scanf("%d",&winsiz);
if(winsiz>9)
{
printf("Window limit exceeded \n");
break;
}

sprintf(buffer,"%d",winsiz);
send(clientSocket,buffer,13,0);
sleep(3);
strcpy(buffer , arr);
send(clientSocket,buffer,13,0);

for(i=0;i<(winsiz);i++)
{
printf(" sent: %c\n",arr[i]);
}
sleep(2);
recv(clientSocket,buffer,1024,0);
nr=atoi(buffer);
printf("%d frames sent successfully\n",nr);

printf("resending: %c\n",arr[nr]);
printf("continue sending\n");
for(i=winsiz;i<9;i++)
{
printf("sent: %c\n",arr[i]);
}
printf("All sent successfully\n");
x=0;
}
return 0;
}
```

Server

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

```
#include <bits/stdc++.h>
#include<sys/types.h>
#include<stdlib.h>
#include<arpa/inet.h>
#include<unistd.h>

int main(){
  int welcomeSocket, newSocket,result,result1,result2,result3;
  char buffer[1024];
  struct sockaddr_in serverAddr;
  struct sockaddr_storage serverStorage;
  socklen_t addr_size;


  welcomeSocket = socket(PF_INET, SOCK_STREAM, 0);
  serverAddr.sin_family = AF_INET;
  serverAddr.sin_port = htons(7891);
  serverAddr.sin_addr.s_addr = INADDR_ANY;
  memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);

  bind(welcomeSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr));

  if(listen(welcomeSocket,5)==0)
    printf("Listening\n");
  else
    printf("Error\n");

  addr_size = sizeof serverStorage;
  newSocket = accept(welcomeSocket, (struct sockaddr *) &serverStorage, &addr_size);

int winsi;
recv(newSocket,buffer,1024,0);
winsi = atoi(buffer);
sleep(6);

recv(newSocket,buffer,1024,0);
int n , i, errorfram=0;
char recvbuf[1024];
//printf("enter which frame which is in error:\n");
//scanf("%d",&errorfram);
errorfram = rand()%4;
n=errorfram;
printf("NoAck %d\n",n);
for(i=0;i<winsi;i++)
{
printf("recieved: %c\n",buffer[i]);
}
strcpy(recvbuf,buffer);
sprintf(buffer,"%d",n);
send(newSocket,buffer,100,0);
sleep(2);
printf("Received resent data: %c\n", recvbuf[n]);
printf("Acknowledging furthur sent\n");
for(i=winsi;i<9;i++)
{
printf("recieved: %c\n",recvbuf[i]);
}
printf("All received successfully\n");

  return 0;}
```

ASSIGNMENT 6

DATE: 31/8/18

A2.1 Create an iperf TCP and UDP server, change the default bandwidth to1000Mb change the interval between periodic bandwidth test to 60secs 120secs and 300secs and find the speed of the network. Also find the maximum throughput achieved during UDP Connection.

Ans. Tuning to a TCP Connection:

Server Side:

```
Command Prompt - iperf3 -s
F:\Setup Files\iperf-3.1.3-win64\iperf-3.1.3-win64>iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 127.0.0.1, port 64592
[  5] local 127.0.0.1 port 5201 connected to 127.0.0.1 port 64593
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec   108 MBytes   907 Mbits/sec
[  5]   1.00-2.00   sec   126 MBytes  1.06 Gbits/sec
[  5]   2.00-3.00   sec   118 MBytes   990 Mbits/sec
[  5]   3.00-4.00   sec   121 MBytes  1.02 Gbits/sec
[  5]   4.00-5.00   sec   116 MBytes   974 Mbits/sec
[  5]   5.00-6.00   sec   115 MBytes   965 Mbits/sec
[  5]   6.00-7.00   sec   118 MBytes   991 Mbits/sec
[  5]   7.00-8.00   sec   119 MBytes   996 Mbits/sec
[  5]   8.00-9.00   sec   120 MBytes  1.00 Gbits/sec
[  5]   9.00-10.00  sec   127 MBytes  1.06 Gbits/sec
[  5]  10.00-10.00  sec   384 KBytes  1.46 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-10.00  sec  0.00 Bytes  0.00 bits/sec                  sender
[  5]   0.00-10.00  sec  1.16 GBytes   997 Mbits/sec                  receiver
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Client Side:

```
F:\Setup Files\iperf-3.1.3-win64\iperf-3.1.3-win64>iperf3 -c 127.0.0.1 5201 -b 1
000M
Connecting to host 127.0.0.1, port 5201
[  4] local 127.0.0.1 port 64593 connected to 127.0.0.1 port 5201
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-1.00   sec   108 MBytes   906 Mbits/sec
[  4]   1.00-2.00   sec   126 MBytes  1.06 Gbits/sec
[  4]   2.00-3.00   sec   118 MBytes   994 Mbits/sec
[  4]   3.00-4.00   sec   121 MBytes  1.01 Gbits/sec
[  4]   4.00-5.00   sec   117 MBytes   979 Mbits/sec
[  4]   5.00-6.00   sec   115 MBytes   962 Mbits/sec
[  4]   6.00-7.00   sec   118 MBytes   991 Mbits/sec
[  4]   7.00-8.00   sec   119 MBytes   995 Mbits/sec
[  4]   8.00-9.00   sec   120 MBytes  1.00 Gbits/sec
[  4]   9.00-10.00  sec   127 MBytes  1.06 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-10.00  sec  1.16 GBytes   997 Mbits/sec                  sender
[  4]   0.00-10.00  sec  1.16 GBytes   997 Mbits/sec                  receiver

iperf Done.

F:\Setup Files\iperf-3.1.3-win64\iperf-3.1.3-win64>
```

Tuning to a UDP Connection:

Server Side:

```
Accepted connection from 127.0.0.1, port 51276
[  5] local 127.0.0.1 port 5201 connected to 127.0.0.1 port 51344
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datag
rams
[  5]   0.00-1.00   sec   108 MBytes   908 Mbits/sec  0.018 ms  0/13868 (0%)
[  5]   1.00-2.00   sec   120 MBytes  1.01 Gbits/sec  0.016 ms  21/15436 (0.14%)

[  5]   2.00-3.00   sec   128 MBytes  1.08 Gbits/sec  0.010 ms  0/16433 (0%)
[  5]   3.00-4.00   sec   119 MBytes   997 Mbits/sec  0.013 ms  0/15210 (0%)
[  5]   4.00-5.00   sec   117 MBytes   981 Mbits/sec  0.006 ms  0/14964 (0%)
[  5]   5.00-6.00   sec   113 MBytes   944 Mbits/sec  0.023 ms  0/14408 (0%)
[  5]   6.00-7.00   sec   119 MBytes   997 Mbits/sec  0.011 ms  67/15272 (0.44%)

[  5]   7.00-8.00   sec   118 MBytes   988 Mbits/sec  0.004 ms  0/15073 (0%)
[  5]   8.00-9.00   sec   124 MBytes  1.04 Gbits/sec  0.017 ms  0/15867 (0%)
[  5]   9.00-10.00  sec   122 MBytes  1.02 Gbits/sec  0.005 ms  0/15614 (0%)
[  5]  10.00-10.00  sec  0.00 Bytes  0.00 bits/sec  0.005 ms  0/0 (0%)
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datag
rams
[  5]   0.00-10.00  sec  0.00 Bytes  0.00 bits/sec  0.005 ms  88/152145 (0.058%)

```

Client Side:

```
F:\Setup Files\iperf-3.1.3-win64\iperf-3.1.3-win64>iperf3 -c 127.0.0.1 5201 -u -
b 1000m
Connecting to host 127.0.0.1, port 5201
[  4] local 127.0.0.1 port 51344 connected to 127.0.0.1 port 5201
[ ID] Interval           Transfer     Bandwidth       Total Datagrams
[  4]   0.00-1.00   sec   108 MBytes   909 Mbits/sec  13867
[  4]   1.00-2.00   sec   121 MBytes  1.01 Gbits/sec  15437
[  4]   2.00-3.00   sec   128 MBytes  1.08 Gbits/sec  16433
[  4]   3.00-4.00   sec   119 MBytes   997 Mbits/sec  15210
[  4]   4.00-5.00   sec   117 MBytes   981 Mbits/sec  14964
[  4]   5.00-6.00   sec   113 MBytes   944 Mbits/sec  14408
[  4]   6.00-7.00   sec   119 MBytes  1.00 Gbits/sec  15273
[  4]   7.00-8.00   sec   118 MBytes   987 Mbits/sec  15073
[  4]   8.00-9.00   sec   124 MBytes  1.04 Gbits/sec  15866
[  4]   9.00-10.00  sec   122 MBytes  1.02 Gbits/sec  15615
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datag
rams
[  4]   0.00-10.00  sec  1.16 GBytes   997 Mbits/sec  0.005 ms  88/152145 (0.058
%)
[  4] Sent 152145 datagrams

iperf Done.
```

Max throughput using UDP: 1.09Mbps

Bandwidth testing at 60sec,120sec and 300sec

```
:\Setup Files\iperf-3.1.3-win64\iperf-3.1.3-win64>iperf3 -s -i 60 300
-----------------------------------------------------------
erver listening on 5201
-----------------------------------------------------------
ccepted connection from 127.0.0.1, port 49873
  5] local 127.0.0.1 port 5201 connected to 127.0.0.1 port 49874
 ID] Interval           Transfer     Bandwidth
  5]   0.00-60.00  sec   112 MBytes  15.6 Mbits/sec
  5]  60.00-120.00 sec   118 MBytes  16.5 Mbits/sec
  5] 120.00-180.00 sec   114 MBytes  15.9 Mbits/sec
  5] 180.00-240.00 sec   113 MBytes  15.8 Mbits/sec
  5] 240.00-300.00 sec   112 MBytes  15.6 Mbits/sec
  5] 300.00-300.00 sec   128 KBytes  2.72 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
 ID] Interval           Transfer     Bandwidth
  5]   0.00-300.00 sec  0.00 Bytes  0.00 bits/sec                  sender
  5]   0.00-300.00 sec   568 MBytes  15.9 Mbits/sec                  receiver
-----------------------------------------------------------
```

Speed of the Network = 15.9Mbps

Assignment 7 Date: 26th October 2018

1. Implement shortest path routing using Djikstra Algorithm

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int minDistance(int dist[], bool sptSet[], int V)
{
  // Initialize min value
  int min = INT_MAX, min_index;

  for (int v = 0; v < V; v++)
    if (sptSet[v] == false && dist[v] <= min)
       min = dist[v], min_index = v;

  return min_index;
}

void parentPath(int path[], int i){
        if(path[i] == -1)
                return;
        parentPath(path, path[i]);
        cout<<i<<"->";
}

int main() {
        int n;
   cout<<"Enter the number of nodes you want to take :"<<endl;
   cin >> n;

      // Create a graph of size n where each edge weight is all nodes as each node is
connected to each other:
        vector< vector<int> >ar(n);
        for( int i=0;i<n;i++){
           ar[i].resize(n);
        }

        for( int i=0;i<n;i++){
           for( int j=0;j<n;j++){
              ar[i][j]=-1;
           }
        }
        for( int i=0;i<n;i++){
           ar[i][i]=0;
        }

      int temp=0;
      for (int i=0;i<n;i++){
        for(int j=i+1;j<n;j++)
                {
                temp=rand()%100;
                ar[i][j]=temp;
                ar[j][i]=temp;
                }
      }
```

```
    //Topology distance path to print
    for (int i=0;i<n;i++){
       for(int j=0;j<n;j++){
                cout<<ar[i][j]<<" ";
       }
       cout<<endl;
    }


    cout<<"Enter the starting node"<<endl;
                int start;
    cin >> start;
    int src = start;
       vector<int> dist(n,0);
       priority_queue <pair<int,int> > g;
       vector<int> visit;
       visit.push_back(start);

       for(int i=0; i<n;i++)
                g.push(make_pair(ar[start][i],i));
       dist[start] = 0;

       //cout<<endl;
       while(!g.empty() || visit.size()<n)
       {
          pair<int,int> v=g.top();
          int a= v.first;
          int b= v.second;
          g.pop();
          if(find(visit.begin(),visit.end(),b)==visit.end()){
          visit.push_back(b);
          for(int i =0;i<n;i++)
          {
          if(ar[b][i] && dist[i] == 0 )
                                                {
                 g.push(make_pair(ar[b][i],i));
                 dist[i] = dist[b]+ar[b][i];
          }
          }
          }
       }
       int cost=0;
       cost+=ar[src][visit[0]];
       for(int i=1;i<visit.size();i++){
                cout<<visit[i]<<endl;
                cost+=ar[visit[i-1]][visit[i]];
                }

       cout<<"min cost = "<<cost<<endl;
   return 0;
}
```

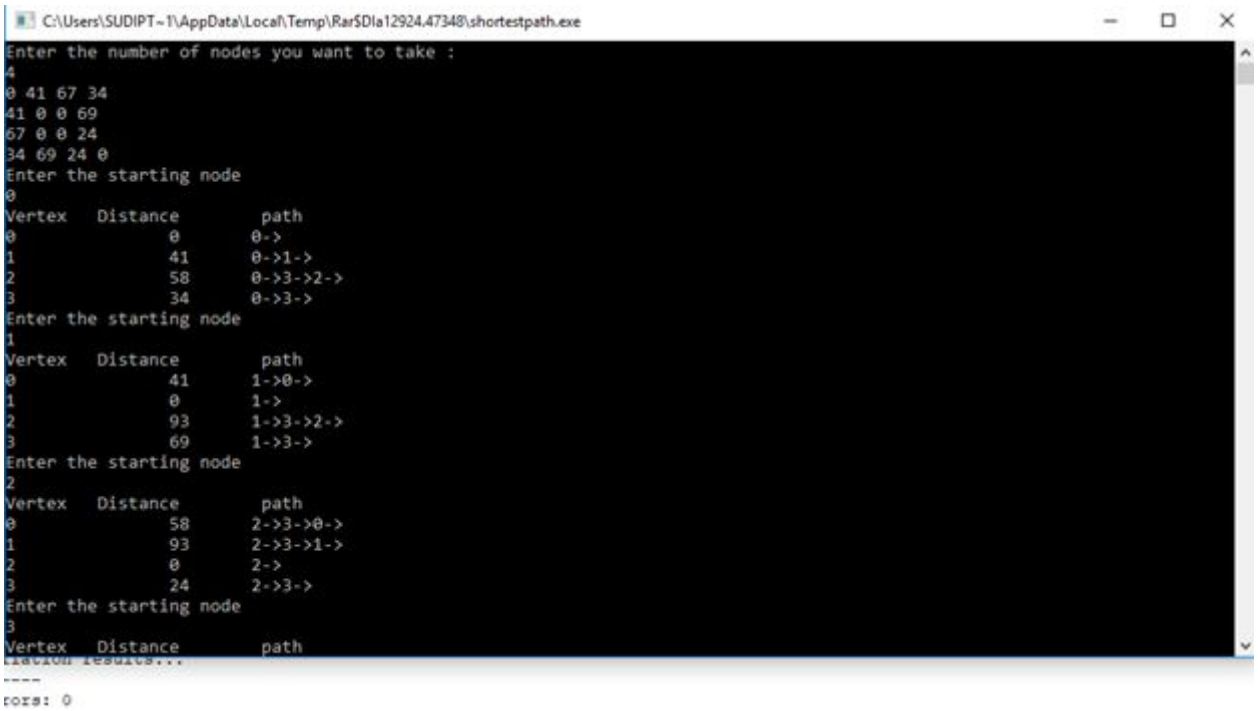
2. Implement shortest path algorithm, using Floyd- warshall's algorithm

Program:

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int minDistance(int dist[], bool sptSet[], int V)
{
  // Initialize min value
  int min = INT_MAX, min_index;

  for (int v = 0; v < V; v++)
    if (sptSet[v] == false && dist[v] <= min)
       min = dist[v], min_index = v;

  return min_index;
}

void parentPath(int path[], int i){
        if(path[i] == -1)
                return;
        parentPath(path, path[i]);
        cout<<i<<"->";
}

int main() {
        int n;
   cout<<"Enter the number of nodes you want to take :"<<endl;
   cin >> n;

      // Create a graph of size n where each edge weight is all nodes as each node is
connected to each other:
        vector< vector<int> >ar(n);
        for( int i=0;i<n;i++){
           ar[i].resize(n);
        }

        for( int i=0;i<n;i++){
           for( int j=0;j<n;j++){
              ar[i][j]=-1;
           }
        }
        for( int i=0;i<n;i++){
           ar[i][i]=0;
        }
      /*cin >> m;
      // read and set edges
      m =queries;
      for (int i = 0; i < m; i++) {
         int u, v;
         cin >> u >> v;
         u--, v--;
         // add each edge to the graph
         ar[u][v]=6;
         ar[v][u]=6;
      }*/
      int temp=0;
      for (int i=0;i<n;i++){
```

```
    for(int j=i+1;j<n;j++)
            {
            temp=rand()%100;
            ar[i][j]=temp;
            ar[j][i]=temp;
            }
  }
  //Topology distance path to print
  for (int i=0;i<n;i++){
    for(int j=0;j<n;j++){
            cout<<ar[i][j]<<" ";
    }
    cout<<endl;
  }

 /* cout<<"Enetr the threshold distance for communication"<<endl;
  int thresh;
  cin>>thresh;
            for (int i=0;i<n;i++){
    for(int j=i+1;j<n;j++){
            if(ar[i][j] > thresh)
            {
                    ar[i][j]=-1;
                    ar[j][i]=-1;
            }
    }
  }
            //Topology distance path to print
  for (int i=0;i<n;i++){
    for(int j=0;j<n;j++){
            cout<<ar[i][j]<<" ";
    }
    cout<<endl;
  }*/

  for(int p=0;p<5;p++){
  cout<<"Enter the starting node"<<endl;
            int start;
  cin >> start;
  int src = start;

  int dist[n];

bool sptSet[n];

int path[n];

for (int i = 0; i < n; i++)
  dist[i] = INT_MAX, sptSet[i] = false,path[src]=-1;

dist[src] = 0;

for (int count = 0; count < n-1; count++)
{
 int u = minDistance(dist, sptSet,n);

 sptSet[u] = true;

 for (int v = 0; v < n; v++)
```

```
        if (!sptSet[v] && ar[u][v]  && ar[u][v] >0 && dist[u] != INT_MAX  && dist[u]+ar[u][v] <
dist[v])
          {
          dist[v] = dist[u] + ar[u][v];
          path[v] = u;
          }
    }

    printf("Vertex \t Distance \t path\n");
          for (int i = 0; i < n; i++)
          {
          cout<< i <<"\t\t"<<dist[i]<<"\t"<<src<<"->";
                    parentPath(path,i);
                    cout<<endl;
                    }
   }
   return 0;
}
```



Assignment 8

1. Implement the following routing algorithm

1.1 Bellman-Ford algorithm

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

struct Edge
{

      int source, destination, weight;
};

struct Graph
{
      int V, E;


      struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
      struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph));

    graph->V = V;   //assigning values to structure elements that taken form
user.

    graph->E = E;

    graph->edge = (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );

      return graph;
}

void FinalSolution(int dist[], int n)
{
    printf("\nVertex\tDistance from Source Vertex\n");
      int i;

      for (i = 0; i < n; ++i){
        printf("%d \t\t %d\n", i, dist[i]);
      }
}

void BellmanFord(struct Graph* graph, int source)
{
      int V = graph->V;

      int E = graph->E;

      int StoreDistance[V];
```

```
      int i,j;



      for (i = 0; i < V; i++)
        StoreDistance[i] = INT_MAX;

    StoreDistance[source] = 0;

      //The shortest path of graph that contain V vertices, never contain "V-1"
edges. So we do here "V-1" relaxations
      for (i = 1; i <= V-1; i++)
      {
        for (j = 0; j < E; j++)
      {
            int u = graph->edge[j].source;

            int v = graph->edge[j].destination;

            int weight = graph->edge[j].weight;

            if (StoreDistance[u] + weight < StoreDistance[v])
                StoreDistance[v] = StoreDistance[u] + weight;
      }
      }


      for (i = 0; i < E; i++)
      {
        int u = graph->edge[i].source;

        int v = graph->edge[i].destination;

        int weight = graph->edge[i].weight;

      if (StoreDistance[u] + weight < StoreDistance[v])
            printf("This graph contains negative edge cycle\n");
      }

    FinalSolution(StoreDistance, V);

    return;
}

int main()
{
      int V,E,S;

    printf("Enter number of vertices in graph\n");
    scanf("%d",&V);

    printf("Enter number of edges in graph\n");
    scanf("%d",&E);

    printf("Enter your source vertex number\n");
    scanf("%d",&S);

      struct Graph* graph = createGraph(V, E);
```

```
      int i;
    for(i=0;i<E;i++){
        printf("\nEnter edge %d properties Source, destination, weight
respectively\n",i+1);
        scanf("%d",&graph->edge[i].source);
        scanf("%d",&graph->edge[i].destination);
        scanf("%d",&graph->edge[i].weight);
      }

    BellmanFord(graph, S);
    //passing created graph and source vertex to BellmanFord Algorithm function

      return 0;
}
```

1.2 Link state algorithm

```
2.  #include<bits/stdc++.h>
3.  using namespace std;
4.
5.  #define N 5
6.  struct node          //link state packets structure
7.  {
8.      int dist[N];
9.      int ttl;
10.  int seq;
11.  char id;
12.}rt[N];
13.int wt[N][N];
14.
15.
16.int minDistance(int dist[], bool sptSet[])
17.{
18.   // Initialize min value
19.   int min = INT_MAX, min_index;
20.
21.   for (int v = 0; v < N; v++)
22.       if (sptSet[v] == false && dist[v] <= min)
23.       min = dist[v], min_index = v;
24.
25.   return min_index;
26.}
27.
28.
29.void dijkstra(int src)
30.{
31.       int distance[N];        //output
32.
33.       bool sptSet[N];
34.
35.
36.       for (int i = 0; i < N; i++) {
37.       distance[i] = INT_MAX, sptSet[i] = false;
38.   }
39.
40.       distance[src] = 0;
41.
```

```
42.
43.     for (int count = 0; count < N-1; count++)
44.     {
45.
46.     int u = minDistance(distance, sptSet);
47.
48.
49.     sptSet[u] = true;
50.
51.
52.     for (int v = 0; v < N; v++)
53.     if (!sptSet[v] && wt[u][v] && distance[u] != INT_MAX && distance[u]+wt[u][v] <
    distance[v])
54.             distance[v] = distance[u] + wt[u][v];
55.  //return dist[dst];
56.  }
57.  for(int i=0;i<N;i++){
58.     rt[src].dist[i]=distance[i];
59.     //cout<<"("<<src<<"->"<<i<<")="<<dist[i]<<"\t";
60.  }
61.  cout<<endl;
62.}
63.
64.
65.int main(){
66.  int graph[N][N];
67.
68.  srand(time(0));
69.  for(int i=0;i<N;i++){
70.     for(int j=i;j<N;j++){
71.             graph[i][i]=0;
72.             if(i!=j){
73.                     graph[i][j]=rand()%5;
74.                     graph[j][i]=graph[i][j];
75.             }
76.     }
77.  }
78.
79.
80.  //2. creating distance matrices
81.  for(int i=0;i<N;i++){
82.     for(int j=0;j<N;j++){
83.             if(graph[i][j]!=0){
84.                     graph[i][j]=1;
85.                     graph[j][i]=graph[i][j];
86.             }
87.     }
88.  }
89.
90.  for(int i=0;i<N;i++){
91.     for(int j=0;j<N;j++){
92.             if(graph[i][j]==1){
93.                     wt[i][j]=rand()%50+1;
94.                     wt[j][i]=wt[i][j];
95.             }else
96.                     wt[i][j]=0;
97.     }
98.  }
99.
100.            cout<<"Path matrix:\n";
```

```
101.            for(int i=0;i<N;i++){
102.            for(int j=0;j<N;j++){
103.                    cout<<graph[i][j]<<"\t";
104.            }
105.            cout<<endl;
106.            }
107.            cout<<endl;
108.            cout<<"Weight matrix:\n";
109.    for(int i=0;i<N;i++){
110.            for(int j=0;j<N;j++){
111.                    cout<<wt[i][j]<<"\t";
112.            }
113.            cout<<endl;
114.            }
115.            cout<<endl;
116.
117.            //3.Detecting neighbours and maintaining them in a ls packet i.e arrays
118.            int neigh[N][N],index;
119.            for(int i=0;i<N;i++){
120.            for(int j=0;j<N;j++){
121.                    neigh[i][j]=-1;
122.            }
123.            }
124.            for(int i=0;i<N;i++){
125.            index=0;
126.            for(int j=0;j<N;j++){
127.                    if(graph[i][j]!=0)
128.                    neigh[i][index++]=j;
129.            }
130.            }
131.
132.
133.            char ch[10]={'A','B','C','D','E','F','G','H','I','J'};
134.
135.            for(int i=0;i<N;i++){           //routing table initialisation
136.            rt[i].id=ch[i];
137.            rt[i].ttl=rand()%5000+1;
138.            rt[i].seq=rand()%11+1;
139.            for(int j=0;j<N;j++){
140.                    rt[i].dist[j]=wt[i][j];
141.            }
142.            }
143.
144.            for(int i=0;i<N;i++){
145.            cout<<"\nLink State Packet for "<<rt[i].id<<"\nSequence no.:
    "<<rt[i].seq<<"\nTTL period: "<<rt[i].ttl<<" ms"<<endl;
146.            cout<<"Neighbours: \nNode\tCost\n";
147.            for(int j=0;j<N;j++){
148.                    if(rt[i].dist[j]!=0)
149.                    cout<<rt[j].id<<"\t"<<rt[i].dist[j]<<endl;
150.            }
151.
152.            }
153.
154.            for(int i=0;i<N;i++){
155.            dijkstra(i);
156.            }
157.
158.            cout<<"After updation: \n";
159.            for(int i=0;i<N;i++){
```

```
        cout<<"\nLink State Packet for "<<rt[i].id<<"\nSequence no.: "<<rt[i].seq<<"\nTTL period: "<<rt[i].ttl<<" ms"<<endl;
        cout<<"Neighbours: \nNode\tCost\n";
        for(int j=0;j<N;j++){
                if(rt[i].dist[j]!=0)
                cout<<rt[j].id<<"\t"<<rt[i].dist[j]<<endl;
        }

        }
}
```

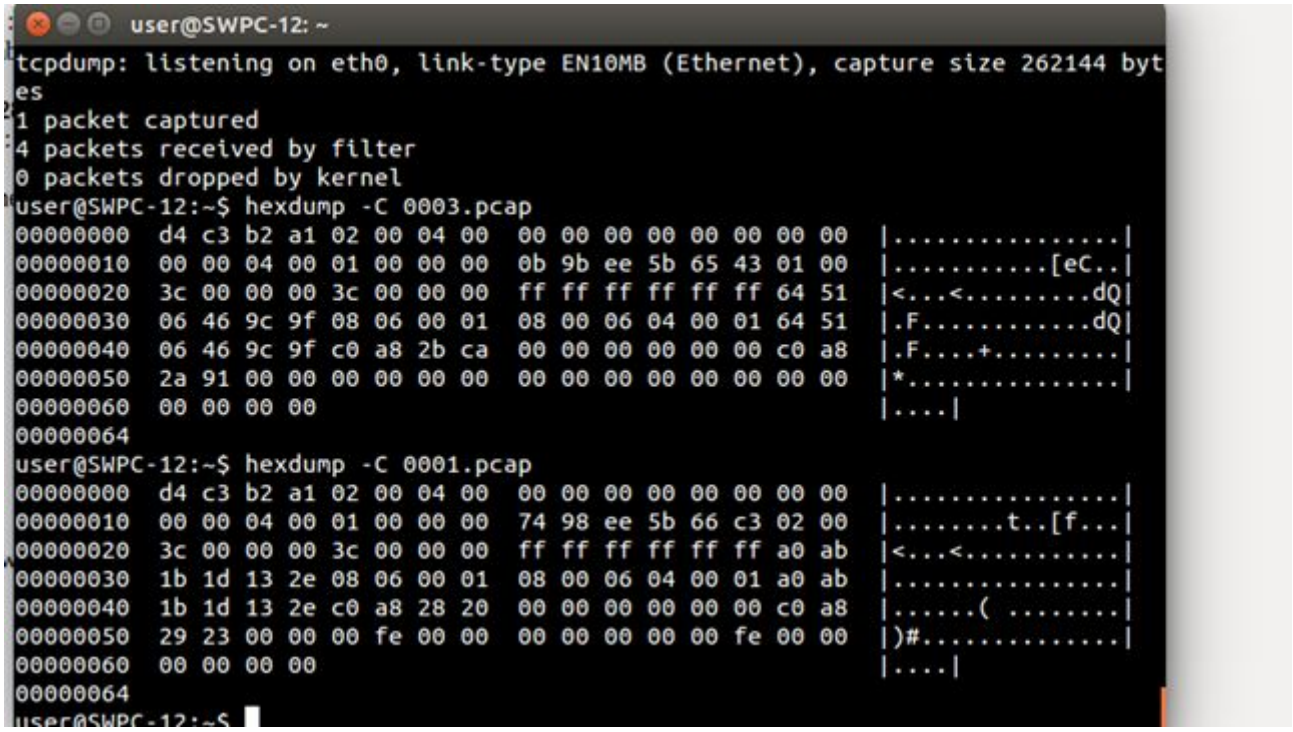
Assignment 9

1. Creating a .pcap file of 1 packet of ARP request

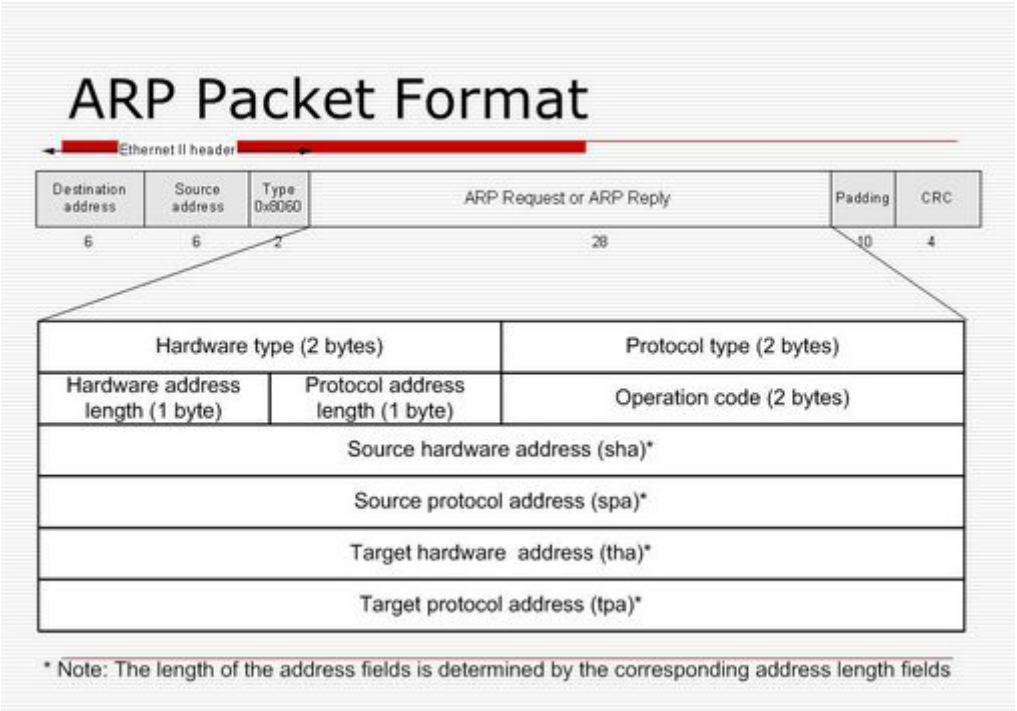
Ans. sudo tcpdump -c 1 -i eth0 arp -w 0003.pcap

2. Getting the hexadecimal values of the .pcap file

Ans. hexdump -C 0003.pcap



```
user@SWPC-12: ~
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 byt
es
1 packet captured
4 packets received by filter
0 packets dropped by kernel
user@SWPC-12:~$ hexdump -C 0003.pcap
00000000  d4 c3 b2 a1 02 00 04 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 04 00 01 00 00 00  0b 9b ee 5b 65 43 01 00  |...........[eC..|
00000020  3c 00 00 00 3c 00 00 00  ff ff ff ff ff ff 64 51  |<...<.........dQ|
00000030  06 46 9c 9f 08 06 00 01  08 00 06 04 00 01 64 51  |.F............dQ|
00000040  06 46 9c 9f c0 a8 2b ca  00 00 00 00 00 00 c0 a8  |.F....+.........|
00000050  2a 91 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |*...............|
00000060  00 00 00 00                                       |....|
00000064
user@SWPC-12:~$ hexdump -C 0001.pcap
00000000  d4 c3 b2 a1 02 00 04 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 04 00 01 00 00 00  74 98 ee 5b 66 c3 02 00  |........t..[f...|
00000020  3c 00 00 00 3c 00 00 00  ff ff ff ff ff ff a0 ab  |<...<...........|
00000030  1b 1d 13 2e 08 06 00 01  08 00 06 04 00 01 a0 ab  |................|
00000040  1b 1d 13 2e c0 a8 28 20  00 00 00 00 00 00 c0 a8  |......( ........|
00000050  29 23 00 00 00 fe 00 00  00 00 00 00 00 fe 00 00  |)#..............|
00000060  00 00 00 00                                       |....|
00000064
user@SWPC-12:~$
```



First 40 bytes: Preamble | SOF |

Next 28 bytes:

00 01	**Ethernet (Hardware Type)**
08 00	**IP (Protocol Type)**
06	**Ethernet Address Size**
04	**Protocol Address Size**
00 01	**Operation Code -> Request (00 02 for reply)**
64 51 06 46 9c 9f	**Sender Ethernet Address**
c0 a8 2b ca	**Sender IP Address**
00 00 00 00 00 00	**Target Ethernet Address**

c0 a8 29 23 **Target IP Address**

Next 18 Bytes: Padding | CRC |