**Demo Experiment**

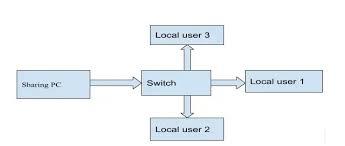
* 1. **Title of the experiment: Lan Setup**

**0.2 Aim of the experiment: Introduction to hardware components and ethernet LAN**

**0.3 Theoretical background for the experiment:**

A local area network (LAN) is a computer network that interconnects devices within a limited geographical area, such as a home, office, or campus. LANs are typically designed using switches as the central networking device. Switches are network devices that enable multiple devices to connect and communicate within a LAN environment.

* 1. **Block Diagram:**



* 1. **Step by step procedure to carry out the experiment:**
* Determine LAN Requirements
* Choose the Switch
* Plan the Network Topology
* Prepare the Cabling
* Connect the Switch
* Connect Devices to the Switch
* Configure IP Addresses
* Test Connectivity
  1. **Conclusion of the experiment:**

In conclusion, the LAN setup experiment using a switch and LAN cables is a straightforward process involving equipment selection, device connectivity, network configuration, and testing. By following the steps outlined, a functional LAN can be established.

**Experiment No: 1**

* 1. **Title of the experiment: Bit stuffing and destuffing**
  2. **Aim of the experiment: Implementation of bit stuffing and destuffing**
  3. **Theoretical background for the experiment:**

[Bit Stuffing](https://www.geeksforgeeks.org/bit-stuffing-in-computer-network/) is a process of inserting an extra bit as **0**, once the frame sequence encountered **5** consecutive **1’s**. Given an [array](https://www.geeksforgeeks.org/array-data-structure/), **arr[]** of size **N** consisting of **0’s** and **1’s,**the task is to return an array after the bit stuffing.

**Examples:**

**Input:** N = 6, arr[] = {1, 1, 1, 1, 1, 1}  
**Output:** 1111101**Explanation:** During the traversal of the array, 5 consecutive 1’s are encountered after the 4th index of the given array. Hence, a zero bit has been inserted into the stuffed array after the 4th index

**Input:** N = 6, arr[] = {1, 0, 1, 0, 1, 0}  
**Output:** 101010

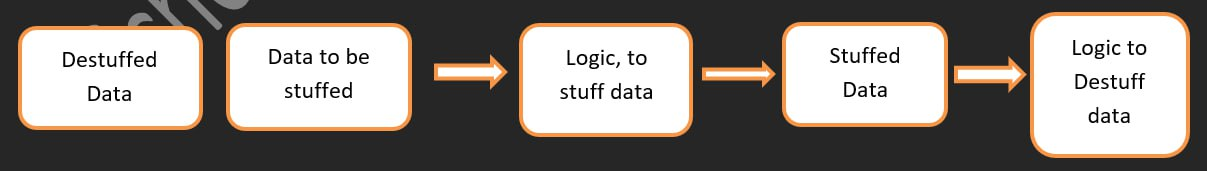
**Bit Destuffing** or **Bit Unstuffing** is a process of undoing the changes in the array made during the [bit stuffing](https://www.geeksforgeeks.org/bit-stuffing-in-computer-network/) process i.e, removing the extra **0** bit after encountering **5** consecutive **1’s**.

**Examples:**

**Input:** N = 7, arr[] = {1, 1, 1, 1, 1, 0, 1}  
**Output:** 111111  
**Explanation:** During the traversal of the array, 5 consecutive 1’s are encountered after the 4th index of the given array. Hence, the next 0 bit must be removed to de-stuffed array.

**Input:** N = 6, arr[] = {1, 0, 1, 0, 1, 0}  
**Output:** 101010

**1.4 Block Diagram:**





* 1. **Step by step procedure to carry out the experiment:**

**Approach:** The idea is to check if the given array consists of 5 consecutive **1’s**. Follow the steps below to solve the problem:

* [Initialize the array **brr[]**](https://www.geeksforgeeks.org/arrays-in-c-cpp/) which stores the stuffed array. Also, create a variable count which maintains the count of the consecutive 1’s.
* [Traverse in a while loop](https://www.geeksforgeeks.org/c-c-while-loop-with-examples/) using a variable **i** in the range **[0, N)** and perform the following tasks:
* If **arr[i]** is **1** then check for the next **4** bits if they are set bits as well. If they are, then insert a 0 bit after inserting all the 5 set bits into the array **brr []**.
* Otherwise, insert the value of **arr[i]** into the array **brr []**.
  1. **C Program/Commands:**

**// C program for the above approach**

**#include <stdio.h>**

**#include <string.h>**

**// Function for bit stuffing**

**void bitStuffing(int N, int arr[])**

**{**

**// Stores the stuffed array**

**int brr[30];**

**// Variables to traverse arrays**

**int i, j, k;**

**i = 0;**

**j = 0;**

**// Loop to traverse in the range [0, N)**

**while (i < N) {**

**// If the current bit is a set bit**

**if (arr[i] == 1) {**

**// Stores the count of consecutive ones**

**int count = 1;**

**// Insert into array brr[]**

**brr[j] = arr[i];**

**// Loop to check for**

**// next 5 bits**

**for (k = i + 1;**

**arr[k] == 1 && k < N && count < 5; k++) {**

**j++;**

**brr[j] = arr[k];**

**count++;**

**// If 5 consecutive set bits**

**// are found insert a 0 bit**

**if (count == 5) {**

**j++;**

**brr[j] = 0;**

**}**

**i = k;**

**}**

**}**

**// Otherwise insert arr[i] into**

**// the array brr[]**

**else {**

**brr[j] = arr[i];**

**}**

**i++;**

**j++;**

**}**

**// Print Answer**

**for (i = 0; i < j; i++)**

**printf("%d", brr[i]);**

**}**

**// Driver Code**

**int main()**

**{**

**int N = 6;**

**int arr[] = { 1, 1, 1, 1, 1, 1 };**

**bitStuffing(N, arr);**

**return 0;**

**}**

**Bit Destuffing**

**// C program for the above approach**

**#include <stdio.h>**

**#include <string.h>**

**// Function for bit de-stuffing**

**void bitDestuffing(int N, int arr[])**

**{**

**// Stores the de-stuffed array**

**int brr[30];**

**// Variables to traverse the arrays**

**int i, j, k;**

**i = 0;**

**j = 0;**

**// Stores the count of consecutive ones**

**int count = 1;**

**// Loop to traverse in the range [0, N)**

**while (i < N) {**

**// If the current bit is a set bit**

**if (arr[i] == 1) {**

**// Insert into array brr[]**

**brr[j] = arr[i];**

**// Loop to check for**

**// the next 5 bits**

**for (k = i + 1;**

**arr[k] == 1**

**&& k < N**

**&& count < 5;**

**k++) {**

**j++;**

**brr[j] = arr[k];**

**count++;**

**// If 5 consecutive set**

**// bits are found skip the**

**// next bit in arr[]**

**if (count == 5) {**

**k++;**

**}**

**i = k;**

**}**

**}**

**// Otherwise insert arr[i] into**

**// the array brr**

**else {**

**brr[j] = arr[i];**

**}**

**i++;**

**j++;**

**}**

**// Print Answer**

**for (i = 0; i < j; i++)**

**printf("%d", brr[i]);**

**}**

**// Driver Code**

**int main()**

**{**

**int N = 7;**

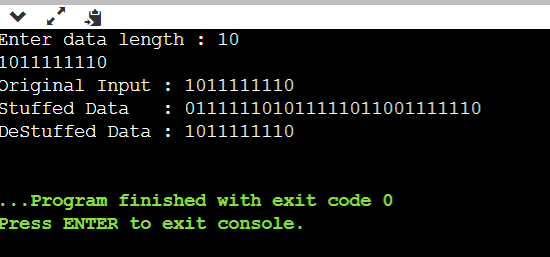
**int arr[] = { 1, 1, 1, 1, 1, 0, 1 };**

**bitDestuffing(N, arr);**

**return 0;**

**}**

* 1. **Results:** (Screen Shots):



* 1. **Conclusion of the experiment:**

Bit stuffing is employed when transmitting data over a communication channel that may introduce synchronization issues due to the occurrence of specific bit patterns. By inserting additional bits into the data stream, bit stuffing helps maintain synchronization between the sender and the receiver.

**Experiment No: 2**

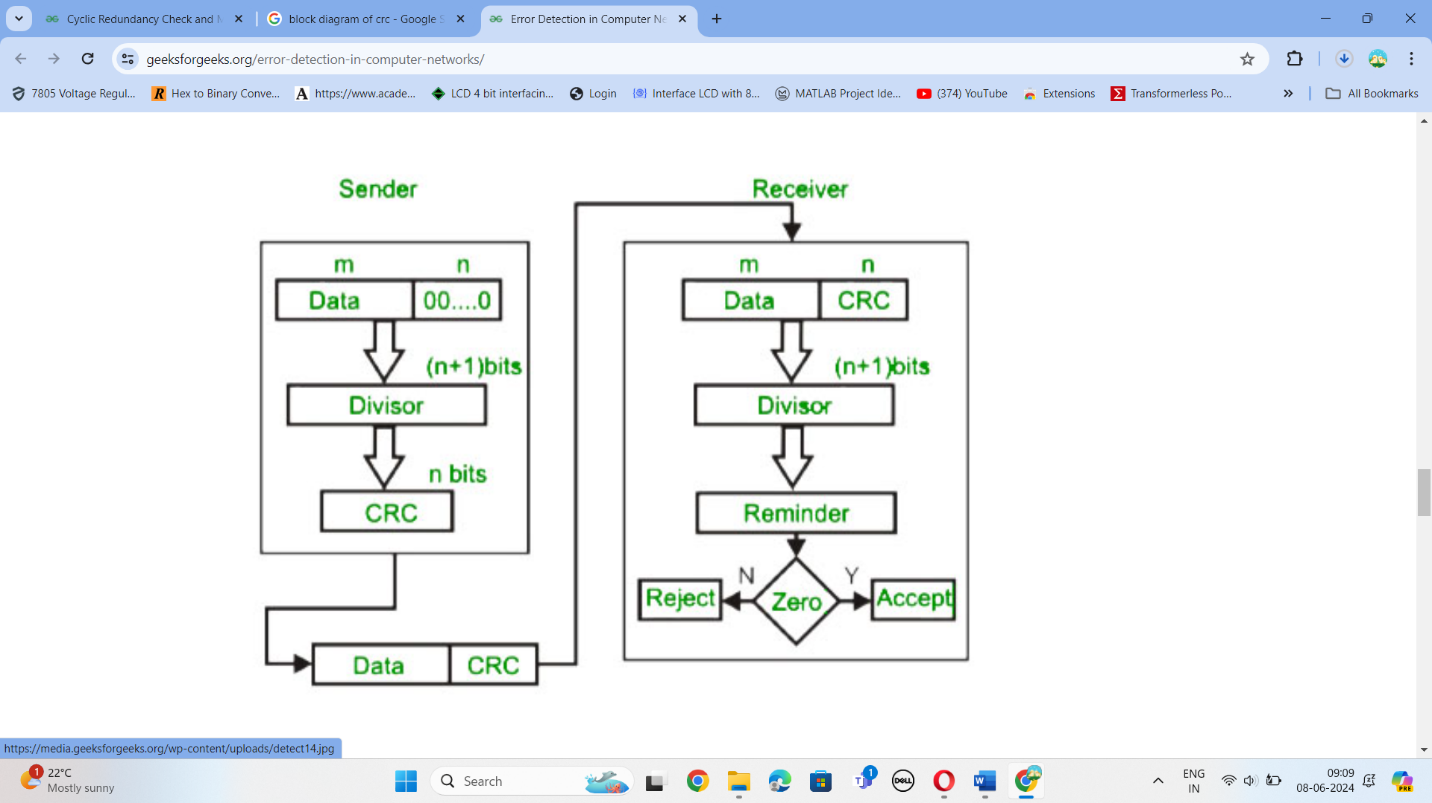
**2.1 Title of the experiment: CRC**

**2.2 Aim of the experiment: Implementation of Cyclic Redundancy Check**

**2.3 Theoretical background for the experiment:**

CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in the communication channel.   
CRC uses **Generator Polynomial**which is available on both sender and receiver side. An example generator polynomial is of the form like x3 + x + 1. This generator polynomial represents key 1011. Another example is x2 + 1 that represents key 101.

**2.4 Block Diagram:**



* 1. **Step by step procedure to carry out the experiment:**

n : Number of bits in data to be sent

from sender side.

k : Number of bits in the key obtained

from generator polynomial.

**Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):**

1. The binary data is first augmented by adding k-1 zeros in the end of the data
2. Use **modulo-2 binary division** to divide binary data by the key and store remainder of division.
3. Append the remainder at the end of the data to form the encoded data and send the same

**Receiver Side (Check if there are errors introduced in transmission)**  
Perform modulo-2 division again and if the remainder is 0, then there are no errors.

* 1. **C Program/Commands:**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<string.h>**

**#define n strlen(key)**

**char data[20];**

**char check\_data[20];**

**char key[20];**

**int data\_len,i,j;**

**void XOR()**

**{**

**for(j = 1;j < n; j++)**

**check\_data[j] = (( check\_data[j] == key[j])?'0':'1');**

**}**

**void receiver()**

**{**

**printf("Enter the received data: ");**

**scanf("%s", data);**

**printf("Data received: %s", data);**

**crc();**

**printf("\nRemainder at receiver side is: %s",check\_data);**

**for(i=0;(i<n-1) && (check\_data[i]!='1');i++);**

**if(i<n-1)**

**printf("\nError detected\n");**

**else**

**printf("\nNo error detected\n");**

**}**

**void crc()**

**{**

**for(i=0;i<n;i++)**

**check\_data[i]=data[i];**

**do{**

**if(check\_data[0]=='1')**

**XOR();**

**for(j=0;j<n-1;j++)**

**check\_data[j]=check\_data[j+1];**

**check\_data[j]=data[i++];**

**}while(i<=data\_len+n-1);**

**}**

**int main()**

**{**

**printf("Input Data: ");**

**scanf("%s",data);**

**printf("\nEnter the key: ");**

**scanf("%s",key);**

**data\_len=strlen(data);**

**for(i=data\_len;i<data\_len+n-1;i++)**

**data[i]='0';**

**printf("\nInput Data with n-1 zeros: %s\n",data);**

**crc();**

**printf("\nRemainder at sender side is: %s",check\_data);**

**for(i=data\_len;i<data\_len+n-1;i++)**

**data[i]=check\_data[i-data\_len];**

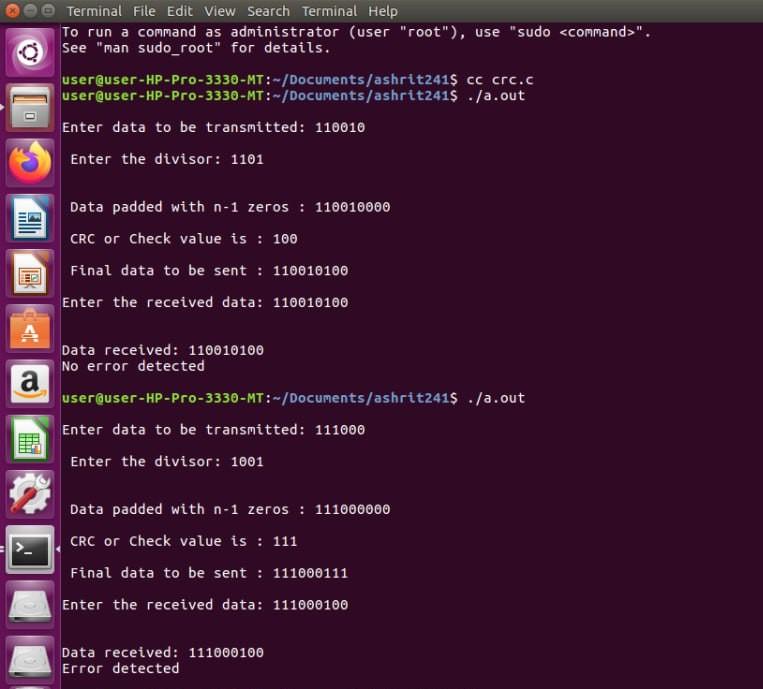
**printf("\nFinal data to be sent: %s\n",data);**

**receiver();**

**return 0;**

**}**

* 1. **Results:** (Screen Shots):

****

* 1. **Conclusion of the experiment:**

Error occurs when the information received by the receiver does not match the information sent by the sender. When digital signals are transmitted, they are susceptible to noise, which can lead to changes in binary bits. For instance, a 0 bit might be altered to 1, or a 1 bit might be flipped to 0.

Data, which is implemented either at the Data Link layer or Transport Layer of the OSI Model, can be distorted by noise during transmission, resulting in corruption. To mitigate such errors, error-detection codes are included as additional data within digital messages. These codes serve to identify any errors that may have occurred during the transmission of the message.

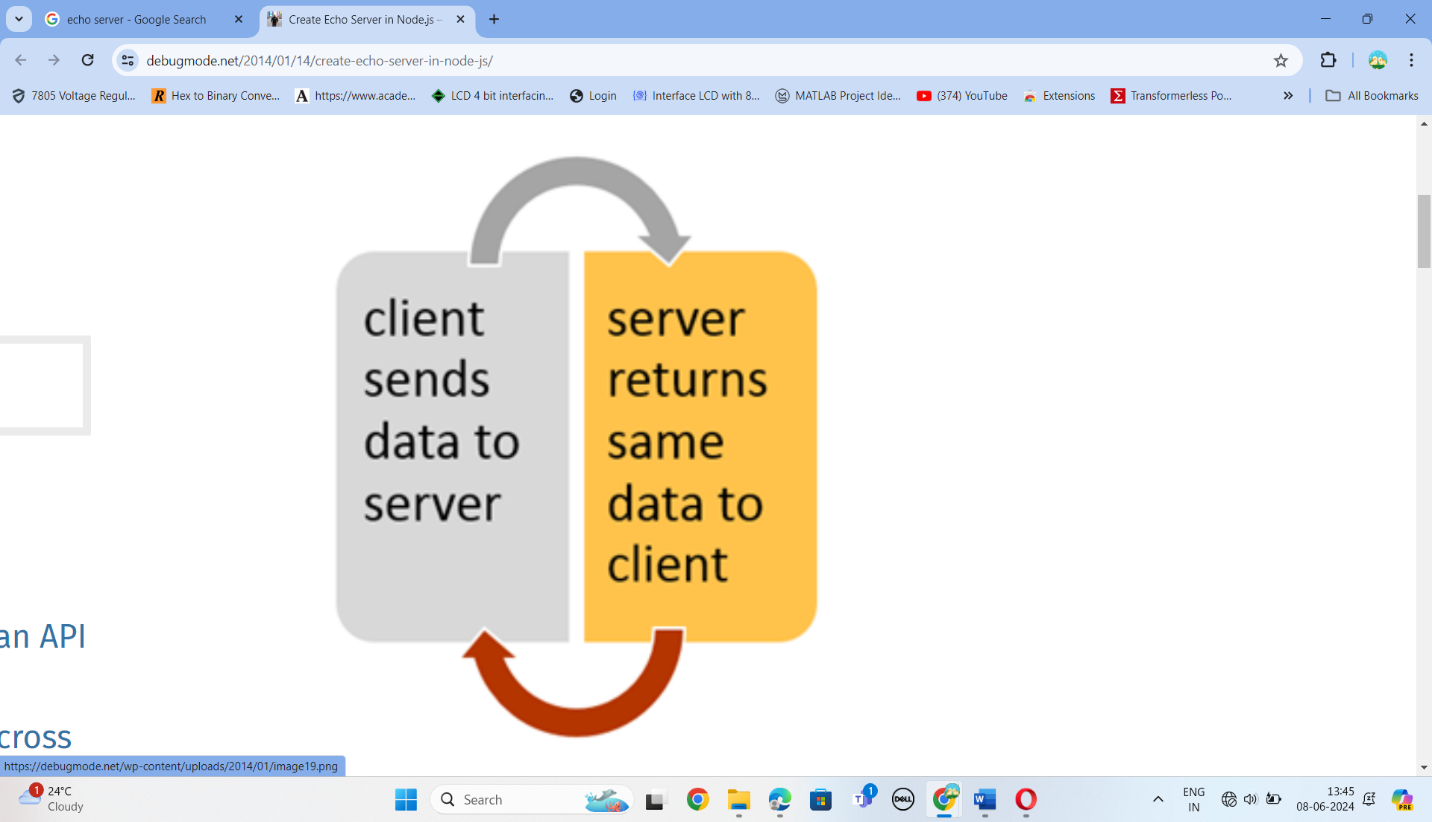
**Experiment No: 3**

* 1. **Title of the experiment: Echo Server**
  2. **Aim of the experiment: Echo server in one or more devices.**

**3.3 Theoretical background for the experiment:**

An Echo Server is an application that allows a client and a server to connect so a client can send a message to the server and the server can receive the message and send, or echo, it back to the client.

* 1. **Block Diagram:**



* 1. **Step by step procedure to carry out the experiment:**

We will briefly describe the functions used:

* socket: This function is used to create a socket which is used later for reading and writing from/to network.
* bind: This function binds the created socket with an IP address and port on the server, for the port we chose 1234 and the IP address used is **INADDR\_ANY** which means you can use any IP address on the server to receive new clients.
* listen: This function instructs the socket to listen to new connection requests with a backlog parameter that defines the number of allowed pending sockets before new connections are refused.
* accept: This function returns a new socket that can be used to serve new clients
* read/write: These functions are used to read data from a socket or write data to it.
* close: This function closes the connection and the socket.
  1. **Codes and commands:**

**Server**

**#include <stdio.h>**

**#include <sys/types.h>**

**#include <sys/socket.h>**

**#include <netinet/in.h>**

**#include<arpa/inet.h>**

**#include<stdlib.h>**

**#include<unistd.h>**

**#define BUFLEN 1024 /\* buffer length \*/**

**int main(int argc, char \*\*argv)**

**{**

**int n;**

**int yes=1;**

**int sd, new\_sd, client\_len, port;**

**struct sockaddr\_in server, client;**

**char buf[BUFLEN];**

**port = atoi(argv[1]);**

**/\* create a stream socket \*/**

**if((sd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)**

**{**

**fprintf(stderr,"can't create a socket\n");**

**exit(1);**

**}**

**/\* Fill the structure fileds with values \*/**

**server.sin\_family = AF\_INET;**

**server.sin\_port = port;**

**server.sin\_addr.s\_addr =inet\_addr("127.0.0.1");**

**// Reuse the port and address**

**if (setsockopt(sd, SOL\_SOCKET, SO\_REUSEADDR, &yes, sizeof(yes)) == -1) {**

**perror("setsockopt");**

**exit(1);**

**}**

**/\* bind an address to the socket \*/**

**if(bind(sd, (struct sockaddr \*)&server, sizeof(server)) == -1)**

**{**

**fprintf(stderr, "can't bind name to socket\n");**

**exit(1);**

**}**

**/\* queue up to 5 connect requests \*/**

**listen(sd,5);**

**while(1)**

**{**

**client\_len = sizeof(client);**

**if((new\_sd = accept(sd, (struct sockaddr \*) &client, &client\_len)) == -1)**

**{**

**fprintf(stderr, "can't accept client \n");**

**exit(1);**

**}**

**n = read(new\_sd, buf, sizeof(buf));**

**printf("The message received by client : %s\n",buf);**

**write(new\_sd, buf,n);**

**close(new\_sd);**

**}**

**close(sd);**

**return(0);**

**}**

**Client**

**#include <stdio.h>**

**#include <netdb.h>**

**#include <sys/types.h>**

**#include <sys/socket.h>**

**#include <netinet/in.h>**

**#include<arpa/inet.h>**

**#include<stdlib.h>**

**#include<unistd.h>**

**#define BUFLEN 1024 /\* buffer length \*/**

**int main(int argc, char \*\*argv)**

**{**

**int n;**

**int sd, port;**

**char buf[BUFLEN];**

**struct sockaddr\_in server;**

**port=atoi(argv[1]);**

**/\* create a stream socket \*/**

**if(( sd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)**

**{**

**fprintf(stderr, "can't create a socket\n");**

**exit(1);**

**}**

**// bzero((char \*)&server, sizeof(struct sockaddr\_in));**

**server.sin\_family = AF\_INET;**

**server.sin\_port = port;**

**server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");**

**/\* connecting to the server \*/**

**if(connect(sd, (struct sockaddr \*)&server, sizeof(server)) == -1)**

**{**

**fprintf(stderr, "can't connect\n");**

**exit(1);**

**}**

**printf("Enter the message to be echoed: ");**

**scanf("%s",buf); /\* get user's text \*/**

**write(sd, buf, BUFLEN); /\* send it out \*/**

**printf("Echoed Messege:\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**n = read(sd, buf, sizeof(buf));**

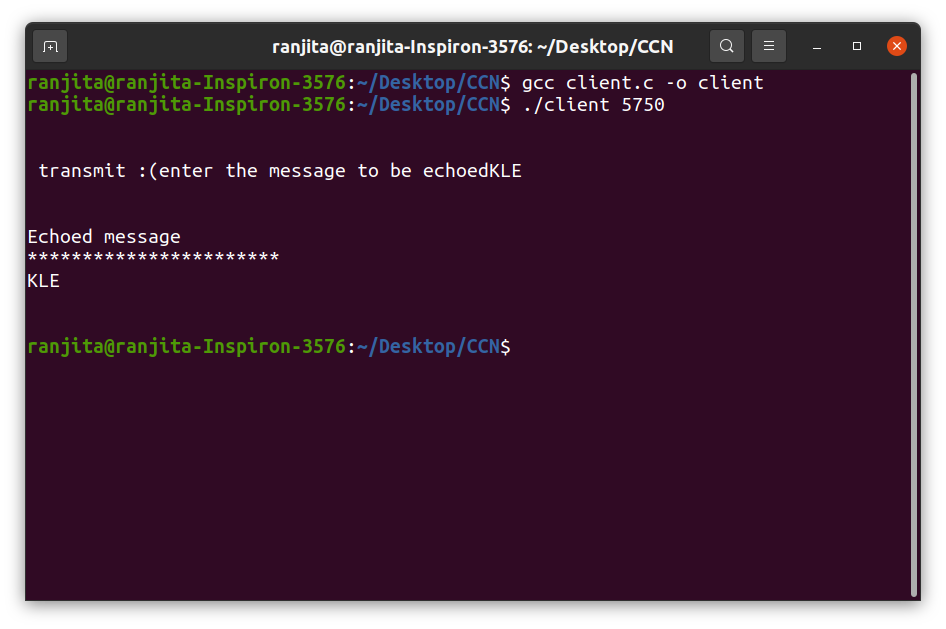
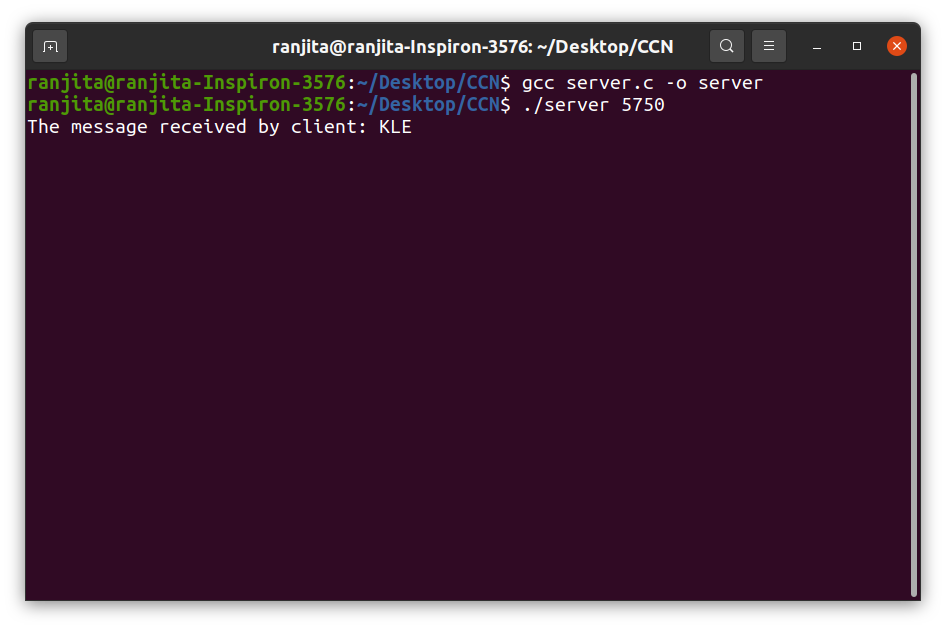
**printf("%s\n",buf);**

**close(sd);**

**return(0);**

**}**

* 1. **Results:** (Screen Shots):



* 1. **Conclusion of the experiment**:

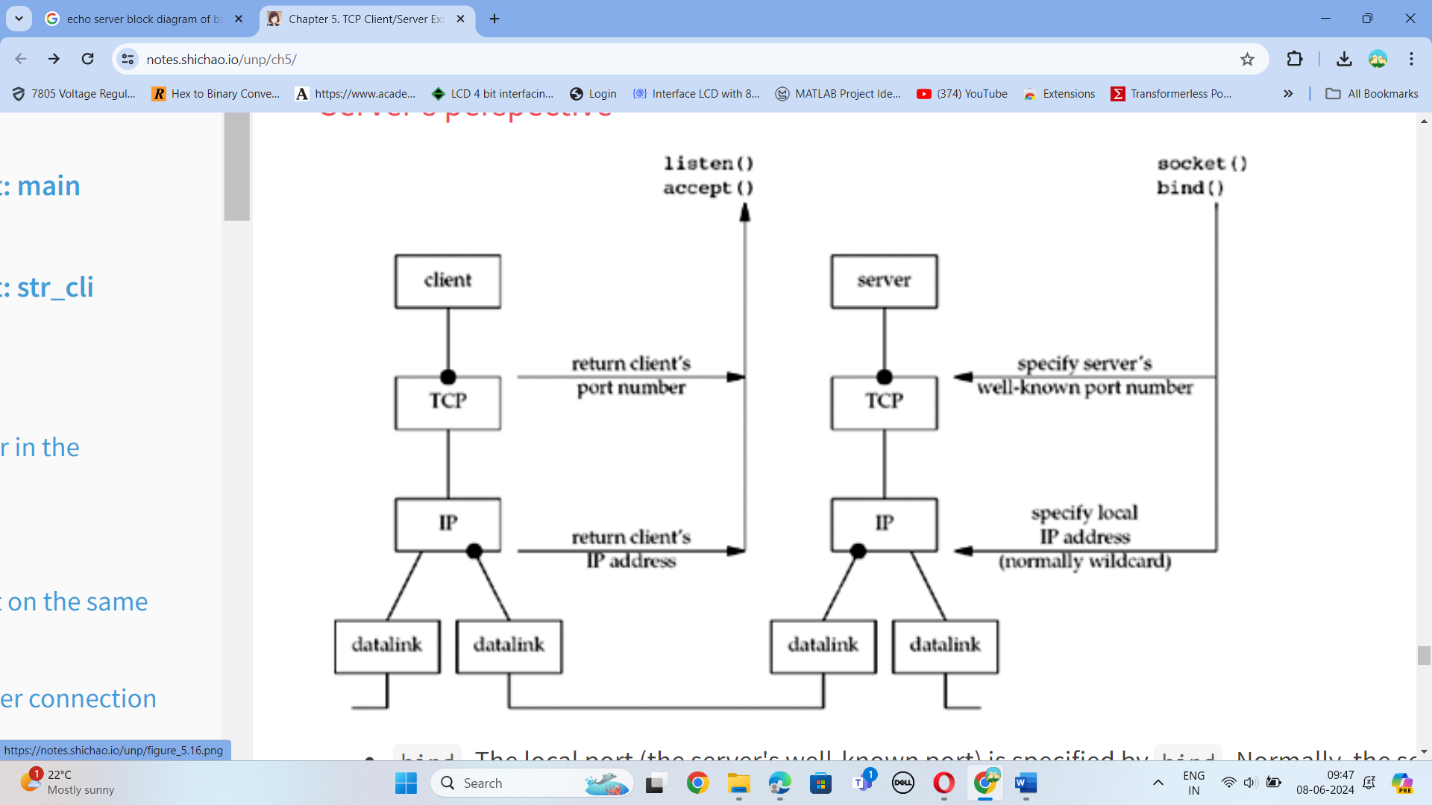
It can be implemented both to connect one or more devices to share the data into one another.

**Experiment No: 4**

* 1. **Title of the experiment: Chat server**
  2. **Aim of the experiment: Implementation of Chat server between two or more devices.**
  3. **Theoretical background for the experiment:**

System calls that allow to access the network functionality of a Unix box are as given  
below. When you call one of these functions, the kernel takes over and does all the  
work for you automatically.  
**Server Side:**  
socket () bind()    connect() listen() accept() send() recv()   close()  
**Client Side:**  
socket () gethostbyname()       connect()   send() recv() close()

* 1. **Block Diagram:**



* 1. **Step by step procedure to carry out the experiment:** **Algorithm**  
     **TCP Server**  
     1. Create a socket  
     2. Bind it to the operating system.  
     3. Listen over it.  
     4. Accept connections.  
     5. Receive data from client and send it back to client.  
     6. Close the socket.  
     **TCP Client**

1.Create a socket.  
2. connect to the server using connect ().  
3. send data to server and receive data from the server.  
4.Close the socket.

* 1. **C Program/Commands:**

**Chat Server**

**#include <stdio.h>**

**#include <sys/types.h>**

**#include <sys/socket.h>**

**#include <netinet/in.h>**

**#include<arpa/inet.h>**

**#include<stdlib.h>**

**#include<unistd.h>**

**#include<string.h>**

**#define SERVER\_TCP\_PORT 5750 /\* well known port \*/**

**#define BUFLEN 256 /\* buffer length \*/**

**#define MAX 80**

**int flag=0;**

**int func(int sockfd)**

**{**

**char buff[MAX];**

**int n;**

**for(;;)**

**{**

**if(flag==1)**

**break;**

**bzero(buff,MAX);**

**n=read(sockfd,buff,sizeof(buff));**

**printf("Message from client is:%s",buff);**

**bzero(buff,MAX);**

**n=0;**

**//while((buff[n++]=getchar())!='\n');**

**printf("Enter message to be sent to client:\n");**

**fgets(buff,sizeof(buff),stdin);**

**n=strlen(buff);**

**if(strncmp("exit",buff,4)==0)**

**{**

**printf("Server Exit ...\n");**

**flag=1;**

**break;**

**}**

**else**

**{**

**write(sockfd,buff,sizeof(buff));**

**bzero(buff,MAX);**

**}**

**} // for loop**

**}**

**int main(int argc, char \*\*argv)**

**{**

**int n;**

**int yes=1;**

**int sd, new\_sd, client\_len, port;**

**struct sockaddr\_in server, client;**

**char buff[BUFLEN];**

**port = atoi(argv[1]);**

**// port=5750;**

**/\* create a stream socket \*/**

**if((sd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)**

**{**

**fprintf(stderr,"can't create a socket\n");**

**exit(1);**

**}**

**/\* bind an address to the socket \*/**

**// bzero((char \*)&server, sizeof(struct sockaddr\_in));**

**server.sin\_family = AF\_INET;**

**server.sin\_port = port;**

**server.sin\_addr.s\_addr =htonl(INADDR\_ANY);**

**if (setsockopt(sd, SOL\_SOCKET, SO\_REUSEADDR, &yes, sizeof(yes)) == -1) {**

**perror("setsockopt");**

**exit(1);**

**}**

**if(bind(sd, (struct sockaddr \*)&server, sizeof(server)) == -1)**

**{**

**fprintf(stderr, "can't bind name to socket\n");**

**exit(1);**

**}**

**/\* queue up to 5 connect requests \*/**

**listen(sd,5);**

**while(1)**

**{**

**client\_len = sizeof(client);**

**if((new\_sd = accept(sd, (struct sockaddr \*) &client, &client\_len)) == -1)**

**{**

**fprintf(stderr, "can't accept client\n");**

**exit(1);**

**}**

**func(new\_sd);**

**close(new\_sd);**

**}**

**close(sd);**

**return(0);**

**}**

**Chat client**

**#include <stdio.h>**

**#include <netdb.h>**

**#include <sys/types.h>**

**#include <sys/socket.h>**

**#include <netinet/in.h>**

**#include<arpa/inet.h>**

**#include<stdlib.h>**

**#include<unistd.h>**

**#include<string.h>**

**#define BUFLEN 256 /\* buffer length \*/**

**#define MAX 80**

**void func(int sockfd)**

**{**

**char buff[MAX];**

**int n;**

**for(;;)**

**{**

**bzero(buff,sizeof(buff));**

**printf("Enter the message to be sent: ");**

**n=0;**

**fgets(buff,sizeof(buff),stdin);**

**if((strncmp(buff,"exit",4))==0)**

**{**

**printf("Client Exit...\n");**

**break;**

**}**

**n=strlen(buff);**

**write(sockfd,buff,n);**

**bzero(buff,sizeof(buff));**

**read(sockfd,buff,sizeof(buff));**

**printf("Message from Server : %s",buff);**

**}**

**}**

**int main(int argc, char \*\*argv)**

**{**

**int n;**

**int sd, port;**

**char buff[BUFLEN];**

**struct sockaddr\_in server;**

**//command line argument**

**port=atoi(argv[1]);**

**/\* create a stream socket \*/**

**if(( sd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)**

**{**

**fprintf(stderr, "can't create a socket\n");**

**exit(1);**

**}**

**// bzero((char \*)&server, sizeof(struct sockaddr\_in));**

**server.sin\_family = AF\_INET;**

**server.sin\_port = port;**

**server.sin\_addr.s\_addr = inet\_addr("10.3.1.142");**

**/\* connecting to the server \*/**

**if(connect(sd, (struct sockaddr \*)&server, sizeof(server)) == -1)**

**{**

**fprintf(stderr, "can't connect\n");**

**exit(1);**

**}**

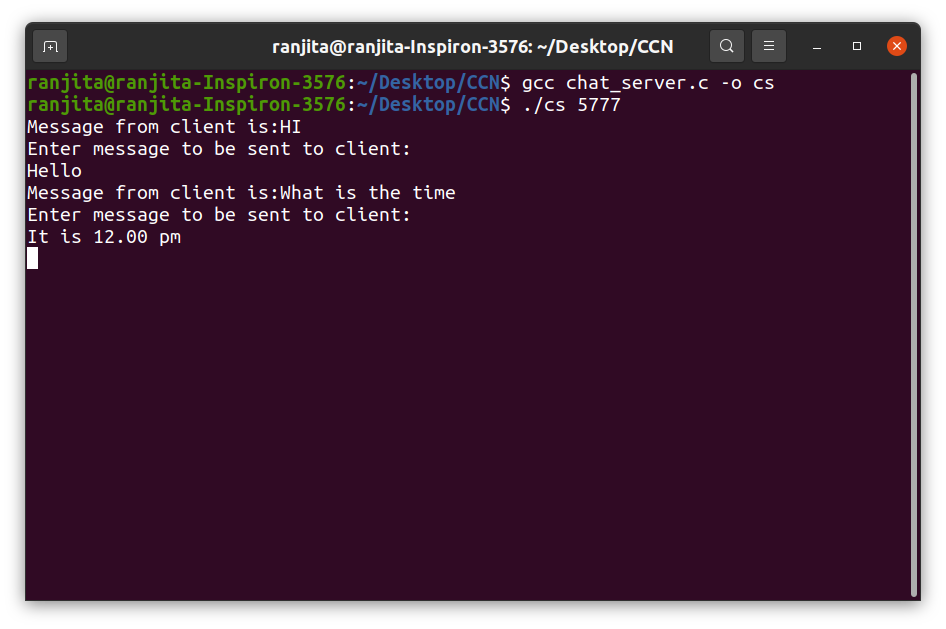
**func(sd);**

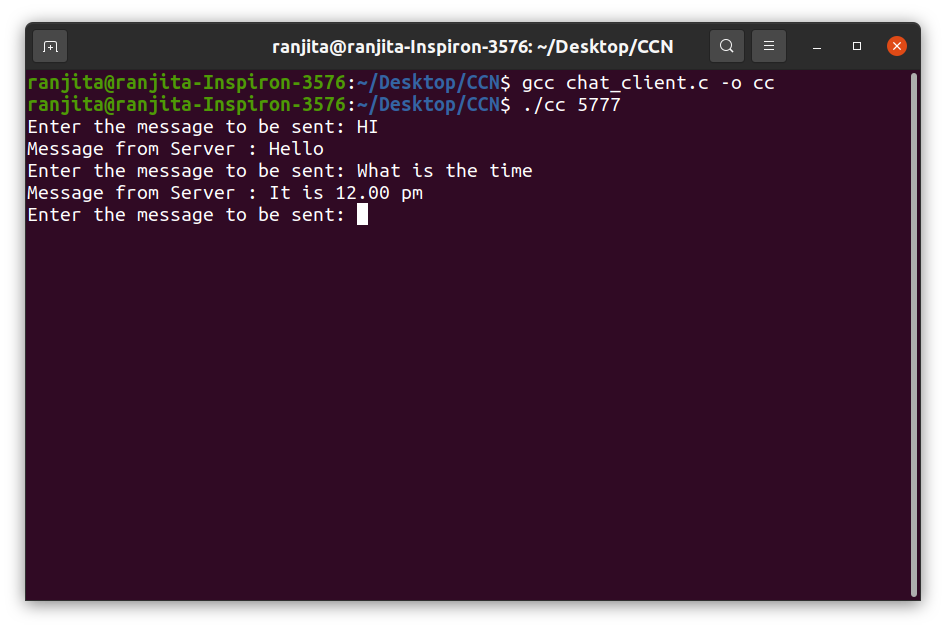
**close(sd);**

**return(0);**

**}**

* 1. **Results:** (Screen Shots):





* 1. **Conclusion of the experiment:**

The client-server model, or client-server architecture, is a distributed application framework dividing tasks between servers and clients, which either reside in the same system or communicate through a computer network or the Internet. The client relies on sending a request to another program to access a service made available by a server. The server runs one or more programs that share resources with and distribute work among clients.

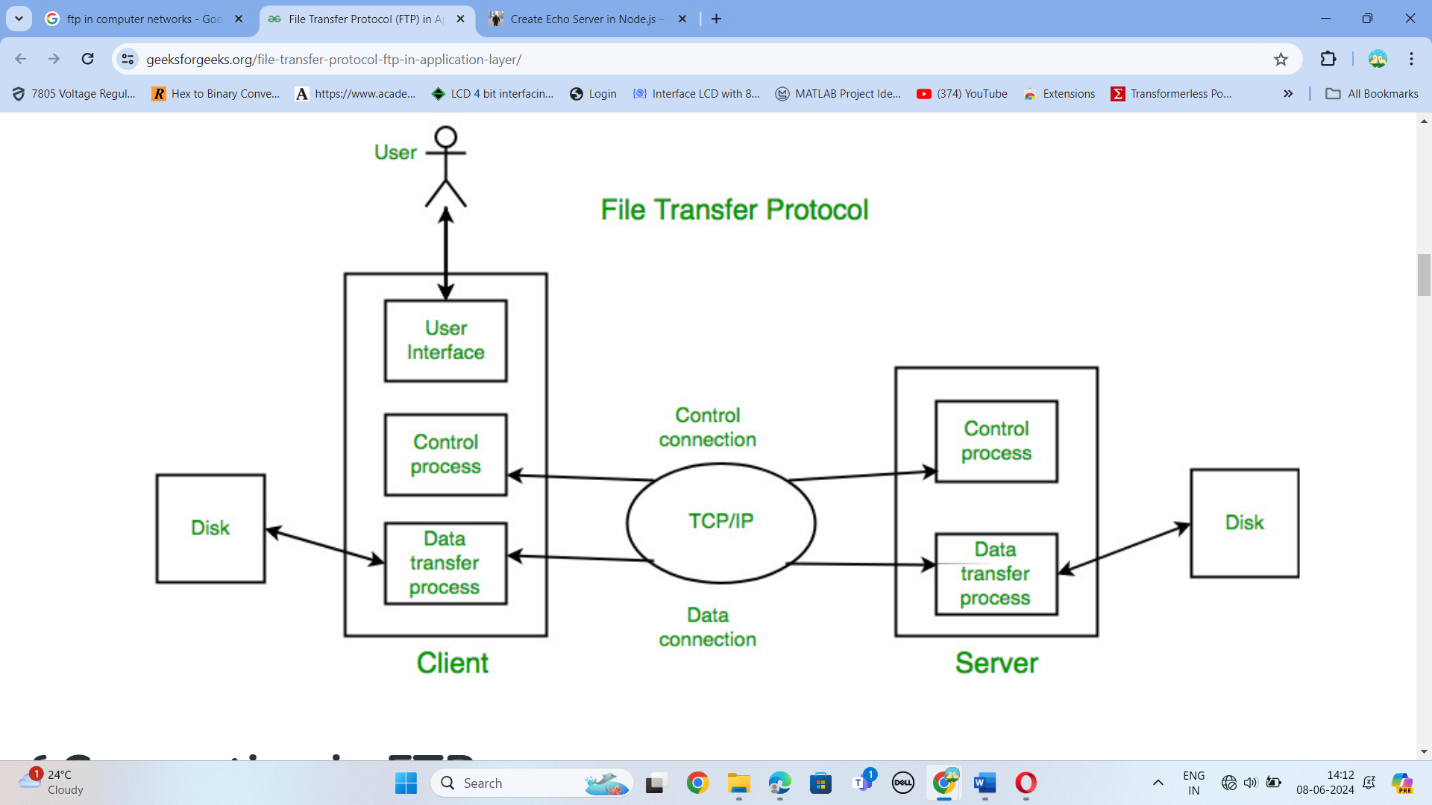
The client server relationship communicates in a request–response messaging pattern and must adhere to a common communications protocol, which formally defines the rules, language, and dialog patterns to be used. Client-server communication typically adheres to the TCP/IP protocol suite.

**Experiment No: 5**

* 1. **Title of the experiment: FTP**
  2. **Aim of the experiment: Implementation of FTP.**
  3. **Theoretical background for the experiment:**

**FTP (File Transfer Protocol)** is a standard network protocol used for the transfer of files from one host to another over a TCP-based network, such as the Internet. FTP works by opening two connections that link the computers trying to communicate with each other. One connection is designated for the commands and replies that get sent between the two clients, and the other channel handles the transfer of data. During an FTP transmission, there are four commands used by the computers, servers, or proxy servers that are communicating. These are “send,” “get,” “change directory,” and “transfer.”

* 1. **Block Diagram:**



**5.4 Step by step procedure to carry out the experiment:**

### **Control Connection**

For sending control information like user identification, password, commands to change the remote directory, commands to retrieve and store files, etc., FTP makes use of a control connection. The control connection is initiated on port number 21.

### **Data connection**

For sending the actual file, FTP makes use of a data connection. A data connection is initiated on port number 20. FTP sends the control information out-of-band as it uses a separate control connection. Some protocols send their request and response header lines and the data in the same TCP connection. For this reason, they are said to send their control information in-band. HTTP and SMTP are such examples.

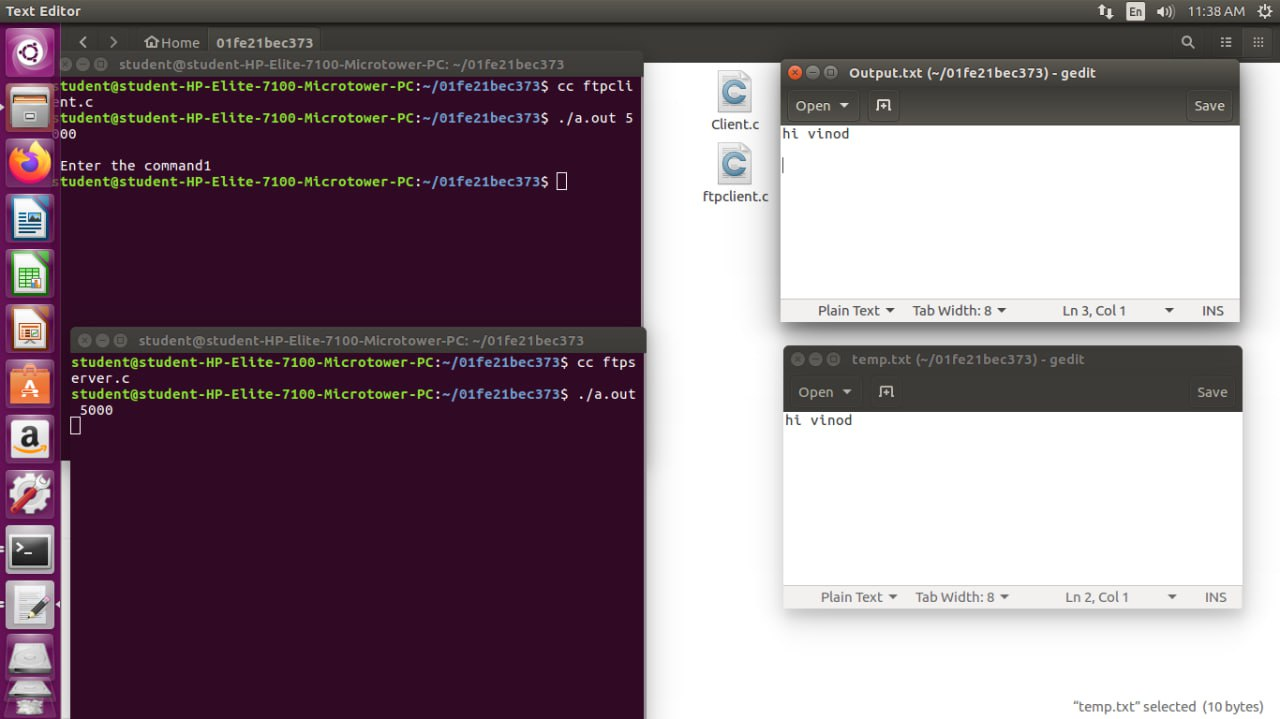
**ALGORITHM:  
SERVER:**  
    STEP 1: Start  
    STEP 2: Declare the variables for the socket  
    STEP 3: Specify the family, protocol, IP address and port number  
    STEP 4: Create a socket using socket() function  
    STEP 5: Bind the IP address and Port number  
    STEP 6: Listen and accept the client’s request for the connection  
    STEP 7: Establish the connection with the client  
    STEP 8: Close the socket  
    STEP 9: Stop  
**CLIENT:**  
    STEP 1: Start  
    STEP 2: Declare the variables for the socket  
    STEP 3:  Specify the family, protocol, IP address and port number  
    STEP 4: Create a socket using socket() function  
    STEP 5: Call the connect() function  
    STEP 6: Close the socket  
    STEP 7: Stop

* 1. **C Program/Commands:**

|  |
| --- |
|  | #include<stdio.h>  #include <sys/socket.h> |
|  | #include <arpa/inet.h> |
|  | #include <sys/stat.h> |
|  | #include <sys/sendfile.h> |
|  | #include <fcntl.h> |
|  | #include <stdlib.h> |
|  | #include <string.h> |
|  |  |
|  | #define FILENAME "a.txt" |
|  | #define SERVER\_IP "127.0.0.1" |
|  | #define SERVER\_PORT 65496 |
|  |  |
|  | int main(int argc , char \*\*argv) |
|  | { |
|  | int socket\_desc; |
|  | struct sockaddr\_in server; |
|  | char request\_msg[BUFSIZ], |
|  | reply\_msg[BUFSIZ]; |
|  |  |
|  | // Variables for the file being received |
|  | int file\_size, |
|  | file\_desc; |
|  | char \*data; |
|  |  |
|  | socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0); |
|  | if (socket\_desc == -1) |
|  | { |
|  | perror("Could not create socket"); |
|  | return 1; |
|  | } |
|  |  |
|  | server.sin\_addr.s\_addr = inet\_addr(SERVER\_IP); |
|  | server.sin\_family = AF\_INET; |
|  | server.sin\_port = htons(SERVER\_PORT); |
|  |  |
|  | // Connect to server |
|  | if (connect(socket\_desc, (struct sockaddr \*)&server, sizeof(server)) < 0) |
|  | { |
|  | perror("Connection failed"); |
|  | return 1; |
|  | } |
|  |  |
|  | // Get a file from server |
|  | strcpy(request\_msg, "Get "); |
|  | strcat(request\_msg, FILENAME); |
|  | write(socket\_desc, request\_msg, strlen(request\_msg)); |
|  | recv(socket\_desc, reply\_msg, 2, 0); |
|  |  |
|  | // Start receiving file |
|  | if (strcmp(reply\_msg, "OK") == 0) { |
|  |  |
|  | recv(socket\_desc, &file\_size, sizeof(int), 0); |
|  | data = malloc(file\_size); |
|  | file\_desc = open(FILENAME, O\_CREAT | O\_EXCL | O\_WRONLY, 0666); |
|  | recv(socket\_desc, data, file\_size, 0); |
|  | write(file\_desc, data, file\_size); |
|  | close(file\_desc); |
|  | } |
|  | else { |
|  |  |
|  | fprintf(stderr, "Bad request\n"); |
|  | } |
|  |  |
|  | return 0; |
|  | } |

**SERVER:**  
#include<stdio.h>  
#include<arpa/inet.h>  
#include<sys/types.h>  
#include<sys/socket.h>  
#include<netinet/in.h>  
#include<netdb.h>  
#include<stdlib.h>  
#include<string.h>  
#include<unistd.h>  
#define SERV\_TCP\_PORT 5035  
#define MAX 60  
int i, j, tem;  
char buff[4096], t;  
FILE \*f1;  
int main(int afg, char \*argv)  
{  
       int sockfd, newsockfd, clength;  
       struct sockaddr\_in serv\_addr,cli\_addr;  
       char t[MAX], str[MAX];  
       strcpy(t,"exit");  
       sockfd=socket(AF\_INET, SOCK\_STREAM,0);  
       serv\_addr.sin\_family=AF\_INET;  
       serv\_addr.sin\_addr.s\_addr=INADDR\_ANY;  
       serv\_addr.sin\_port=htons(SERV\_TCP\_PORT);  
       printf("\nBinded");  
       bind(sockfd,(struct sockaddr\*)&serv\_addr, sizeof(serv\_addr));  
       printf("\nListening...");  
       listen(sockfd, 5);  
       clength=sizeof(cli\_addr);  
       newsockfd=accept(sockfd,(struct sockaddr\*) &cli\_addr,&clength);  
       close(sockfd);  
       read(newsockfd, &str, MAX);  
       printf("\nClient message\n File Name : %s\n", str);  
       f1=fopen(str, "r");  
       while(fgets(buff, 4096, f1)!=NULL)  
       {  
            write(newsockfd, buff,MAX);  
            printf("\n");  
       }  
       fclose(f1);  
       printf("\nFile Transferred\n");  
       return 0;  
}

* 1. **Results:** (Screen Shots):



* 1. **Conclusion of the experiment:**

File transfer protocol server (commonly known as FTP Server) is computer software that facilitates the secure exchange of files over a TCP/IP network. It runs the file transfer protocol (FTP), a standard communication protocol that operates at the network level, to establish a secure connection between the devices in a client-server architecture and efficiently transmit data over the internet

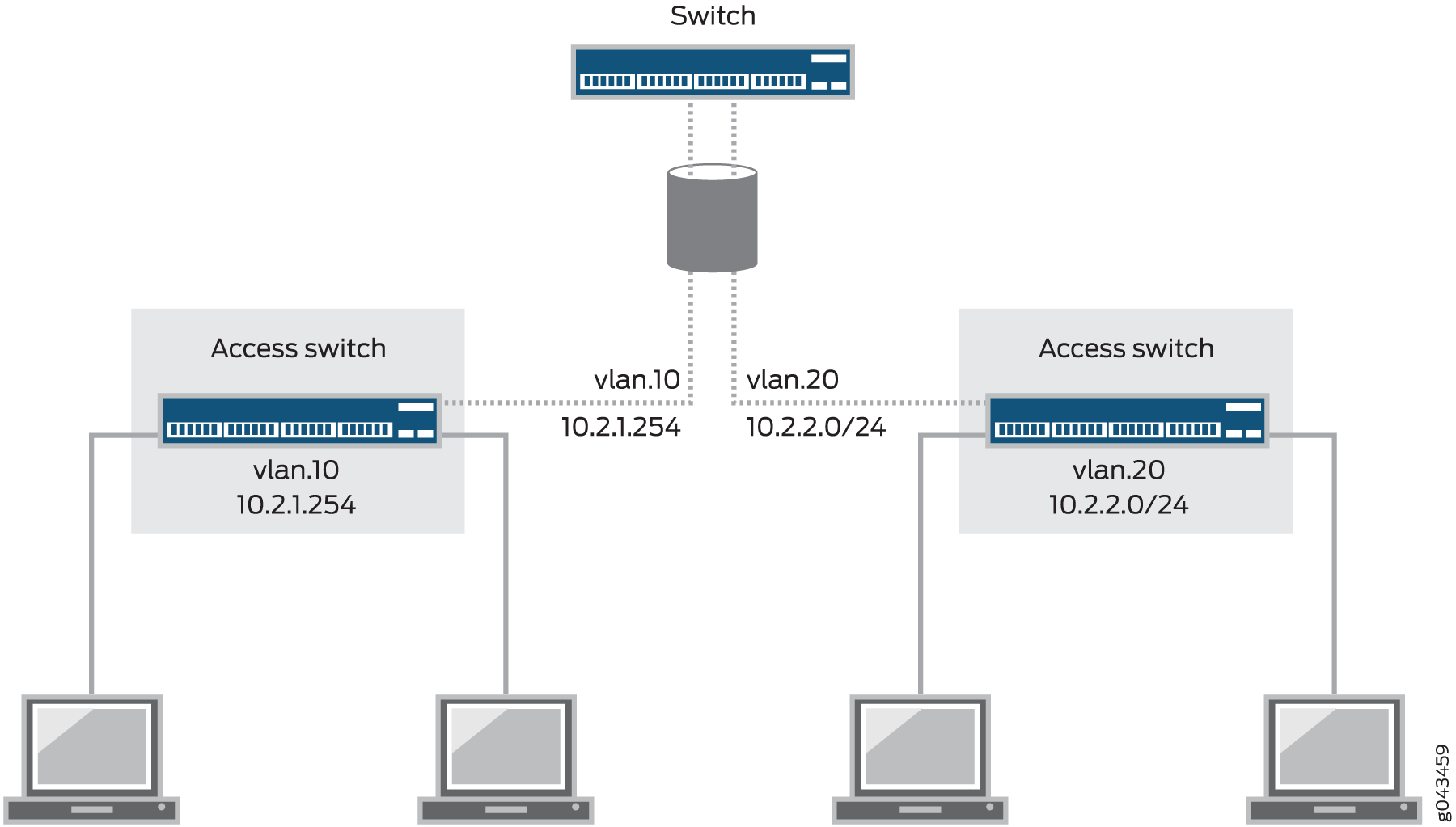
**Experiment No: 6**

* 1. **Title of the experiment: VLAN Setup 1**
  2. **Aim of the experiment: Setting up VLAN using Switch**

**6.3 Theoretical background for the experiment:**

A VLAN is a logical grouping of devices within a local area network (LAN) that communicate with each other as if they were on the same physical network, regardless of their physical location. VLANs are used to enhance network security, improve performance, and simplify network administration. A broadcast domain is a logical division of a computer network where all devices can directly send broadcast messages to each other. In a traditional LAN, all devices connected to the same physical network segment belong to the same broadcast domain. VLANs help create multiple broadcast domains within a single physical network. Devices within the same VLAN can communicate with each other directly without the need for routing. However, for communication between VLANs, a routing device (such as a Layer 3 switch or router) is required. Inter-VLAN communication can be achieved through various methods, including router-on-a-stick, Layer 3 switches, or virtual routing and forwarding (VRF).

* 1. **Block Diagram:**



* 1. **Step by step procedure to carry out the experiment:**
* Identify VLAN Requirements
* Access the Switch Configuration Interface
* Create VLANs
* Assign Ports to VLANs
* Verify VLAN Configuration
* Test VLAN Connectivity
  1. **C Program/Commands:**

set vlans ccn vlan-id 55

set vlans vlsi vlan-id 66

set interfaces ge-0/0/6 unit 0 family ethernet-switching vlan members ccn

set interfaces ge-0/0/10 unit 0 family ethernet-switching vlan members vlsi

set interfaces vlan unit 55 family inet address 192.168.10.1/24

set interfaces vlan unit 66 family inet address 30.30.10.1/24

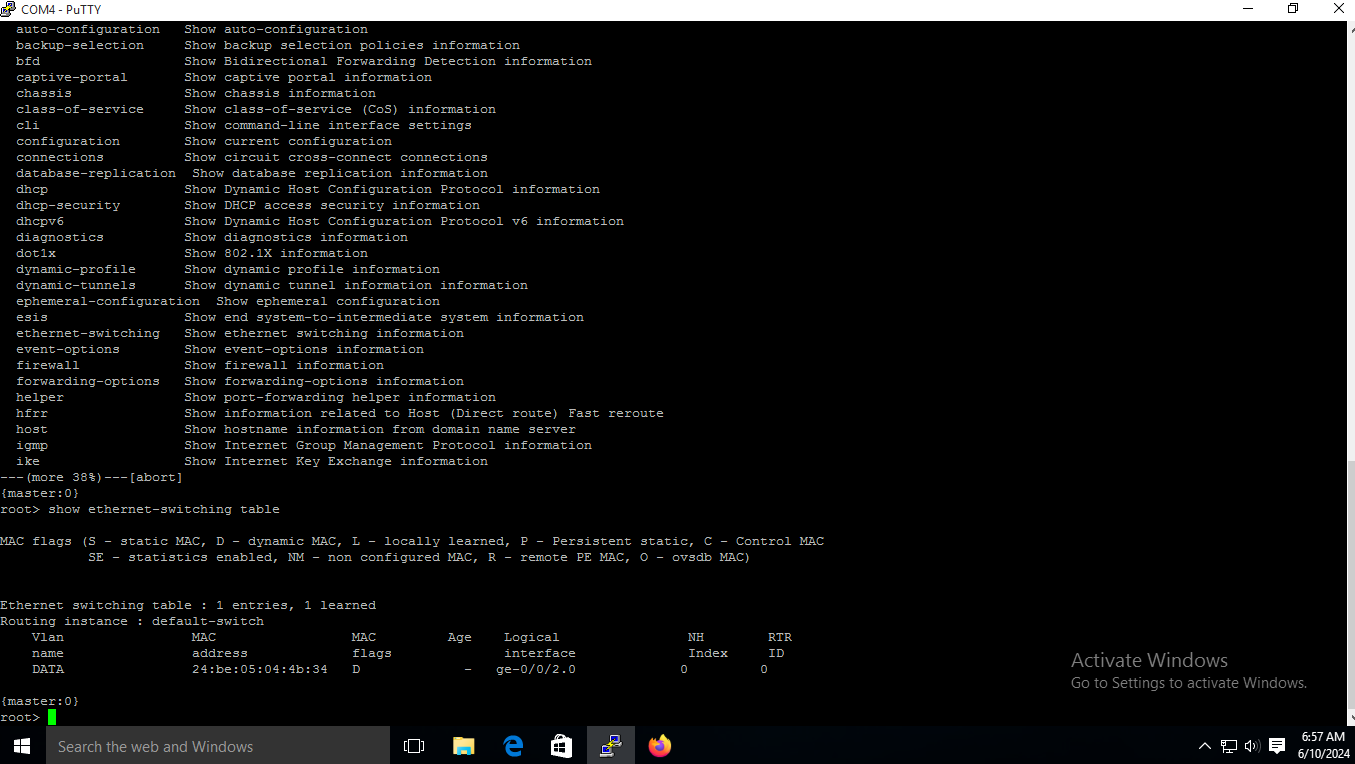
set vlans ccn l3-interface irb.55

set vlans vlsi l3-interface irb.66

set interfaces ge-0/0/23 unit 0 family init addres 20.20.20.1/30

set routing -options static route 0.0.0.0/0 next \_hop 20.20.20.2

* 1. **Result:(Screen shot)**

****

* 1. **Conclusion of the experiment:**

The VLAN setup experiment provided hands-on experience in configuring and implementing VLANs within a network infrastructure. It highlighted the importance of proper VLAN configuration, VLAN membership assignment and inter-VLAN communication.

**Experiment No: 7**

* 1. **Title of the experiment: Open Shortest Path First** (**OSPF)**

# Aim of the experiment: Implementation of Open Shortest Path First (OSPF) protocol States.

**7.3 Theoretical background for the experiment:**

Open shortest path first (OSPF) is a **link-state routing protocol** that is used to find the best path between the source and the destination router using its own shortest path first (SPF) algorithm. A link-state routing protocol is a protocol that uses the concept of triggered updates, i.e., if there is a change observed in the learned routing table then the updates are triggered only, not like the distance-vector routing protocol where the routing table is exchanged at a period of time.

Open shortest path first (OSPF) is developed by Internet Engineering Task Force (IETF) as one of the Interior Gateway Protocols (IGP), i.e., the protocol which aims at moving the packet within a large autonomous system or routing domain. It is a **network layer protocol** that works on protocol number 89 and uses AD value 110. OSPF uses multicast address 224.0.0.5 for normal communication and 224.0.0.6 for updating to the designated router (DR)/Backup Designated Router (BDR).

* 1. **Block Diagram:**







* 1. **Step by step procedure to carry out the experiment:**
* Access the CLI
* Basic Settings
* Enter OSPF Configuration Mode
* Assign Router ID
* Commit the Configuration
* Check OSPF Neighbors
* Check OSPF Routes
* Verify OSPF Database
* Ping Test
* Debugging

**7.6 C Program/Commands:**

**DEVICE 1:**

set interfaces ge-0/0/0 unit 0 family inet address

10.10.10.1/24

set interfaces ge-0/0/1 unit 0 family inet address

10.10.10.2/24

set interfaces ge-0/0/2 unit 0 family ethernet-switching vlan members VOICE

set interfaces vian unit 10 family inet address 192.168.1.1/24

set protocols ospf area 0.0.0.10 interface ge-0/0/0.0 metric 5

set protocols ospf area 0.0.0.10 interface ge-0/0/1.0 metric 10

set protocols ospf area 0.0.0.10 interface irb. 10

set vlans VOICE vlan-id 10

set vlans VOICE 13-interface irb.10

set interfaces irb.10 family inet address

192.168.1.1/24

**DEVICE 2:**

set interfaces ge-0/0/0 unit 0 family inet address

10.10.10.8/24

set interfaces ge-0/0/1 unit 0 family inet address

10.10.10.7/24

set interfaces ge-0/0/2 unit 0 family ethernet-switching vlan members DATA

set interfaces vlan unit 20 family inet address

172.16.1.1/24

set protocols ospf area 0.0.0.10 interface ge-0/0/0.0

metric 5

set protocols ospf area 0.0.0.10 interface ge-0/0/1.0

AS 10.10.10.8/24

set interfaces ge-0/0/1 unit @ family inet address

10.10.10.7/24

set interfaces ge-0/0/2 unit 0 family ethernet-switching vlan members DATA

set interfaces vlan unit 20 family inet address

172.16.1.1/24

set protocols ospf area 0.0.0.10 interface ge-0/0/0.0

metric 5

set protocols ospf area 0.0.0.10 interface ge-0/0/1.0

metric 10

set protocols ospf area 0.0.0.10 interface irb.20

set vlans DATA vlan-id 20

set vlans DATA 13-interface irb.20

set interfaces irb.20 family inet address

172.16.1.1/24

**7.7 Results:** (Screen Shots):

A computer screen with white text

Description automatically generated

A computer screen with white text

Description automatically generated

**7.8 Conclusion of the experiment:**

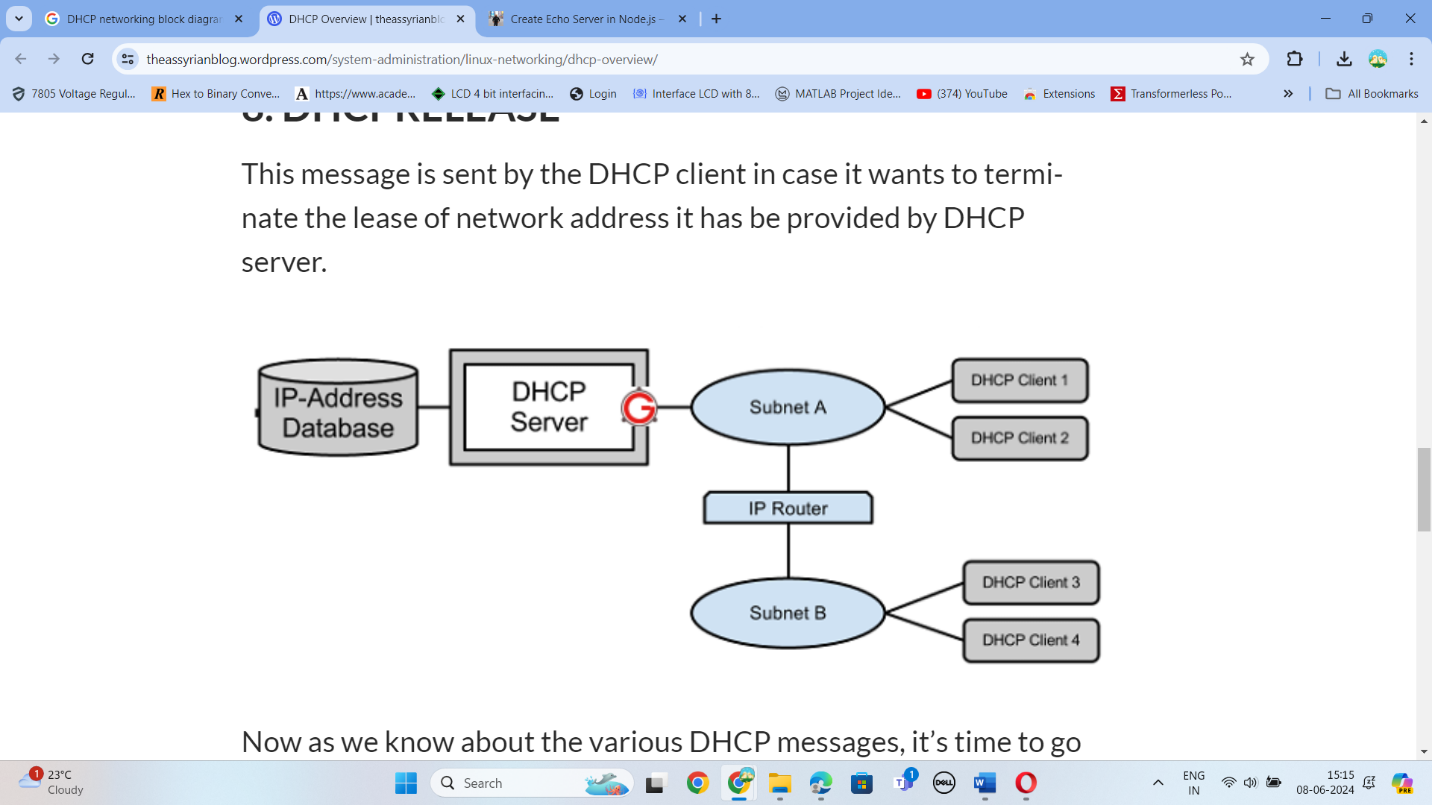
All the routers in the same area of the OSPF network maintain the same link-state database that describes the area topology. Each router receives link-state advertisement (LSA) messages containing information about neighboring routers and path costs from the other routers in that area. Using these LSAs, each router generates the link-state database and uses the SPF algorithm

**Experiment No: 8**

* 1. **Title of the experiment: Dynamic Host Configuration Protocol (DHCP)**
  2. **Aim of the experiment: Implementation of Dynamic Host Configuration Protocol**
  3. **Theoretical background for the experiment:**

Dynamic Host Configuration Protocol (DHCP) is a client/server protocol that automatically provides an Internet Protocol (IP) host with its IP address and other related configuration information such as the subnet mask and default gateway. RFCs 2131 and 2132 define DHCP as an Internet Engineering Task Force (IETF) standard based on Bootstrap Protocol (BOOTP), a protocol with which DHCP shares many implementation details. DHCP allows hosts to obtain required TCP/IP configuration information from a DHCP server.

* 1. **Block Diagram:**



* 1. **Step by step procedure to carry out the experiment:**
* **Step 1: DHCP Discover:**

When you connect to a new device, it still does not have an IP address. It will search for an IP address. It will call over the network for a DHCP server. This request will arrive to all the devices, and the server will also get it.

* **Step 2 DHCP Offer:**

The DHCP hears the call, and answers with an IP address, which it offers to the newly connected device.

* **Step 3 DHCP Request:**

The IP address arrives at the device. The device will accept it and will send a request to use it.

* **Step 4 DHCP Pack:**

The server gets the accepting message from the device. It will provide the IP address to the device, together with the subnet mask and the DNS Server. It will write a record with the information of the newly connected device that usually includes the MAC address of the connected device, the IP address that was assigned, and the expiration date of that IP address. The DHCP leases the IP address for a limited time only. After the time passes, the IP address will go back to the IP pool of available IP addresses and can be assigned to a new device again.

* 1. **C Program/Commands:**

**set system services dhcp-local-server group V16 interface irb.16**

**set access address-assignment pool V16 family inet network 172.16.30.0/24**

**set access address-assignment pool V16 family inet range V16-RANGE low 172.16.30.10**

**set access address-assignment pool V16 family inet range V16-RANGE high 172.16.30.40**

**set access address-assignment pool V16 family inet dhcp-attributes maximum-lease-time 36000 set access address-assignment pool V16 family inet dhcp-attributes server-identifier**

**172.16.30.1**

**set access address-assignment pool V16 family inet dhcp-attributes name-server 8.8.8.8**

**set access address-assignment pool V16 family inet dhep-attributes router 172.16.30.1**

**set interfaces ge-0/0/10 unit 0 family ethernet-switching vlan members V16 set interfaces ge-0/0/9 unit 0 family ethernet-switching vlan members V16**

**set vlans V16 vlan-id 16**

**set vlans V16 13-interface irb.16**

**set interfaces irb.16 family inet address 172.16.30.2/24**

**set interfaces ge-0/0/8 unit 0 family ethernet-switching vlan members V16**

**AFTER DHCP EXPERIMENT**

**›clear dhep server binding all**

* 1. **Results:** (Screen Shots)

A computer screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen with white text

Description automatically generated

* 1. **Conclusion of the experiment:**

**IP address configuration on which you can rely**

DHCP makes very few errors regarding the IP address configuration. There might be some occasional errors related to the network typographic and IP conflicts when the DHCP server assigns the same IP to different devices.

calculate the shortest path spanning tree.

OSPF was designed and developed by the IETF for TCP/IP environments, mainly large enterprise networks. OSPF version 2 is defined in RFC 2328 of the IETF Network Working Group. This protocol is broadly implemented in enterprise routers. [IPv6](https://www.techtarget.com/searchnetworking/definition/IPv6-Internet-Protocol-Version-6) revisions to this standard are captured in OSPF version 3 and defined in IETF RFC 5340.