



**BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT
YELAHANKA – BANGALORE – 64**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Project Based Learning for the Academic Year 2016-2017

“FILE TRANSFER IN LINUX”

Under the Guidance

Ravi Kumar B N
Assistant Professor
Department of CSE

Team Members

Students Name	USN	Semester	Subject
SWATHI B R	1BY14CS077	6	UNIX SYSTEM PROGRAMMING
SWATHI N S	1BY14CS078		
TANUJA S	1BY14CS081		
TANUSHREE S	1BY14CS082		

Signature of Guide

Signature of HOD

INTRODUCTION:

BRIEF INTRODUCTION:

File sharing is the public or private sharing of computer data or space in a network with various levels of access privilege. While files can easily be shared outside a network (for example, simply by handing or mailing someone your file on a diskette), the term *file sharing* almost always means sharing files in a network, even if in a small local area network.

PROBLEMS OF EXISTING SYSTEMS

The problem with existing file sharing networks is that:

- you have no control over who you share information with.
- Bandwidth limitation for transferring large files.
- Only files can be shared but there are no private chats.
- It lacks security in sharing the files because there is no encrypted file sharing.
- Slow transfer between two internal/LAN PC's ,using latest Retroshare 0.4.05a on two pc's on same LAN they only transfer at about 43Kb/s.

MOTIVATION:

File sharing allows a number of people to use the same file or file by some combination of being able to read or view it, write to or modify it, copy it, or print it. File sharing can also mean having an allocated amount of personal file storage in a common file system.

To share the information, it is desirable to have a simple method of sharing the information with two personal computer systems. Here we are going to implement a network file-sharing application that makes sending a file to another machine on the local network as easy as dragging-and-dropping.

SCOPE:

The main principle is all about networking concepts. There is a receiver and a sender. One of them creates a Wi-Fi hotspot. And the other connects to it. Now a local network is formed. Both will have IP addresses. Now, sender can select a file and then click send. The file is transferred through TCP and the receiver is identified by the IP address .Now, for the high speed part ,the file is actually sent in chunks, and the progress is shown based on how much chunks are sent. Client-server architecture is used to transfer files over LAN/WAN.

STATEMENT OF PROBLEM:

Even though we have many protocols such as FTP, UDP, TCP, ICMP to send files through LAN/WAN, we need to know the best protocols for the situations, so we use the applications to decide which protocols to use.

LIMITATION OF PROJECT:

The system has only a limited scope like WIFI direct if it out bounds the bandwidth limit, the files cannot be transferred. Hence we must use the LAN connectivity to share the files. The system works on same operating system environment.

LITERATURE SURVEY:

NitroShare is a tool used to transfer files from one machine to another on the same network. Just install it on your Ubuntu systems and you will all set. NitroShare will instantly find each other systems in your network and start sharing files and folders. Nitroshare is written in C++ programming language, and it is completely free and open source. This application was developed using the QT framework and therefore runs on any platform supported by QT, including Linux, Microsoft Windows, and Mac OS X.

Retroshare is the next generation of peer to peer sharing networks. Unlike Nitroshare or any other classic peer to peer software, it only connects to trusted persons and not just anyone. Also, it not just a file sharing utility like Nitroshare, It is completely decentralized communication platform. It sends data over LAN and WAN.

REQUIREMENTS SPECIFICATION:

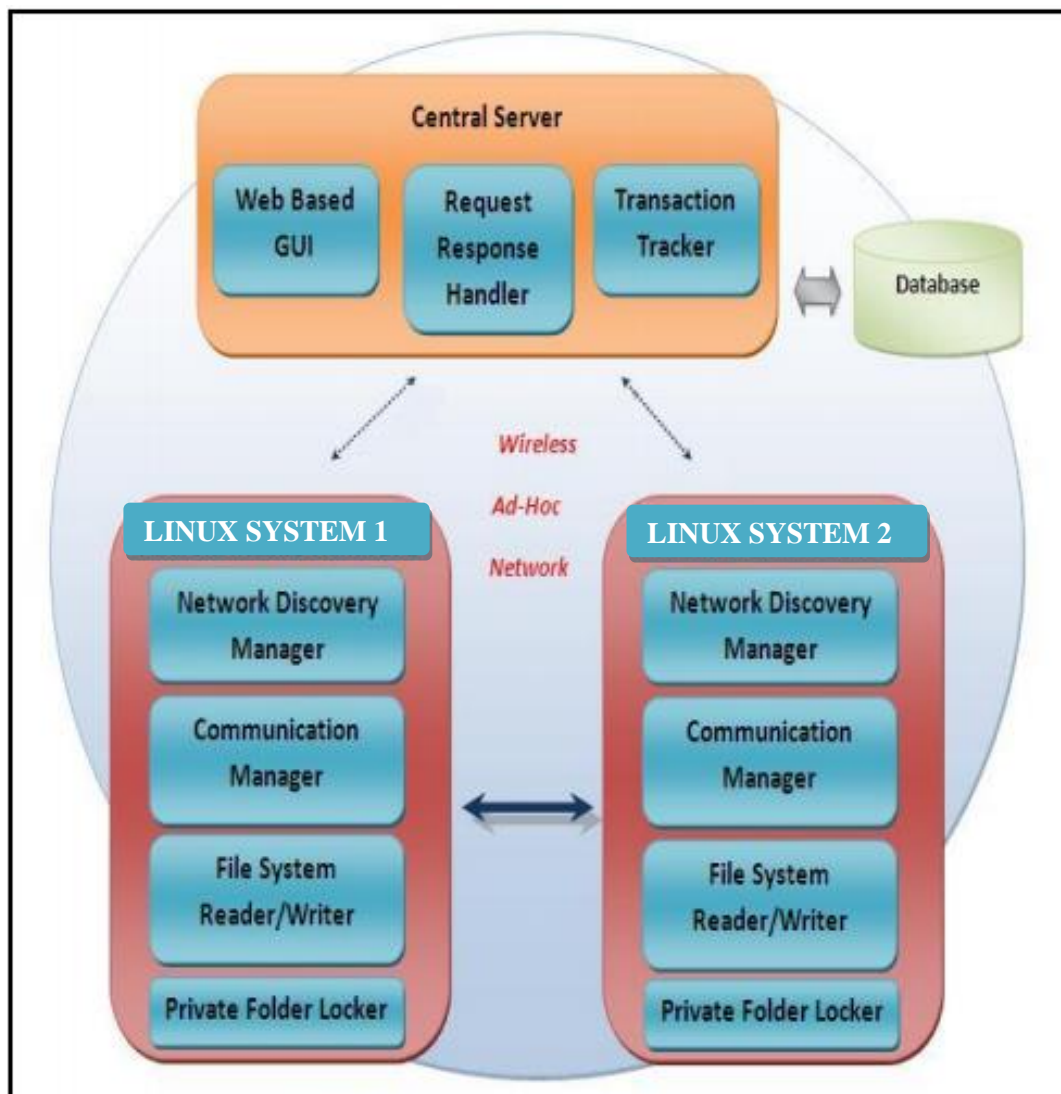
FUNCTIONAL REQUIREMENTS:

Connects two PC's having LINUX environment and shares the file at high speed.

- Be able to asynchronously copy a file from one server to another.
- Be able to recover from partial or failed transfers due to problems in the network or disk systems.
- Report to a user on the progress of a given request.
- Allow a user to signal the relative priorities of requests.
- Handle a multi-file file request as an atomic unit.
- Access files which reside on a set of data servers.
- Other protocols should be usable as the transport protocol.
- Interact with a Mass Storage System when scheduling requests.

NON-FUNCTIONAL REQUIREMENTS:

- The system is more reliable, whenever there is no WIFI direct available, the files are shared through LAN/WAN connections.
- File transfer across network systems are really fast. Size is not limited.
- You can transfer files or folders of any size. It transfers anything.
- You can either transfer a file, folder, image or media. Dynamic file compress during transfer to decrease transfer time and bandwidth.
- CRC checksum generation to ensure file integrity during transfer.
- Full compatibility with clients running on other operating systems.
- Be able to monitor its bandwidth usage, and alter the number of transfers in order to “fill” the available bandwidth.

METHODOLOGY:**SYSTEM ARCHITECTURE****Fig. 1 System Architecture**

1 .Client Side

- A. Network Discovery Manager: It is responsible for handling the connectivity of the system, both detecting Wi-Fi Access Points and determining when peers have become disconnected.
- B. Communication Manager: It deals with sending request to the server for searching contents and receiving list of peers containing required contents.
- C. File System Reader/Writer: It is responsible for reading files and writing files to the system.
- D. Private Folder Locker: It allows user to make content available to other peers.

2. Server Side

- A. Transaction tracker: it keeps a track of transactions carried out by users.
- B. Request Response Handler: It handles request from user and provides response accordingly.
- C. GUI: It provides interaction to administrator.

TECHNOLOGIES:

Bluetooth:

The rate of data transfer between Bluetooth devices is about 3 megabits per second. This is significantly lower than Wi-Fi. The range of communication is limited.

Wi-Fi Technology:

Wi-Fi technology is an alternative to wired technology, which is commonly used for connecting and receives data at high speed. Devices in wireless mode. Wi-Fi (Wireless Fidelity) is a generic term that refers to IEEE 802.11 communication standards for Wireless Local Area Networks. It uses radio technologies to transmit.

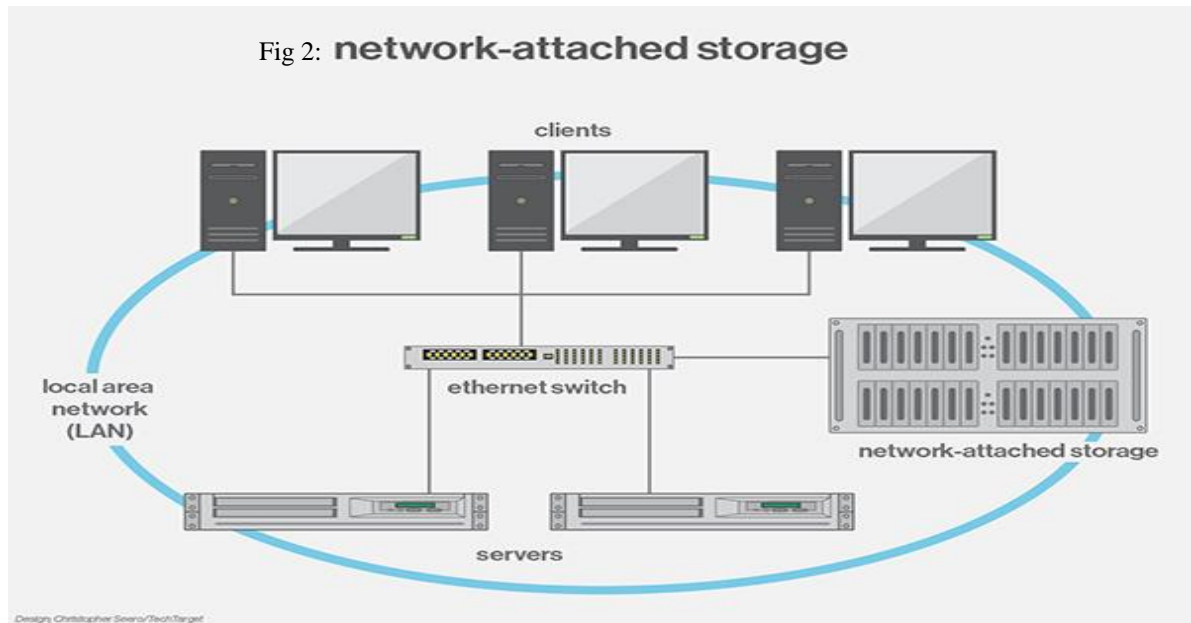
Following are the Advantages of Wi-Fi over Bluetooth:

1. The most prominent advantage which Wi-Fi has over Bluetooth is that Wi-Fi operates at a much faster rate of 11mbps, whereas Bluetooth only operates at a much slower rate of around 720kbps. This makes Bluetooth too slow for video transfers or moving large amounts of large photo images from a digital camera.
2. Wi-Fi is also designed to link up entire networks, rather than computer to computer.

Peer to Peer Technology:

A peer to peer (abbreviated to P2P) computer network is one in which each computer in network can act as a client or server for the other computers in the network, allowing shared access to various resources such as files, peripherals and sensors without the need for a central server. P2P network can be set up within the home, a business, or over the Internet. Each network type requires all computers in the network to use same or a compatible program to connect to each other and access files and other resources found on the other computer.

P2P networks can be used for sharing contents such as audio, video, data, or anything in digital format. With this model, peers are both suppliers and consumers of resources, in contrast to the traditional client-server model where only the server supply (send), and client consume (receive).



SYSTEM IMPLEMENTATION:

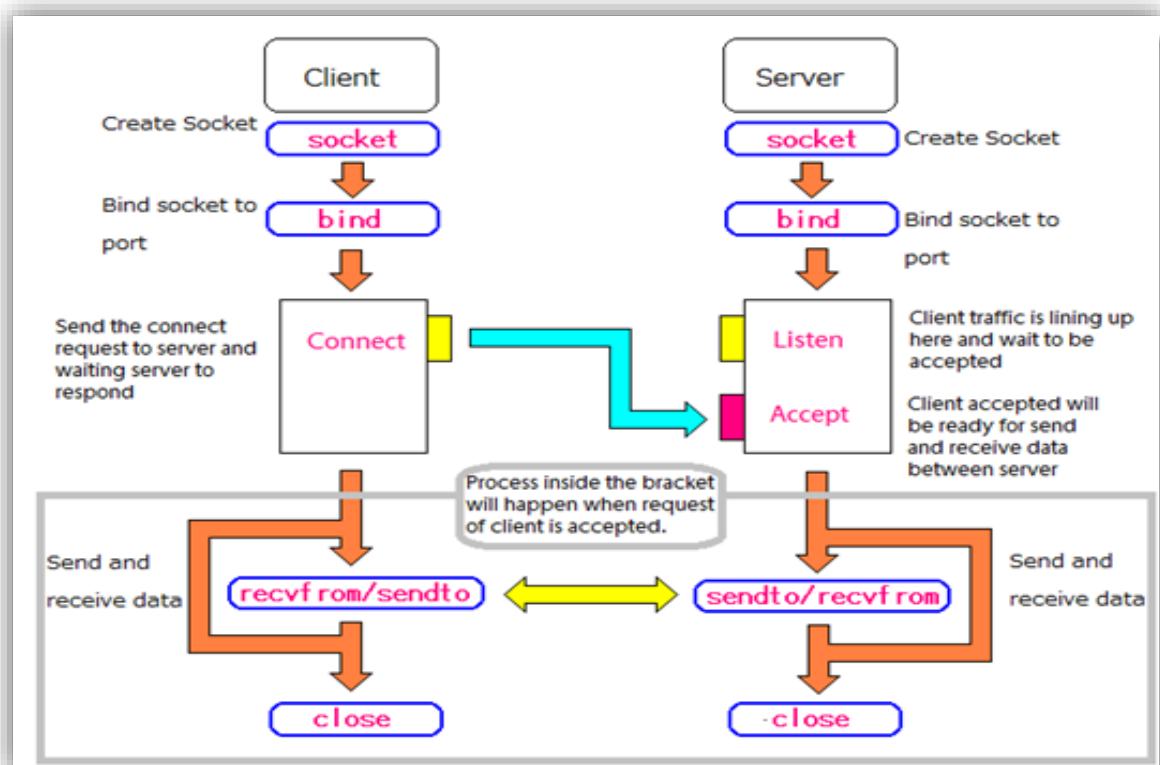


Fig 3: Implementation using TCP/IP sockets.

Server code:

```
#define _XOPEN_SOURCE 700
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <netdb.h> /* getprotobyname */
#include <netinet/in.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <unistd.h>
int main(int argc, char **argv)
{
    char *file_path = "rec.txt";
    char buffer[BUFSIZ];
    char proto_name[] = "tcp";
    int client_sockfd;
    int enable = 1;
    int filefd;
    int i;
    int server_sockfd;
    socklen_t client_len;
    ssize_t read_return;
    struct protoent *protoent;
    struct sockaddr_in client_address, server_address;
    unsigned short server_port = 4567u;
    if (argc > 1)
    {
        file_path = argv[1];
        if (argc > 2)
        {
            server_port = strtoul(argv[2], NULL, 10);
        }
    }
```

```
}

/* Create a socket and listen to it.. */
protoent = getprotobyname(proto_name);
if (protoent == NULL)
{
    perror("getprotobyname");
    exit(EXIT_FAILURE);
}
server_sockfd = socket( AF_INET, SOCK_STREAM, protoent->p_proto );
if (server_sockfd == -1)
{
    perror("socket");
    exit(EXIT_FAILURE);
}
if (setsockopt(server_sockfd, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(enable)) < 0)
{
    perror("setsockopt(SO_REUSEADDR) failed");
    exit(EXIT_FAILURE);
}
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = htonl(INADDR_ANY);
server_address.sin_port = htons(server_port);
if (bind(server_sockfd, (struct sockaddr*)&server_address, sizeof(server_address)) == -1 )
{
    perror("bind");
    exit(EXIT_FAILURE);
}
if (listen(server_sockfd, 5) == -1)
{
    perror("listen");
    exit(EXIT_FAILURE);
}
fprintf(stderr, "listening on port %d\n", server_port);
```



```
while (1)
{
    client_len = sizeof(client_address);
    puts("waiting for client");
    client_sockfd = accept( server_sockfd,(struct sockaddr*)&client_address, &client_len);
    filefd = open(file_path,O_WRONLY | O_CREAT | O_TRUNC,S_IRUSR | S_IWUSR);
    if (filefd == -1)
    {
        perror("open");
        exit(EXIT_FAILURE);
    }
    do
    {
        read_return = read(client_sockfd, buffer, BUFSIZ);
        if (read_return == -1)
        {
            perror("read");
            exit(EXIT_FAILURE);
        }
        if (write(filefd, buffer, read_return) == -1)
        {
            perror("write");
            exit(EXIT_FAILURE);
        }
    } while (read_return > 0);
    close(filefd);
    close(client_sockfd);
    return EXIT_SUCCESS;
}
// return EXIT_SUCCESS;
}
```

Client code:

```
#define _XOPEN_SOURCE 700
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <netdb.h> /* getprotobyname */
#include <netinet/in.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    char protoname[] = "tcp";
    struct protoent *protoent;
    char *file_path = "sample.tmp";
    char *server_hostname = "100.127.5.225";
    char *server_reply = NULL;
    char *user_input = NULL;
    char buffer[BUFSIZ];
    in_addr_t in_addr;
    in_addr_t server_addr;
    int filefd;
    int sockfd;
    ssize_t i;
    ssize_t read_return;
    struct hostent *hostent;
    struct sockaddr_in sockaddr_in;
    unsigned short server_port = 4567;
    if (argc > 1)
    {
        file_path = argv[1];
        if (argc > 2)
```

```
{
    server_hostname = argv[2];
    if (argc > 3)
    {
        server_port = strtol(argv[3], NULL, 10);
    }
}
```

```
file fd = open(file_path, O_RDONLY);
```

```
if (filefd == -1)
```

```
{
    perror("open");
    exit(EXIT_FAILURE);
}
```

```
/* Get socket. */
```

```
protoent = getprotobyname(proto_name);
```

```
if (protoent == NULL)
```

```
{
    perror("getprotobyname");
    exit(EXIT_FAILURE);
}
```

```
sockfd = socket(AF_INET, SOCK_STREAM, protoent->p_proto);
```

```
if (sockfd == -1)
```

```
{
    perror("socket");
    exit(EXIT_FAILURE);
}
```

```
/* Prepare sockaddr_in. */
```

```
hostent = gethostbyname(server_hostname);
```

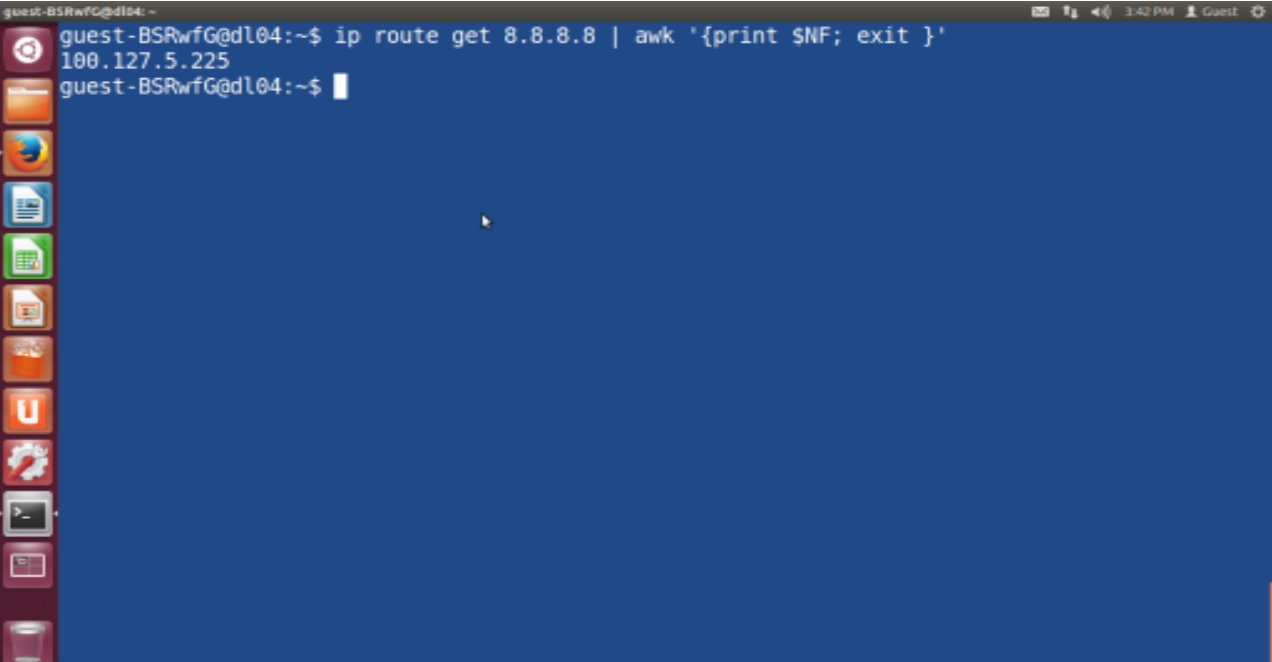
```
if (hostent == NULL)
```

```
{
    fprintf(stderr, "error: gethostbyname(\"%s\")\n", server_hostname);
}
```

```
        exit(EXIT_FAILURE);
    }
    in_addr = inet_addr(inet_ntoa(*(struct in_addr*)(hostent->h_addr_list)));
    if (in_addr == (in_addr_t)-1)
    {
        fprintf(stderr, "error: inet_addr(\"%s\")\n", *(hostent->h_addr_list));
        exit(EXIT_FAILURE);
    }
    sockaddr_in.sin_addr.s_addr = in_addr;
    sockaddr_in.sin_family = AF_INET;
    sockaddr_in.sin_port = htons(server_port);
    /* Do the actual connection. */
    if (connect(sockfd, (struct sockaddr*)&sockaddr_in, sizeof(sockaddr_in)) == -1)
    {
        perror("connect");
        return EXIT_FAILURE;
    }

    while (1)
    {
        read_return = read(filefd, buffer, BUFSIZ);
        if (read_return == 0)
            break;
        if (read_return == -1)
        {
            perror("read");
            exit(EXIT_FAILURE);
        }
        if (write(sockfd, buffer, read_return) == -1)
        {
            perror("write");
            exit(EXIT_FAILURE);
        }
    }
}
```

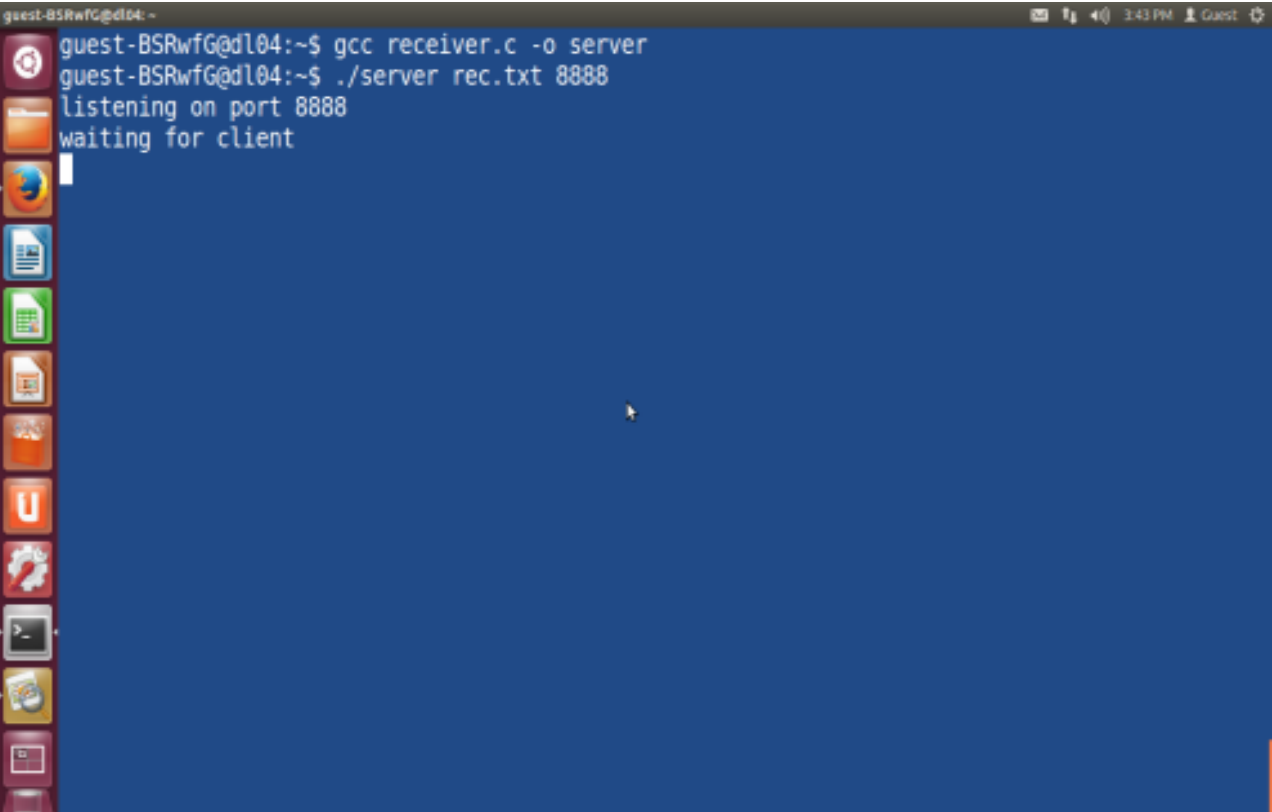
```
    free(user_input);  
    free(server_reply);  
    close(filefd);  
    exit(EXIT_SUCCESS);  
}
```

OUTPUT:

```
guest-BSRwfG@dl04: ~  
guest-BSRwfG@dl04:~$ ip route get 8.8.8.8 | awk '{print $NF; exit }'  
100.127.5.225  
guest-BSRwfG@dl04:~$
```

A terminal window with a blue background and a vertical sidebar of application icons on the left. The terminal shows the command `ip route get 8.8.8.8 | awk '{print $NF; exit }'` being executed, resulting in the output `100.127.5.225`. The prompt `guest-BSRwfG@dl04:~$` is visible at the top and bottom of the terminal area.

Fig 4:To get the IP address of server.



```
guest-BSRwfG@dl04: ~  
guest-BSRwfG@dl04:~$ gcc receiver.c -o server  
guest-BSRwfG@dl04:~$ ./server rec.txt 8888  
listening on port 8888  
waiting for client  
|
```

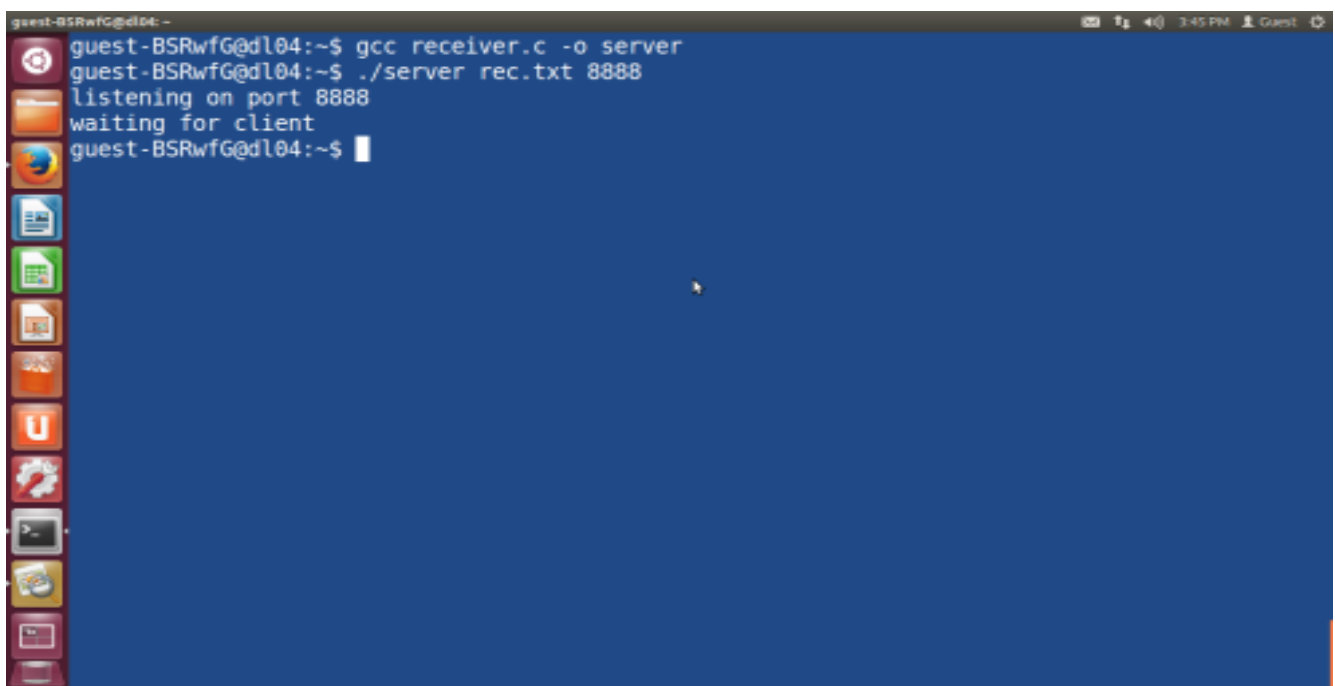
A terminal window with a blue background and a vertical sidebar of application icons on the left. The terminal shows the compilation of `receiver.c` into `server` using `gcc`, followed by the execution of `./server rec.txt 8888`. The output shows the program is `listening on port 8888` and `waiting for client`. A vertical bar `|` is visible on the left side of the terminal area.

Fig 5: Server program executed and it is listening for Client to connect.

A terminal window with a yellow background. The title bar reads 'guest-cMw2Ac@dl05: ~'. The terminal shows three lines of command and output: 'gcc sender.c -o client', './client send.txt 100.127.5.225 8888', and a prompt. On the left, there is a vertical dock with various application icons. The top right corner shows system icons and the time '3:45 PM'.

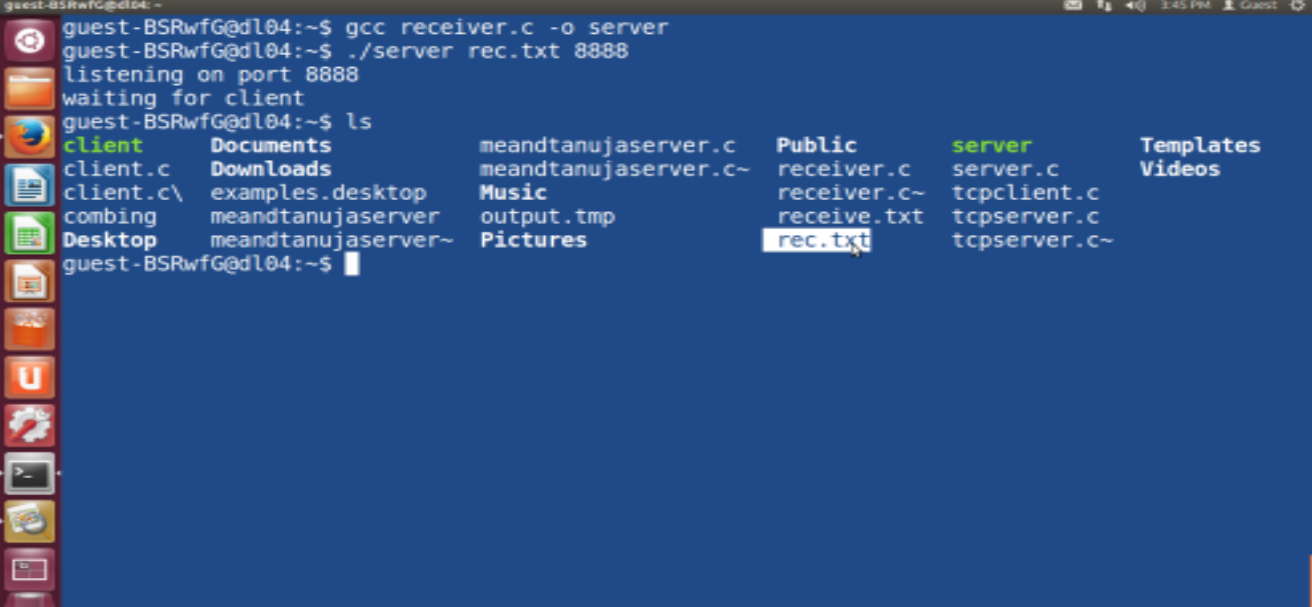
```
guest-cMw2Ac@dl05: ~  
guest-cMw2Ac@dl05:~$ gcc sender.c -o client  
guest-cMw2Ac@dl05:~$ ./client send.txt 100.127.5.225 8888  
guest-cMw2Ac@dl05:~$
```

Fig 6: Client program executed and connected to Server, now it is sending file and communication through port number 8888.

A terminal window with a blue background. The title bar reads 'guest-BSRwfG@dl04: ~'. The terminal shows three lines of command and output: 'gcc receiver.c -o server', './server rec.txt 8888', and 'listening on port 8888 waiting for client'. On the left, there is a vertical dock with various application icons. The top right corner shows system icons and the time '3:45 PM'.

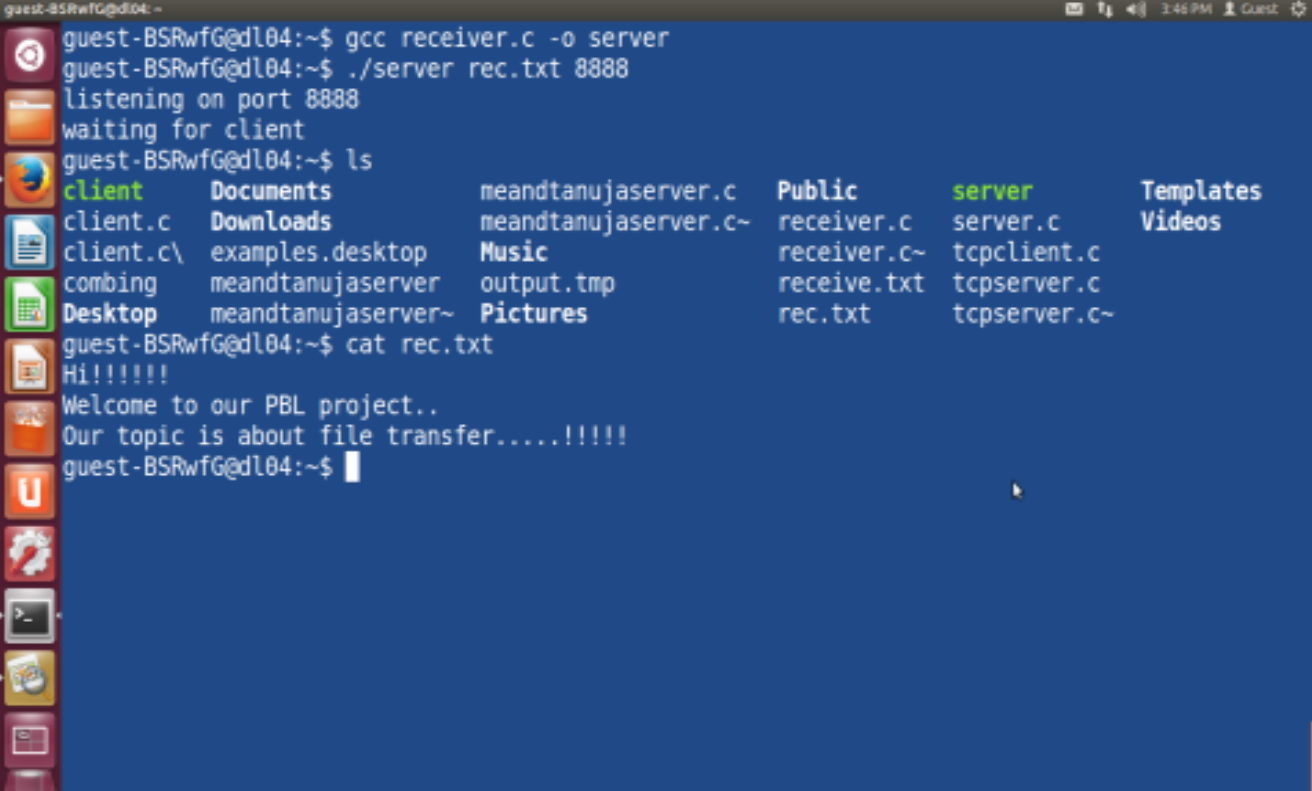
```
guest-BSRwfG@dl04: ~  
guest-BSRwfG@dl04:~$ gcc receiver.c -o server  
guest-BSRwfG@dl04:~$ ./server rec.txt 8888  
listening on port 8888  
waiting for client  
guest-BSRwfG@dl04:~$
```

Fig 7: Server side after receiving the file.

A terminal window titled 'guest-BSRwfG@dl04: ~' with a blue background. The terminal shows the execution of a C program 'server' which listens on port 8888. A file named 'rec.txt' is shown in the 'Public' directory. The left sidebar contains icons for various applications and a file manager showing a directory tree with folders like 'client', 'Desktop', and 'Documents'.


```
guest-BSRwfG@dl04:~$ gcc receiver.c -o server
guest-BSRwfG@dl04:~$ ./server rec.txt 8888
listening on port 8888
waiting for client
guest-BSRwfG@dl04:~$ ls
client      Documents  meandtanujaserver.c  Public      server      Templates
client.c    Downloads  meandtanujaserver.c~ receiver.c  server.c    Videos
client.c\   examples.desktop  Music               receiver.c~ tcpclient.c
combing     meandtanujaserver  output.tmp          receive.txt tcpserver.c
Desktop     meandtanujaserver~ Pictures            rec.txt     tcpserver.c~
guest-BSRwfG@dl04:~$
```

Fig 8: Sent file received at server.

The same terminal window as in Fig 8, but now showing the contents of 'rec.txt' using the 'cat' command. The output is a multi-line message. The file manager sidebar remains visible on the left.

```
guest-BSRwfG@dl04:~$ gcc receiver.c -o server
guest-BSRwfG@dl04:~$ ./server rec.txt 8888
listening on port 8888
waiting for client
guest-BSRwfG@dl04:~$ ls
client      Documents  meandtanujaserver.c  Public      server      Templates
client.c    Downloads  meandtanujaserver.c~ receiver.c  server.c    Videos
client.c\   examples.desktop  Music               receiver.c~ tcpclient.c
combing     meandtanujaserver  output.tmp          receive.txt tcpserver.c
Desktop     meandtanujaserver~ Pictures            rec.txt     tcpserver.c~
guest-BSRwfG@dl04:~$ cat rec.txt
Hi!!!!!!
Welcome to our PBL project..
Our topic is about file transfer.....!!!!!!
guest-BSRwfG@dl04:~$
```

Fig 9: Displaying the contents of received file at Server System.



The screenshot shows a terminal window with a yellow background. The window title is 'guest-cMw2Ac@dl05: ~'. The terminal output is as follows:

```
guest-cMw2Ac@dl05:~$ gcc sender.c -o client
guest-cMw2Ac@dl05:~$ ./client send.txt 100.127.5.225 8888
guest-cMw2Ac@dl05:~$ cat send.txt
Hi!!!!!!
Welcome to our PBL project..
Our topic is about file transfer.....!!!!!!
guest-cMw2Ac@dl05:~$
```

The left sidebar of the terminal window contains several application icons, including a web browser, a file manager, and a terminal icon.

Fig 10: Displaying the contents of sent file at Client System.

EXPECTED OUTCOMES:

CONCLUSION AND FUTURE WORK

We have proposed a peer-to-peer model that permits efficient file sharing between two PC's over a low-cost transport. The employment of P2P protocols and applications in the Linux platform is becoming a promising solution which permits a wide number of users to share their own contents (data, audio, video, etc.) with each other without using costly and centralized network infrastructure. Small segments are desirable when multiple clients are involved, as they allow for increased parallelism of downloads.

The use of socket for content sharing is more optimal, as it only requires two sockets: one for downloading, and one for uploading, for each peer, regardless of the number of peers within the network area. Peer-to-peer sharing allows for efficient content distribution using low-cost links that do not impose a load on the system carrier infrastructure.

It is expected that the proposed protocol will allow system users to communicate among each other in real time by using various P2P applications.

- The application provides appropriate information to users according to the chosen service.
- This application for systems working in Linux environment is a very effective tool which can be used to a great extent. The application is portable and can be easily installed and used on any system having Linux OS.
- Here we are implementing sharing files in same platform .For future scope we can implement for cross environment.

REFERENCES:

Text Books:

[1] Behrouz A. Forouzan, DATA COMMUNICATIONS AND NETWORKING, by McGraw-Hill, a business unit of The McGraw-Hill Companies. Inc.

[2] Tanenbaum, Computer Networks, 5e, by Pearson Education.

Websites:

[3]https://en.wikibooks.org/wiki/Alevel_Computing/AQA/Computer_Components,_The_Stored_Program_Concept_and_the_Internet/Structure_of_the_Internet/Client_server_model.

[4]www.developer.mozilla.org,https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps.