

## **Project Title:**

P2P Bit-Torrent



Session: 2018

## **Submitted by:**

Muhammad Waqas Rasheed  
2018-CS-605

Muhammad Saad Fareed  
2018-CS-632

## **Experience of Members:**

We both have Intermediate Experience in the Socket Programming. We have done a short course from udemy.com also we have done Application Development in Socket Programming by Using Socket.io library. In Future we have plan to make a research work on TCPIP Stack.

## **Submitted to:**

Mam Anam Iftikhar

Department of Computer Science, New  
Campus University of Engineering and  
Technology Lahore, Pakistan

## Objectives:

Project is supposed to implement client to client file transfer application connected to a common server with security and efficiency. Here server provides path of communication among clients.

## Abstract:

The main aim of the project is to develop an application in which Two clients that are connected to a server can transfer files like we can See in Share it application. This project has been developed to carry out the processes easily and quickly by peer to peers file transfer. We have done this Through Socket Programming so that our Server or HUB will monitor all Activities at Run time like what Files are sharing Between Clients.

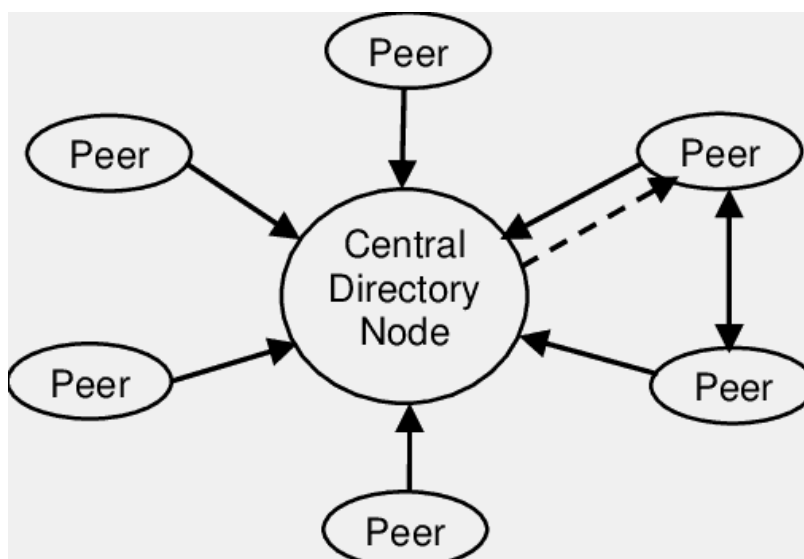
## Features:

In This Project We are trying to make an application in which two clients that are connected to the server can transfers files with each other. Client connect to Nodes that are created By Hub to transfer file or to communicate with clients. Following are some Features of Our Application:

- File of any type can be transfer with any client but Due to use of TCP its speed is low.
- We can make multiple nodes to connect Clients each other.
- It will Download at the speed of 0.1KB per sec that lower than the speed we got from UDP.
- For Efficient Transfer we have use TCP instead of UDP so Data will Transfer Efficiently with No loss of Data.
- Data will Transfer in the Form of Packets instead of once completely.

## Theoretical Aspects:

Socket Programming is Used extensively throughout the project. We have used Peer to peer approach with TCP. In TCP we have Data will Transfer in the Form of Packets instead of once completely. In our project the downloading a file is equal to 100Bytes per second and so on. In peer-to-peer approach our Client will act as both giver and taker both simply it can download file that is available in other peer and also other peers will download files from client side. The approach we used for peer-to-peer is Centralize server system in which we have a central system that connect the networks or peers just like we have in bit-torrent. Centralize system provides us a better security to maintain our activities efficiently and proper monitoring of Activities of peers.



In any distributed system (such as a P2P network) there are usually trade-offs to be done in terms of efficiency, security, scalability, robustness and decentralization. The fundamental difference between the two approaches you mention is that one prioritizes robustness, while the other prioritizes efficiency. A decentralized index approach tends to be more robust (no single point of failure), but it is usually tricky to make it as efficient as a centralized approach. In terms of sociability, decentralized approaches have a bigger potential, but it is not trivial to ensure that a given decentralized system actually scales well from both a theoretical and a practical point of view.

P2P is never "pure" centralized (represented by a traditional client-server architecture) - if you talk about centralized P2P approaches, it means hybrid systems, where several meta information about the data and peers are stored on multiple "centralized" / well-known server entities. These information sets may include parameters like the file availability, IP-addresses, latency value, etc.

The data transmission is still organized in a decentralized process, directly from peer to peer. In a pure decentralized approach, the data exchange of these meta information has to be managed without such well-defined server systems.

## **Methodology:**

BitTorrent is all about sharing data between peers this can be done by P2P file sharing using the suggested methodology. In this we have 2 ways to implement Bit-Torrent one is from UDP and other is TCP. If our main goal is speed efficiency we have to choose UDP instead of TCP and when our main goal is Data Quality and we have to maintain our Data instead of its transfer rate we use TCP. In this project we have use TCP. In peer-to-peer we have 2 types to implement Bit-Torrent 1<sup>st</sup> is Centralize System and 2<sup>nd</sup> is Decentralize system. The Decentralize system is purely peer-to-peer system but in centralized we have Hybrid systems. In centralized system we have a central server system that is connected to different nodes and nodes then connected to Clients or peers. In this we can manage to server to monitoring the whole system that what data is transfer to peer and their connection.

In our Project we have use TCP and Centralize system to implement Bit-Torrent. In this 1<sup>st</sup> we have implement TCP connection by creating Hub as a main centralized system and then this centralized system is connected to nodes that behave as a communicator between node and Hub. For every different client or peer we have a separate folder but having same code of client or peer this is because of no availability of different system that behave as client. In mode connection we have implement TCP. All have done using Threads. We are attempting to create a file sharing application (more than likely been done in other ways) and we are using threading so we can receive files at any time otherwise if our system is busy its will delay our transfer. And we use threads also to transfer data in the form of packets form.

## Code:

### hub.c

```
/* Hub is a server for both Node and Client. */
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netinet/ip.h>
#include<arpa/inet.h>
#include<pthread.h>
#include<errno.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

/* Node to hub -> Registration */
struct msgreg_t {
    int mtype;
    short rvport;
};

/* Client to hub -> Request node list */
struct msggetnode_t {
    int mtype;
};

struct node {
    struct sockaddr_in address;
    long port;
    int del;
};

/* Hub to client -> list of nodes */
struct listmsg_t {
    int mtype;
    struct node nodeList[10];
};

/* Auxiliary message struct */
struct recvmsg_t {
    int mtype;
    long port;
};

int nodes = 0;
struct node nodeList[10];
struct listmsg_t listmsg;
```

```
pthread_mutex_t mtx_msg = PTHREAD_MUTEX_INITIALIZER;
```

```
pthread_mutex_t mtx10 = PTHREAD_MUTEX_INITIALIZER;
```

```
pthread_cond_t cond10 = PTHREAD_COND_INITIALIZER;
```

```
int addNode(struct sockaddr_in nodeAddr, long port)
```

```
{
    nodeList[nodes].address = nodeAddr;
    nodeList[nodes].port = port;
    nodeList[nodes].del = 0;
    listmsg.nodeList[nodes].address = nodeAddr;
    listmsg.nodeList[nodes].port = port;
    listmsg.nodeList[nodes].del = 0;
    nodes++;
    return 0;
}
```

```
int remNode(struct sockaddr_in nodeAddr, int port)
```

```
{
    int i;
    for (i=0;i<nodes;i++)
    {
        if (nodes == 1)
        {
            nodeList[0].del = 1;
            nodes--;
            return 0;
        }

        if (nodeAddr.sin_addr.s_addr == nodeList[i].address.sin_addr.s_addr &&
nodeAddr.sin_port == nodeList[i].address.sin_port)
        {
            int j;
            int k;
            for (j=i;j<nodes;j++)
                nodeList[j] = nodeList[j+1];
            nodeList[nodes].del = 1;
            nodes--;
        }
    }
    return 0;
}
```

```
void* handleThem(void* arg)
```

```
{
    int sock = *((int*) arg);
    int i, len;
```

```

struct recvmsg_t recvmsg;

struct sockaddr_in address; // node or client, whatever
len = sizeof(address);

getpeername(sock, (struct sockaddr*)&address, &len);
// new socket structure

i = recv(sock, &recvmsg, sizeof(struct recvmsg_t), 0);
if (i < 0)
{
    perror("[Hub] recv error:");
    return NULL;
}

if (recvmsg.mtype == 1)
{
    /* Node requested registration */
    pthread_mutex_lock(&mtx10);
    while(nodes >= 10)
    {
        printf("Too many nodes; Waiting...\n");
        pthread_cond_wait(&cond10, &mtx10);
    }
    // Add node (Register)
    addNode(address, recvmsg.port);
    printf("++ %d nodes.\n", nodes);

    printf("Added %s : %ld\n", inet_ntoa(*(struct in_addr *)&nodeList[nodes-
1].address.sin_addr.s_addr), nodeList[nodes-1].port);

    i = 0;
    printf("Currently available:\n");
    while (nodeList[i].address.sin_port != 0)
    {
        printf("[%d] %s : %ld\n", i+1,
inet_ntoa(nodeList[i].address.sin_addr), nodeList[i].port);
        i = i + 1;
    }

    pthread_cond_broadcast(&cond10);
    pthread_mutex_unlock(&mtx10);
}
else if (recvmsg.mtype == 2)
{
    /* Client requested list of nodes */
    int ok;
    listmsg.mtype = 3;
    //listmsg.nodeList = nodeList;

```

```

        ok = send(sock, &listmsg, sizeof(struct listmsg_t ), 0);

        if (ok<0)
        {
            perror("[Hub] Send error:");
        }

        printf("Sent message to the client.\n");
        i = 0;
        printf("Currently available:\n");
        while (nodeList[i].address.sin_port != 0)
        {
            printf("[%d] %s : %ld\n", i+1,
inet_ntoa(listmsg.nodeList[i].address.sin_addr),listmsg.nodeList[i].port);
            i = i + 1;
        }

    }
    else
    {
        /* Unrecognized message */
    }

/* TODO: Node may also disconnect from the hub
    pthread_mutex_lock(&mtx10);
    clients--;
    printf("-- %d clients.\n",clients);
    pthread_cond_broadcast(&cond10);
    pthread_mutex_unlock(&mtx10);
*/

    sleep(1);

close(sock);

return NULL;
}

int main() {
    struct sockaddr_in address, them;
    int sock, sock_them;
    int len;

    pthread_t thr;

    printf("START WITH THE NAME OF ALLAH\n");
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock<0) {

```

```

        perror("[Hub] Error creating:");
    }
    printf("Created sock...\n");

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(1026);

    if (bind(sock, (struct sockaddr*) &address, sizeof(struct sockaddr))<0) {
        perror("[Hub] Error binding:");
    }
    printf("Binded...\n");

    listen(sock, 20);

    while(1) {
        len = sizeof(struct sockaddr_in);
        sock_them = accept(sock, (struct sockaddr*) &them, &len);
        printf("Accepted...\n");
        pthread_create(&thr, 0, handleThem, &sock_them);
        sleep(1+rand()%2);
    }

    return 0;
}

```

.....

## node.c

/\* Node is a client for Hub and a server for Client. \*/

```

#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netinet/ip.h>
#include<arpa/inet.h>
#include<pthread.h>
#include<errno.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/time.h>
#include <sys/stat.h>
#include <fcntl.h>
char fold[100];
/* Message sent to hub */
struct hubmsg_t {
    int mtype;

```



```

    long port;
};

/* Message to/from client */
struct clientmsg_t {
    int mtype;
    char name[4000];
    long from; // pos or length
};

struct sockaddr_in address, client_addr, hub_addr;

void* handleClient(void* arg)
{
    printf("Accepted connection...\n");

    int sock = *((int*) arg);
    int i, len, f;
    char file[256];

    struct clientmsg_t clientmsg;
    struct clientmsg_t sent;

    struct sockaddr_in address; // client
    len = sizeof(address);

    getpeername(sock, (struct sockaddr*)&address, &len);
    // new socket structure

    i = recv(sock, &clientmsg, sizeof(struct clientmsg_t), 0);
    if (i < 0)
    {
        perror("[Node] recv from client error:");
        return NULL;
    }
    else printf("Received message...\n");

    if (clientmsg.mtype == 4)
    {
        /* Client requested stuff */
        strcpy(file, fold);
        strcat(file, clientmsg.name);

        printf("Sending file %s...\n", file);

        /* Open file, bin mode, read-only. */
        f = open(file, O_RDONLY);

        if (f < 0)

```

```

    {
        printf("[Node] Error opening the file '%s'\n", file);
        perror(":");
        return NULL;
    }

    if (lseek(f, clientmsg.from, SEEK_SET) < 0)
    {
        printf("Can't seek in %s.\n",file);
        return NULL;
    }

    sent.mtype = 5;
    sent.from = read(f, sent.name, 100);

    if (sent.from < 0)
    {
        printf("Error reading file %s.\n", file);
        return NULL;
    }

    int ok;
    ok = send(sock, &sent, sizeof(struct clientmsg_t), 0);

    if (ok<0)
    {
        perror("[Node] Send to client error:");
        return NULL;
    }

}
else
{
    /* Unrecognized message (0 = Client finished) */
}

return NULL;
}

```

```

int main() {
    printf("START WITH THE NAME OF ALLAH\n");
    int sock, sockhub, sock_client;
    int i, len;
    printf("Enter Pairing Peer Folder name:");
    scanf("%s",fold);
    strcat(fold, "/");
    //printf("Your Folder is \n");
    //printf(fold);
}

```

```

struct hubmsg_t hubmsg;
struct clientmsg_t clientmsg;
pthread_t thr;

printf("Starting...\n");

sockhub = socket(AF_INET, SOCK_STREAM, 0);
if (sock<0) {
    perror("[Node] Error creating:");
}

printf("Created sock...\n");

/* First of all, the node must connect, i.e. send message to the hub. */

hub_addr.sin_family = AF_INET;
hub_addr.sin_addr.s_addr = INADDR_ANY;
hub_addr.sin_port = htons(1026);

if (connect(sockhub, (struct sockaddr*) &hub_addr, sizeof(struct sockaddr))<0) {
    perror("[Node] Connect error:");
}

struct timeval tv;
struct timezone tz;

gettimeofday(&tv, &tz);

srand(tv.tv_usec);

hubmsg.mtype = 1;
hubmsg.port = 1026 + rand()%1000;

printf("Created message to send...\n");

i = send(sockhub, &hubmsg, sizeof(struct hubmsg_t), 0);
if (i<0) perror("[Node] can't send to hub :");

//printf("Sent.\n");

/* Second of all, the node must be able to accept connections from clients.
I.e. create a new socket, as the socket used for the communication with the hub cannot be
reused.
(as it already acts like a client, and we need one to act like a server) */

sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock<0) {
    perror("[Node] Error creating:");
}

```

```

    }

    printf("Created second socket...\n");

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(hubmsg.port);

    if (bind(sock, (struct sockaddr*) &address, sizeof(struct sockaddr))<0) {
        perror("[Node] Error binding");
    }

    printf("Binded...\n");

    if (listen(sock, 20) < 0)
        perror("[Node] won't listen");

    printf("Listening...\n");

    while(1) {
        len = sizeof(struct sockaddr_in);
        sock_client = accept(sock, (struct sockaddr*) &client_addr, &len);
        pthread_create(&thr, 0, handleClient, &sock_client);
        sleep(1);
    }
    close(sock_client);
    close(sockhub);
    close(sock);
    return 0;
}

```

## client.c

```

/* Client is a client for both Node and Hub. */
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netinet/ip.h>
#include<arpa/inet.h>
#include<errno.h>
#include<stdio.h>
#include<string.h>
#include<sys/time.h>
#include<stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/stat.h>

```

```

#include <fcntl.h>

struct node {
    struct sockaddr_in address;
    long port;
    int del;
};

struct node nodeList[10];

/* Client to hub -> Request List */
struct hubmsg_t {
    int mtype;
    long port;
};

/* Hub to client -> Node List */
struct listmsg_t {
    int mtype;
    struct node nodeList[10];
};
struct listmsg_t listmsg;

/* Message to/from node */
struct nodemsg_t {
    int mtype;
    char name[4000]; // name or data
    long from; // pos or length
};

int num = 0;
char filename[128];
long from;
int get;

pthread_mutex_t mutexFrom = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexGet = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t condGet = PTHREAD_COND_INITIALIZER;

/* Subprogram to download the file */

void *handleFile (void* arg)
{
    int no = *((int *)arg);
    struct nodemsg_t nodemsg;
    struct nodemsg_t recm;
    int socknode, len, i, j, f;

```

```

while(1)
{
    socknode = socket(AF_INET, SOCK_STREAM, 0); // client's
    if (socknode<0) {
        perror("[Client] Error creating:");
    }

    printf("Created sock...\n");

    /* Manipulate! */
    listmsg.nodeList[no].address.sin_port = htons(listmsg.nodeList[no].port);

    if (connect(socknode, (struct sockaddr*) &listmsg.nodeList[no].address, sizeof(struct
sockaddr)) < 0) {
        printf("[Client] Connect error at %d, %s :: %d :\n",no,
inet_ntoa(listmsg.nodeList[no].address.sin_addr), listmsg.nodeList[no].address.sin_port);
        perror(":");
        break;
    }

    printf("Connected to [%d] %s : %ld\n", no,
inet_ntoa(listmsg.nodeList[no].address.sin_addr),listmsg.nodeList[no].port);

    nodemsg.mtype = 4;
    nodemsg.from = -1;
    strcpy(nodemsg.name,filename);

    /* Set the position to read from. */
    pthread_mutex_lock(&mutexFrom);
    if (from >= 0)
    {
        nodemsg.from = from;
        from = from + 100;
    }
    pthread_mutex_unlock(&mutexFrom);

    if (nodemsg.from < 0)
    {
        close(socknode);
        return;
    }

    i = send(socknode, &nodemsg, sizeof(struct nodemsg_t), 0);
    if (i<0) { perror("[Client] can't send to node :"); break; }

    printf("[Client] Sent to node.\n");

    i = recv(socknode, &recm, sizeof(struct nodemsg_t), 0);

```

```

if (i<0)
{
    printf("Error when downloading from %d. >:(\n",i);
    break;
}

printf("[Client] Got from node.\n");

if (recm.from == 0)
{
    /* Downloaded full file. */
    pthread_mutex_lock(&mutexFrom);
    from = -1;
    pthread_mutex_unlock(&mutexFrom);
    close(socknode);
    return;
}

pthread_mutex_lock(&mutexGet);
while (get > 0)
    pthread_cond_wait(&condGet, &mutexGet);
get = 1;
/* Access to the file */
f = open(filename, O_WRONLY);
if (f < 0)
{
    printf("Cannot open file. %d\n", no);
    break;
}

if (lseek(f, nodemsg.from, SEEK_SET) < 0)
{
    printf("Can't seek in file.%d\n",no);
    break;
}

if (write(f, recm.name, recm.from) != recm.from)
{
    printf("Can't write properly in file.%d\n",no);
    break;
}

close(f);

get = 0;
pthread_cond_broadcast(&condGet);
pthread_mutex_unlock(&mutexGet);

close(socknode);

```

```

        sleep(1);
    }

}

int main()
{
    printf("START WITH THE NAME OF ALLAH\n");
    struct sockaddr_in hub_addr;
    int sockhub;
    int len, i;

    struct hubmsg_t hubmsg;
    struct nodemsg_t nodemsg;

    /* First, the client requests the list of nodes from the Hub.*/

    sockhub = socket(AF_INET, SOCK_STREAM, 0);
    if (sockhub<0) {
        perror("[Client] Error creating socket:");
    }

    hub_addr.sin_family = AF_INET;
    hub_addr.sin_addr.s_addr = INADDR_ANY;
    hub_addr.sin_port = htons(1026); // hub port

    if (connect(sockhub, (struct sockaddr*)&hub_addr, sizeof(struct sockaddr))<0) {
        perror("[Client] Connect error:");
    }

    hubmsg.mtype = 2;
    hubmsg.port = -1;

    printf("Created message to send...\n");

    i = send(sockhub, &hubmsg, sizeof(struct hubmsg_t), 0);
    if (i<0) perror("[Node] can't send to hub :");

    printf("Sent.\n");

    i = recv(sockhub, &listmsg, sizeof(struct listmsg_t), 0);
    if (i<0) perror("[Node] can't recv from hub :");

    printf("Got message.\n");

    *((struct node*)nodeList) = *((struct node*)listmsg.nodeList);

    printf("Stored list locally.\n");
}

```



```

i = 0;
printf("List of available nodes:\n");
while (listmsg.nodeList[i].address.sin_port != 0)
{
    printf("[%d] %s : %ld\n", i+1,
inet_ntoa(listmsg.nodeList[i].address.sin_addr),listmsg.nodeList[i].port);
    i = i + 1;
}
num = i;

/* Then, the client must connect to all of the available nodes. */

/* Client enters the name of the file to be downloaded...*/
printf("%d nodes. \nPlease enter the name of the file you want to download:\n", num);
gets(filename);

int f;
/* Create file */
f = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0777);

if (f < 0)
{
    printf("Couldn't create %s\n",filename);
    return 1;
}
close(f);

/* That's right, we need num threads. How awesome is that?*/

struct sockaddr_in address;
int sock;

pthread_t thr[num];
i = 0;
while (i < num && listmsg.nodeList[i].address.sin_port != 0)
{
    if (pthread_create(&thr[i], 0, handleFile, &i) != 0)
    {
        printf("Error creating thread %d.\n",i);
    }
    sleep(1);
    i = i + 1;
}

i = 0;
while (i < num && listmsg.nodeList[i].address.sin_port != 0)
{
    pthread_join(thr[i], NULL);

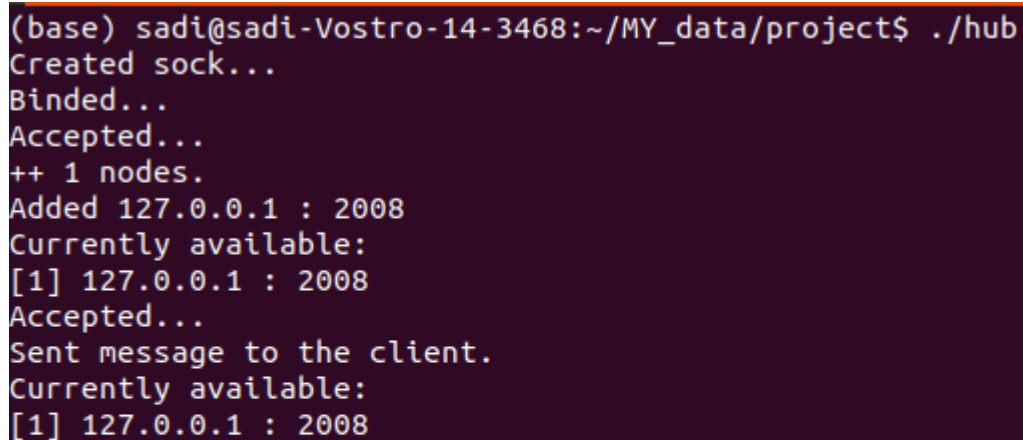
```

```
i=i + 1;  
}  
  
printf("Your File is 100 downloaded.\n");
```

```
char s[1];  
gets(s);  
  
i = 0;  
  
close(sockhub);  
return 0;  
}
```

.....

## Testing:



```
(base) sadi@sadi-Vostro-14-3468:~/MY_data/project$ ./hub  
Created sock...  
Binded...  
Accepted...  
++ 1 nodes.  
Added 127.0.0.1 : 2008  
Currently available:  
[1] 127.0.0.1 : 2008  
Accepted...  
Sent message to the client.  
Currently available:  
[1] 127.0.0.1 : 2008
```

*Figure 1: hub.c*

```
(base) sadi@sadi-Vostro-14-3468:~/MY_data/project$ ./node 192.1686.42.190 8080
START WITH THE NAME OF ALLAH
Enter Pairing Peer Folder name:client1
Starting...
Created sock...
Created message to send...
Created second socket...
Binded...
Listening...
Accepted connection...
Received message...
Sending file client1/saad.txt...
Accepted connection...
Received message...
Sending file client1/saad.txt...
```

Figure 2: node.c

```
(base) sadi@sadi-Vostro-14-3468:~/MY_data/project/client2$ ./client 192.168.42.190 8000
START WITH THE NAME OF ALLAH
Created message to send...
Sent.
Got message.
Stored list locally.
List of available nodes:
[1] 127.0.0.1 : 2008
1 nodes.
Please enter the name of the file you want to download:
saad.txt
Created sock...
Connected to [0] 127.0.0.1 : 2008
[Client] Sent to node.
[Client] Got from node.
Created sock...
Connected to [0] 127.0.0.1 : 2008
[Client] Sent to node.
[Client] Got from node.
Your File is 100 downloaded.

(base) sadi@sadi-Vostro-14-3468:~/MY_data/project/client2$
```

Figure 3: client.c

## **Conclusion:**

P2P Bit-Torrent allows two users to quickly and securely send and receive data, which can lead to gains in efficiency and productivity. With this feature, teams can create a secure collaborative environment where data can easily be shared without fear of external cyber security threats. To monitor file transfer activity hub is used and for connection between peers we have node this all done under a single umbrella of Bit-Torrent.

## **What we have learned from this Project:**

- Team Work and Leadership Qualities.
- TCP and its implementation.
- Peer-to-peer connection.
- Centralized peer-to-peer system.
- Hybrid System.
- Transfer of data in Packets using Threads.
- How to Share other than text file like pdf.

## **Future Work:**

In Future we have to work on seeds as well as. This project is only sharing application yet but we will have to add some more functionalities like uploading in decentralized system with pure peer-to-peer system.

.....

# END OF REPORT