

CMPE 207

Project Report



SpartaTorrent

A BitTorrent like file sharing

Submitted to : Dr. Xiao Su

7th December, 2010

Fall 2010



San José State
UNIVERSITY

Submitted by:

Ashwin Raut-006306079

Aaditya Singhvi- 006481826

Aditya Kumar Verma- 006835062

Neeraj Naik-006925542

Account used to submit the code- Aaditya.Singhvi

Content

1. Abstract.....	2
2. Introduction.....	2
3. Modules.....	5
3.1.storrent generator	5
3.2 Webserver.....	7
3.3 Tracker.....	8
3.3.1 Data Structure	8
3.3.2 Algorithm.....	9
3.4 Manager.....	10
3.4.1 Algorithm.....	10
3.4.2 Seeder.....	10
3.4.2.1 Algorithm	10
3.4.3 Leecher.....	11
3.4.3.1 Algorithm	11
3.5 Updater.....	11
3.5.1.1 Algorithm.....	11
3.6 CRC generator.....	11
4. Logical Flow.....	12
5. Test Scenario with Snapshots.....	17
6. Conclusion.....	19
7. References.....	20

8. Appendix.....	21
------------------	----

1. Abstract

Peer-to-Peer communications contributes to large network traffic and BitTorrent protocol is one of its growing applications. It is estimated that around 40% to 70 % of network traffic is as a result of P2P communication [1] in the year 2008/2009. BitTorrent protocol is a widely used P2P file sharing distribution system and various clients/software have been developed such as μ Torrent, BitTorrent, LimeWire etc. In this paper, we have developed SpartaTorrent- a BitTorrent based file sharing application that is developed on C programming language platform that can be used for sharing large sized files and different file types like text, audio, image, video, etc.

2. Introduction

SpartaTorrent is an application that primarily uses BitTorrent protocol. BitTorrent protocol was developed in the year 2001 by a programmer Bram Cohen [2] using python language, who now owns BitTorrent Inc.

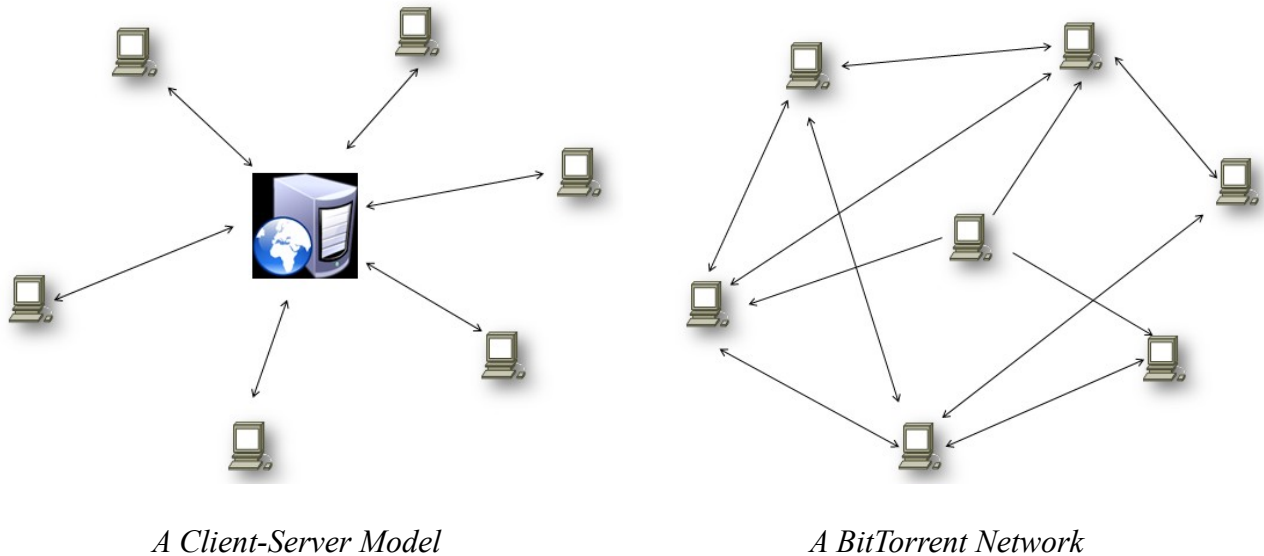


Figure 1: Comparison of Client-Server and BitTorrent Network

As shown in the figure 1, BitTorrent Network differs from the normal Client-Server Model, as a client server model only the server acts as the source, and is centralized. While in case of a BitTorrent Protocol, all the peers can partially act as source of the file, as it is P2P communication. Although the tracker can be

considered a centralized body, there can be multiple trackers, keeping track of seeders and leechers of a torrent file.

BitTorrent Protocol [3] has many components/entities taking part in file sharing. The network in which all the peers/computers take part is called as swarm. The parts of a BitTorrent Protocol are as follows:

- a. Torrent - A file to be downloaded or shared.
- b. Peer- A peer is the computer in the network taking part in the file sharing
- c. Leecher- A peer(s) who wants the file and downloads
- d. Seeder- A peer(s) who has the complete file and shares among the leecher
- e. Tracker- A server that keeps the track of all the peers(seeders and leechers) in the network
- f. .torrent generator- Generates a file with details of tracker like the URL or IP. It generates a file with extension .torrent and also called as metainfo file
- g. Info file- Used to check the integrity of a file using cryptography either SHA1 or Message digest 5
- h. Web Server – It is a web server hosting the .torrent file

The over-all communication in a BitTorrent Protocol is shown in the figure(s) below:

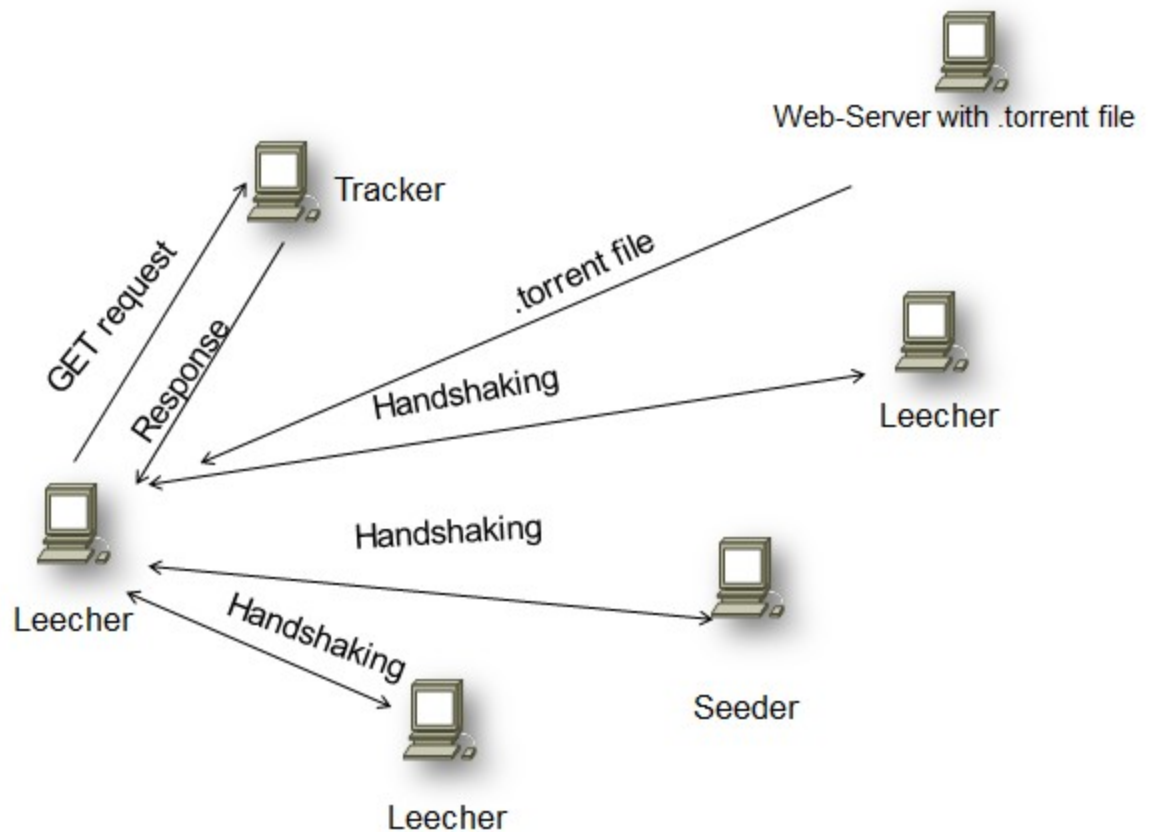


Figure 2(a): BitTorrent Protocol

Figure 2(a) shows how the leecher gets the .torrent file from the web server and then interacts with the tracker to get the seeder and leecher information in the swarm. Once the leecher gets the information it starts handshaking with all the leechers and seeders in its swarm.

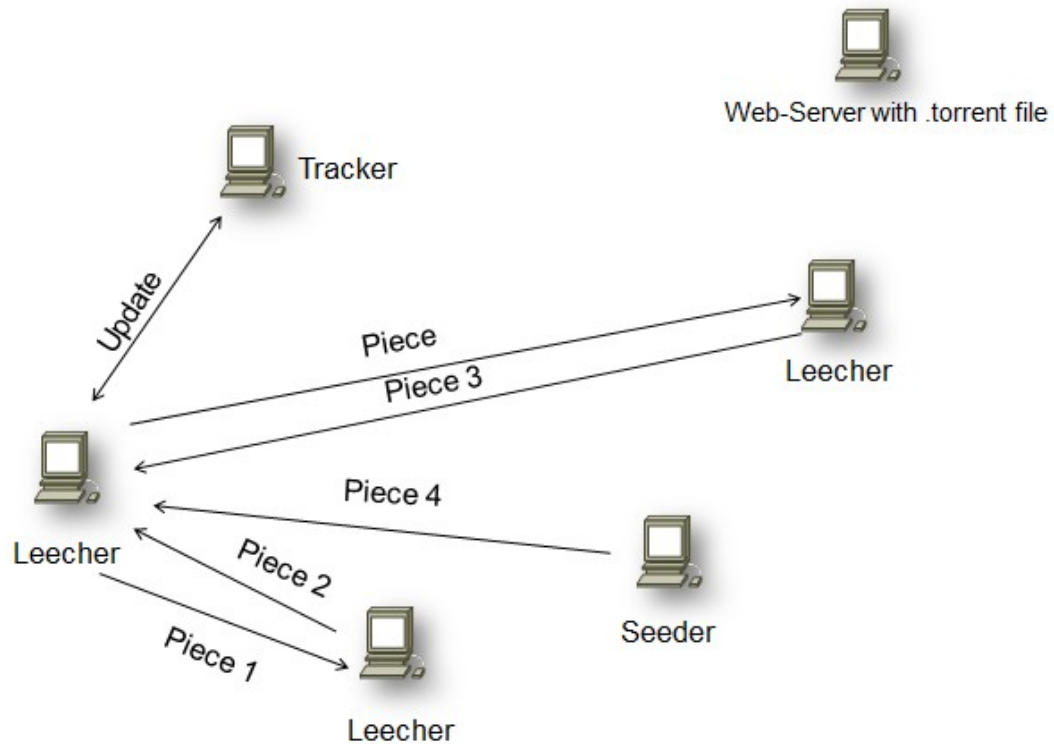


Figure 2(b): BitTorrent Protocol

Figure 2 (a) shows how the signaling takes place, while figure 2(b) shows how file transfer takes place in pieces. The leecher ‘leeches’ different parts from different seeder and leechers. Also, partial seeding starts as shown, where leecher also shares the piece with other leechers. After the leecher gets all the pieces it integrates all the pieces and checks for integrity (using SHA1 /MD5 encryption and decryption). If it has all the pieces, it changes from leecher to seeder. All the entities take almost equal part in file sharing.

In the project SpartaTorrent, we have different modules that are .torrent generator, CRC generator, Tracker, Manager (Leecher, seeder, and Updater). All these entities are comparable to the BitTorrent Protocol, with few changes, like for integrity check using CRC CCITT. Instead of generating .torrent file in SpartaTorrent, a .torrent extension file is generated (SpartaTorrent). Each of the modules is defined with data structure and algorithm in the MODULE section of this paper.

The interaction between these different modules is explained in the LOGICAL FLOW section of this paper.

3. Modules

In SpartaTorrent, we have different modules that are .torrent generator, Web Server, SpartaCRC, Tracker, Manager (Seeder, Leecher, and updater).

3.1 .torrent generator- This module generates a file that has details of the tracker and torrent file TrackerIP, TrackerPort, FileSize, Pieces, PieceSize, LastPieceSize, FileCRC are the fields of the .torrent file

Input- filename

Output- A new file with name filename.torrent

- TrackerIP – The tracker IP address is in dotted decimal format, e.g. 192.168.1.86
- TrackerPort – The port number is any assumed port number between 1024 and 49,151
- FileSize- The file size is calculated by the generator for the file to be shared
- Pieces- The file is broken up into small pieces of equal size 64kB to 1MB and last piece is either less than 64kB to 1MB or equal to 64kB to 1MB

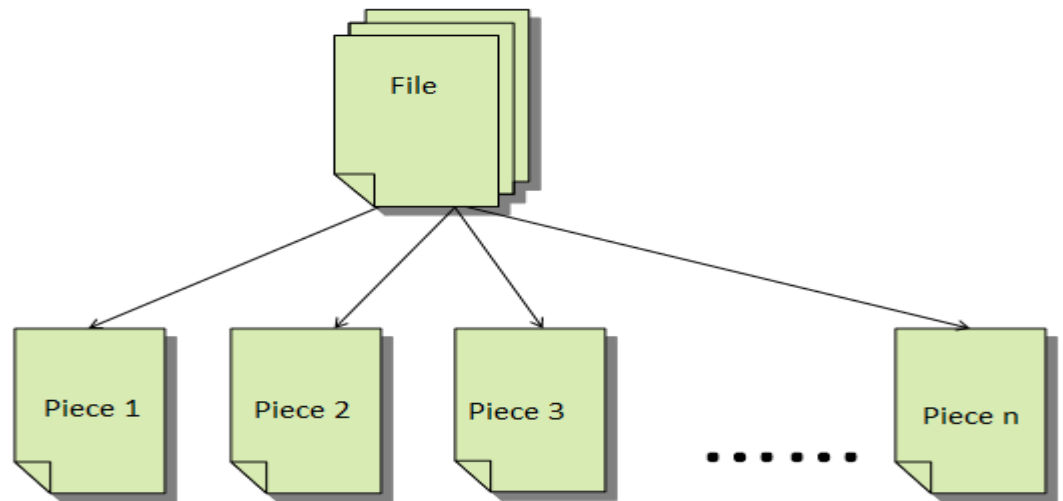
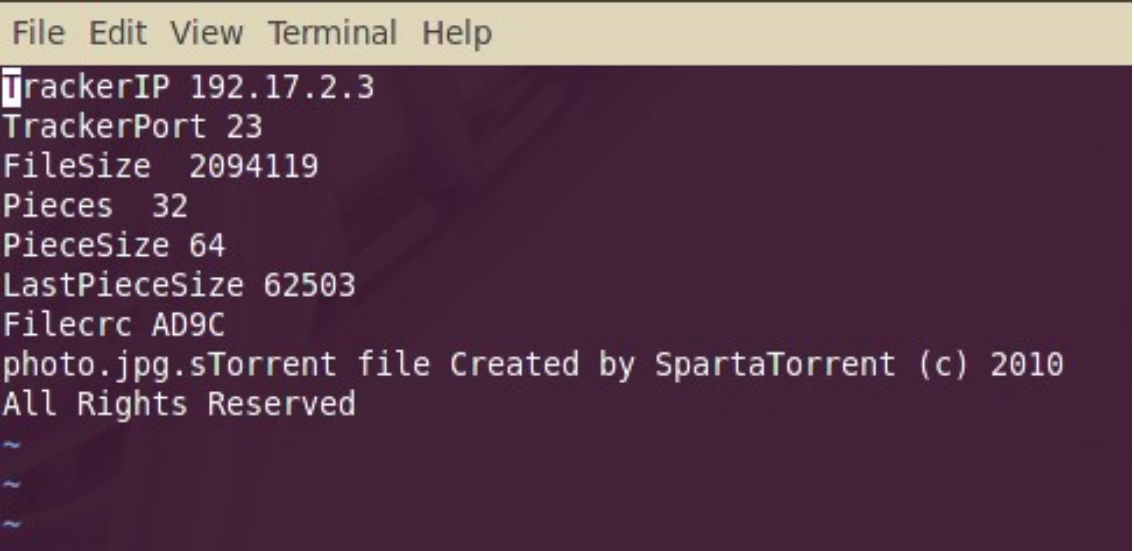


Figure 3: A file broken up into n piece where n-1 pieces of 64kB to 1MB and nth piece less than or equal to 64kBto 1MB

- PieceSize- The file size is fixed of 64kB to 1MB
- LastPieceSize- It is calculated depending on the FileSize and number of pieces of the file.

- FileCRC- It is a unique checksum generated using spartacrc () function which generates a cyclic redundancy checksum and CRC16 CCITT [2] is used which produce a 4digit hex unique number. Even if, a single bit in the file is changed the FileCRC changes. It helps in checking the integrity of the file



```
File Edit View Terminal Help
TrackerIP 192.17.2.3
TrackerPort 23
FileSize 2094119
Pieces 32
PieceSize 64
LastPieceSize 62503
Filecrc AD9C
photo.jpg.storrent file Created by SpartaTorrent (c) 2010
All Rights Reserved
~
~
~
```

Figure 4: Snap-Shot of the photo.jpg.storrent file generated

3.2 Web-Server

The webserver host the .torrent file generated by the .torrent generator. In the project, the leecher connects to the webserver to get the .torrent file that contains all the details. In this project, we have used <http://araut.webs.com/> as the web-server hosting .torrent file in Project section of the webpage.

CMPE 207 - Bit-Torrent like File Sharing Project

This is a CMPE 207 Course (Network Programming) Project. The project comprises of implementing a Bit-torrent like file sharing software in C. The objective of this project is to learn and implement peer-to-peer systems.

Following Components are built:

1. Torrent Tracker.
2. Peer Software (Torrent Seeder and leecher software).
3. Web Server Hosting *.torrent files (These are hosted on this page below).

Figure 1: System Architecture.

Download "*.torrent" Files:

1. file1.torrent
2. prism_2.gif.torrent
3. frog_3.jpg.torrent
4. dog_5.jpg.torrent
5. kiwi.flv.torrent

Figure 5: Snap-Shot of the Webpage hosting .torrent file

In the project the .torrent generator and web server hosting the .torrent file is on the same peer.

3.3 Tracker

This module is heart of the SpartaTorrent. It is the centralized system that plays important role in the file sharing. It is multi-threaded server. It keeps the track of all the seeders and leechers in the swarm. Tracker

is updated at regular interval and it stays alive all the time. Tracker is not directly taking part in file sharing, i.e. it doesn't have the file to be shared but it had the most useful information. It has all information like where is the seeder and leechers for a .torrent file

3.3.1 Data Structure of Tracker

The data structure of the tracker of the SpartaTorrent consists of a linked list of a linked list. The structure of the linked list is shown below in figure 6. The head of the linked list points to the filename of the file1. Each node on the first column of the figure below shows different filenames that the tracker keeps a track. Each node of the first column has three fields that are next part of file1, next node (next filename i.e. filename of file2) and filename itself. The structure for the node is as below

```
struct node
{
    char *value;
    struct node *next;
    struct file_part *node_part;
};
```

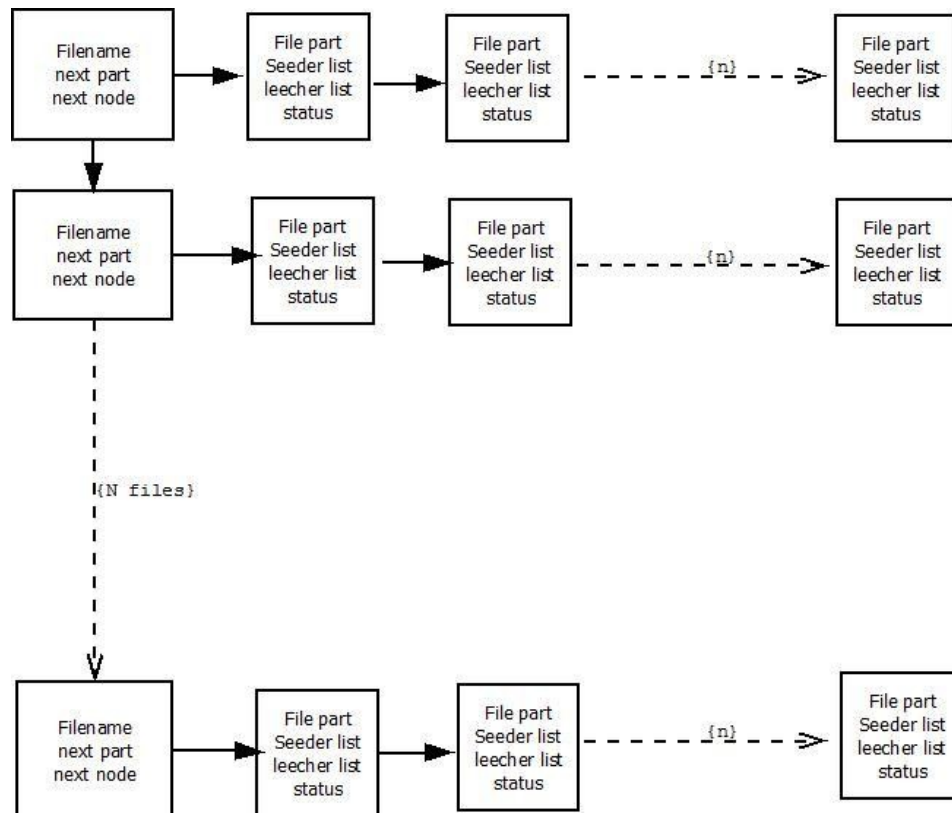


Figure 6: Linked List of a Linked List

The structure for rest of the nodes in the linked list of a linked list consist of file part number, seeder list, leecher list, and next node fields. Horizontally, the linked will be for same file, so there are

n nodes horizontally for a file having n parts. The structure for each node except for the first column in figure 6 is given below:

```
struct seeder{
    int seeder_id;
    struct sockaddr_in sd_sin; // struct sockaddr_in6 sd_sin ( for IPV6)
};

struct leecher{
    int leecher_id;
    struct sockaddr_in lc_sin; // struct sockaddr_in6 lc_sin ( for IPV6)
};

struct file_part{
    struct file_part *next;
    int part_num;
    struct seeder seeder_list[MAX_SEEDS] ; //list of seeders.
    struct leecher leecher_list[MAX_LEECHES] ; //list of leechers.
};
```

3.3.2 Algorithm of the Tracker

1. Start the Tracker (Multi-threaded Server)
2. Create the socket
3. Bind the socket
4. Listen at a pre-defined port
5. Accept the incoming connection from the peer (leecher/seeders) and assign to separate thread
6. Read the incoming message from peer
7. Parse the message and analyze the pre-defined protocol (explained in the logical flow section)
8. Depending on the protocol, send reply to the peer
9. Exit the thread
10. Close socket

3.4 Manager

The Manager module in SpartaTorrent is a combination of both seeder and leecher functions. The manager module plays an important role in file sharing and it decides to be a seeder or a leecher depending on the CRC checksum. If the CRC value is not equal to that in .torrent file, the manager calls the leecher function and if the CRC value matches the CRC field in the .torrent file, it calls the seeder function. As CRC matches means that the peer has all the parts of the file, doesn't need to download and can share among the leechers. Seeder and leecher are not two different programs in the SpartaTorrent project, manager is the program and it calls the sub-module depending on the CRC checksum.

3.4.1 Algorithm of Manager

- 1.** Parse the .torrent file and populate fields
- 2.** Check if file is present on the local directory
- 3.** If yes, call seeder function()
- 4.** If no, call leecher function() and start partial seeding
- 5.** If file download, call spartanCRC() function to check the integrity of the file
- 6.** If CRC matches, exit leecher thread and continue partial seeding
- 7.** If CRC doesn't match, call the leecher() function again.

3.4.2 Seeder

Seeder, as defined in the BitTorrent Protocol, is a peer that has the complete file that is being shared over the network. In the SpartaTorrent project, this module is a sub-module of the manager function. The manager decides to call Seeder () function or to be seeder if the file is present in the local file directory and the CRC checksum matches the CRC as in the .torrent file.

3.4.2.1 Algorithm of the Seeder

- 1.** Read the data from the socket

2. Parse the message to know which file and part number to seed
3. Traverse to the particular part number and open the file
4. Read from the file to a buffer
5. Write buffer to the socket
6. Wait for ACK, if positive ACK goto step 4 else goto step 5
7. Loop until end-of-file

3.4.3 Leecher

Leecher, as defined in the BitTorrent Protocol, is a peer that doesn't have the complete file that is being shared over the network. In the SpartaTorrent project, this module is a sub-module of the manager function. The manager decides to call Leecher() function or to be the leecher if the file doesn't exist on the file directory, or if it does exist but the CRC checksum doesn't match the CRC field in the .torrent file.

3.4.3.1 Algorithm of the Leecher

1. Traverse to the particular part number and open the file
2. Create new piece
3. Read peers IP and port number from the data structure
4. Create a stream socket
5. Connect to the remote peer
6. Send message to the peer indicating which file and part number is required
7. Receive data from the socket
8. Loop until RecvData equals PieceSize
9. Update data structure
10. Close socket

3.5 Updater

The Updater is a module that is called by manager, leecher and seeder to update the data structures to the tracker. It also gets the data structure updated from the tracker.

The Updater uses communication protocol on how to communicate to the tracker that is explained in detail in the logical flow part. In the project, we have used our defined protocol.

3.5.1 Algorithm Updater

1. Get data from a module(seeder, leecher or manager), along with requirement
2. Formulate a buffer according to the tracker communication protocol
3. Open a socket with tracker and send/receive data
4. If get update, then extracts data from the received buffer
5. Segregates seeder/leecher information according to file parts
6. Update the main data structure to reflect the latest leecher and seeder information

3.6 CRC generator

Cyclic Redundancy Checksum is a hash function used in SpartaTorrent instead of MD5 or SHA1 for checking the integrity of a file. It uses CRC-16-CCITT for integrity check of the file. It helps in deciding whether the manager has to call seeder function or leecher function. The SpartaCRC () function is also used to check if the leecher has downloaded all the pieces of a file that is shared.

The CRC generation is included in the .torrent generator file. So, this section is integrated in the .torrent file and also used in leecher function to check the integrity of the file.

4 Logical Flow

The SpartaTorrent is an application for file sharing of a large size between peers. The communication starts with the request from the leecher for a .torrent file of the file that is to be downloaded. This request is made to the web server hosting the .torrent file. In this communication, HTTP protocol is used, leecher/peer sends a GET request and the web server sends that .torrent file to the leecher requesting for

it. In the flowchart 1, all over logical flow is shown. Later in the flowchart 2 and 3, how leecher leeches & partially seed and seeder seeds in shown respectively.

Figure 7: Flowchart 1

Figure 8: Flowchart 2

Figure 9: Flowchart 2

The seeder and leecher function are part of the manager function, depending on condition it check it decides to be a seeder or a leecher. A leecher also partially seeds in SpartaTorrent, if some parts are downloaded and after all parts are downloaded it act as seeder. In figure 8 and figure 9, the flowchart of each explains it, how it changes from leecher to seeder and how seeder listen for incoming connection from any new leecher.

4.1 Communication between Tracker and Manager/Updater

The tracker and manager communicate with each other with some pre-defined protocols in SpartaTorrent project. Figure 10, shows the packet how the message goes to the tracker and how does the tracker interpret the message. There is a Protocol code field which has one of the following values as shown in the figure 10 and meaning of each is explained in detail below.

Figure 10: Protocol defined for SpartaTorrent

Messages TO TRACKER

A) 0x00 Initial Seeder Update

This is message from seeder to tracker that it has a whole file and is available for seeding

0\$filename\$number_of_parts\$peer_id\$seeder_server_port_number

NO REPLY from tracker to seeder.

ACTION on tracker: Create a data structure for file in tracker main data structure.

B) 0x01 Partial seeding update

This is message from leecher to the tracker that it has a specific part number and is available for seeding

1\$filename\$part_num\$peer_id\$seeder_server_port_number

NO REPLY from tracker to leecher.

ACTION on tracker: Updates the fields in file part data structure.

C) 0x02 Gets information on file part request

This is request from leecher, requesting the latest seeder info on specific part

Assumption: 2 seeders are present

2\$seq_no\$filename\$part_num\$

REPLY from tracker to leecher: 2\$seq_no\$filename\$part_num\$peer_id:seeder1IPAddr: port,
peer_id:seeder2IPAddr: port\$

ACTION on tracker: Read fields for file part from data structure and send reply.

D) 0x03 Gets information on whole file request

This is request from leecher, requesting the complete data structure related to the file

Assumption: one seeder is present.

3\$seq_no\$filename\$

If file found in data structure:

REPLY from tracker to leecher: 2\$seq_no\$filename\$part_num\$peer_id:seeder1IPAddr:
port,peer_id\$

(NOTE: This reply will be in for loop i.e. there will be a reply for each file part until end.)

If file not found in data structure:

REPLY from tracker to leecher: FNF (this means file not found or not present in data structure
implications: no seeder present for it)

FNF is a string.

ACTION on tracker: read fields for file part from data structure and send reply: do this all the file
parts.

Figure 11: Sequence diagram showing the protocol communication between updater and tracker

5 Test Scenario

1. Working Test Case (Partial Seeding on Leecher)

In this case, a file is shared between leecher and seeders. Here the number of seeder and leecher can be more than one. Leecher also takes part in partial seeding.

We carried out the test for video file, image file and text file successfully

Figure 12: Appending file part after file download

2. No Seeder

In this case, the leecher comes into the network and no seeder is present to share the file with the leecher. The leecher will keep on trying to look for seeder in the network.

3. No Leecher

In this case, the seeder is in the network and no leecher is present that wants a file. The seeder will in listening state, and waits for any incoming connection

4. File Incomplete

In this case, there is not whole file present. This is possible if few parts of the files are deleted manually, in such case the CRC checksum will fail and network will not have any complete file. Since the CRC will not match that in .torrent file

5. Multiple Peer on same system

In this case, we executed peers on same computer. The seeder and leecher were present on the same computer.

6 Conclusion

BitTorrent is a P2P protocol, used for large distribution of file of different types like video, audio, text etc., efficiently. It has many entities working together like manager, tracker, leecher, seeder, updater, .torrent

generator, CRC check. Each entity has been implemented and tested for functionality. The final integrated code has been tested successfully.

File is downloaded based on different test cases and also by varying file part size. The larger the files the greater part size for file downloads and upload. The implementation increases the download speed as there are many seeder in the swarm and leecher also act as partial seeder.

Thus, over-all the speed of file download and other functionality is better than Client-server model.

7 Reference

- [1] http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009
<http://www.ipoque.com/userfiles/file/ipoque-Internet-Study-08-09.pdf>
- [2] http://en.wikipedia.org/wiki/Bram_Cohen
- [3] J.Fonseca, B. Reza, L. Fjeldsted, DIKU, “BitTorrent Protocol version 1.0,” April 2005
- [4] Beej’s Guide to Network Programming <http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html>
- [5] W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff, “UNIX Network Programming, the Sockets Networking API”, Volume 1, 3rd edition, Addison Wesley, 2004.
- [6] Douglas E. Comer and David L. Stevens, “Internetworking with TCP/IP, Client-Server Programming and Applications”, Volume 3, Linux/POSIX sockets version

8 Appendix

8.1 CRC.H

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

#define iRemainder 0xFFFF
#define XORing      0x0000
#define returnvalue FALSE
#define remaindervalue FALSE
#define crccheck 0x29B1
#define TRUE 1
#define FALSE 0
#define SIZEOFCRC 2
#define SBUFFERSIZE 8
#define poly 0x1021
typedef unsigned short SpartanCRC;
typedef union {
    unsigned char temp[SIZEOFCRC];
    SpartanCRC SpartanCRCvalue;
}int2c;

void SpartanCRCpopulate(void);
SpartanCRC crc16(unsigned char const data[], int lastbyte);
SpartanCRC calculatecrc(char * filename);
```

8.2 Dstruct.h

```
9.
10. #ifndef DSTRUCT
11. #define DSTRUCT
12. #include "header.h"
```

```
13. #include "crc.h"
14. #define MAX_SEEDS 10
15. #define MAX_LEECHES 10
16. #define PARTSIZE 1048576
17. #define NUMBER_OF_THREADS 10
18. #define BUFF 30
19. struct seeder_args{
20.
21.     int socket;
22.     struct file_part *head;
23.
24. }seed_arguments[NUMBER_OF_THREADS];
25.
26. struct seeder{
27.     int seeder_id;
28.     struct sockaddr_in sd_sin;
29. };
30.
31. struct leecher{
32.     int leecher_id;
33.     struct sockaddr_in lc_sin;
34. };
35.
36. // file_part is a linked list.
37. struct file_part{
38.
39.     char filename[MAX_FILENAME_LEN];           //name of the file to which the part belongs.
40.     int part_num;           //part number (can be used for something like hash checking)
41.
42.     FILE *data_ptr;         //ptr to the file created for this part ie whr the data of this part is stored.
43.                             //file descriptor for FILE* data_ptr.
44.     int file_size;          //Number of bytes (part size)
45.
46. //     int offset;           //will hold the offset to the starting point of the part. each part being
    64kb.
47.
48.     struct seeder seeder_list[MAX_SEEDS];      //list of seeders.
49.     struct leecher leecher_list[MAX_LEECHES];  //
50.
51.     float percent_downloaded;
52.     enum status{SEEDING,COMPLETE,LEECHING,PENDING}download_status;
53.     struct file_part *next;
54.
55. };
56.
57. void create_list(struct file_part **headref,char * filename ,int pieces, int piecesize, int lpsize);
58. //enum check_file(struct file_part *,enum ,int );
59. //FILE* join_file(struct file_part *head);//once all parts are downloaded, this fn will join and make a
    complete file.after downloading, this function will first cross
    check 'filename' & 'part_num' to see if file is in order.
60. void print_list(struct file_part *head);//might be needed while debugging.
```

```
61. static float calc_percent(struct file_part* );// HELPER FUNCTION:calculates the data in a file part
62. int random_select(int max_pieces);
63. int leecher_client(struct file_part *head,char *filename,int part_num);
64. void seeder_server(int sock,struct file_part *head);
65. void partial_seeder_server(int sock);
66. // Functions used by seeder
67. void main_seeder(struct file_part *head,int partial);
68. int main_leecher(char *trackerIP,unsigned int trackerport,char * filename, int pieces,int lpsize, int
    fileSize,int peer_id,struct file_part *head,crc filecrc_parsed);
69. void initial_tracker_update(char *trackerIP,unsigned int trackerport,char * filename, int pieces,int
    peer_id);
70. void info_req2track_populate(char *trackerIP,unsigned int trackerport,char * filename, int pieces,int
    peer_id,struct file_part *head);
71. //Functions used by leecher
72.
73.
74. #endif //DSTRUCT
75.
```

8.3 header.h

```
#include "dstruct.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
#include <time.h>
#include <netdb.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>

#include <arpa/inet.h>
#include <netinet/in.h>

#define MAX_FILENAME_LEN 30
#define SEEDERSERVERPORT 7508 // MOVE IT HEADER FILE
#define SOCK_BUFF 1024
#define WAITTIMEFORSEEDER 30 //in seconds

#define SUCCESS 0
#define FAILURE 1
#define CONNECTERROR 1
```

8.4 tracker.h

```
9
10 #ifndef TRACKER
11 #define TRACKER
12
13 #include <sys/types.h>
14 #include <sys/signal.h>
15 #include <sys/socket.h>
16 #include <sys/time.h>
```

```
17 #include <sys/resource.h>
18 #include <sys/wait.h>
19 #include <sys/errno.h>
20 #include <netinet/in.h>
21 #include <netdb.h>
22 #include <pthread.h>
23
24 #include <unistd.h>
25 #include <stdlib.h>
26 #include <stdio.h>
27 #include <string.h>
28
29 #define NO_OF_PARTS 1000
30 #define MAX_SEEDS 5
31 #define MAX_LEECHES 5
32 #define MAX_PEERS 20 // total peers in swarm
33
34 #define TRACKER_PORT 7003
35 #define BUF_SIZE 128
36 #define NUMBER_OF_THREADS 10
37 #define IPV4_SIZE 20 // some extra size included
38
39 //void append_file_part(struct node **,char *, int , int ,int, char *);
40 //void Update(struct node **,char *, int , int ,int);
41 //void append_list ( struct node **, char * );
42
43 /*****Data Structures *****/
44
45 struct node
46 {
47     char value[BUF_SIZE];
48     struct node *next;
49     struct file_part *node_part;
50 }
51 };
52
53 struct seeder{
54     int seeder_id;
55     struct sockaddr_in sd_sin;// struct sockaddr_in6 sd_sin ( for IPV6)
56 };
57
58 struct leecher{
59     int leecher_id;
60     struct sockaddr_in lc_sin;// struct sockaddr_in6 lc_sin ( for IPV6)
61 };
62
63 struct file_part{
64
65     struct file_part *next;
66     int part_num;
67     struct seeder seeder_list[MAX_SEEDS] ; //list of seeders.
68     struct leecher leecher_list[MAX_LEECHES] ; //list of leechers.
```



```
69     //float percent_downloaded;
70     //enum status{SEEDING,COMPLETE,LEECHING,PENDING}download;
71
72 };
73
74 /*****Function Declarations*****/
75
76 void * communicate(void *);
77
78 struct thread_data
79 {
80     int      ssocket;
81     struct sockaddr_in client_sin;
82 };
83
84 struct thread_data thread_data_array[NUMBER_OF_THREADS];
85
86
87 #endif
88
```

8.5 crc.c

```
#include "crc.h"
#define FIXED 8
#define MSB (1<<(WDTH-1))
#define WDTH (FIXED*sizeof(SpartanCRC))

SpartanCRC SpartanCRCTable[256];

#if (returnvalue == TRUE)
#undef returnvalue
#define returnvalue(x) ((unsigned char) reflect((x),FIXED))
#else
#undef returnvalue
#define returnvalue(x) (x)
#endif

#if (remaindervalue == TRUE)
#undef remaindervalue
#define remaindervalue(x) ((SpartanCRC) reflect((x), WDTH))
#else
#undef remaindervalue
#define remaindervalue(x) (x)
#endif

static unsigned long reflect(unsigned long msg, unsigned char lastbit){

    unsigned long reflection = 0x00000000;
    unsigned char valueofbit;
```

```
        for (valueofbit=0; valueofbit<lastbit; ++valueofbit){
            if (msg & 0x01){
                reflection |= (1<<((lastbit-1)-valueofbit));
            }
            else {
                msg=(msg>>1);
            }
        }
        return (reflection);
    }
}
```

```
void SpartanCRCpopulate(void)
{
    SpartanCRC R; // R is the remainder
    int N;         // N/D --> N is dividend and D is Divisor
    unsigned char valueofbit;

    for(N=0;N<256;++N){
        R=N<<(WIDTH-FIXED);
        for (valueofbit=FIXED;valueofbit>0;--valueofbit){
            if (R & MSB)
            {
                R=(R<<1)^poly;
            }
            else
            {
                R=(R<<1);
            }
        }
        //end of first for loop
        SpartanCRCTable[N]=R;
    } //end of second for loop
}
```

```
SpartanCRC crc16(unsigned char const data[], int lastbyte)
{
    SpartanCRC R=iRemainder;
    unsigned char msg;
    int value;
    for(value=0;value<lastbyte;++value){
        msg=returnvalue(data[value])^(R>>(WIDTH-FIXED));
        R=SpartanCRCTable[msg]^(R<<FIXED);
    }
    return (remaindervalue(R)^XORing);
}
```

/*Main function*/

```
SpartanCRC calculatecrc(char *filename)
{
    unsigned char buf[SBUFFERSIZE+3]={0};
    FILE *fp;
    size_t n;
    int2c union_SpartanCRC;
    SpartanCRC temp;
    int i;
    fp=fopen(filename, "r");
    char SpartanCRCstring[SIZEOFCRC];
    n=fread(buf,1,SBUFFERSIZE,fp); /* Reading file 8 bytes*/
}
```

```
SpartanCRCpopulate();
union_SpartanCRC.SpartanCRCvalue=crc16(buf, strlen(buf)); /*Union is defined in SpartanCRC.h*/
for(i=0;i<SIZEOFCRC;i++){
    SpartanCRCstring[i]=union_SpartanCRC.temp[i];
}

SpartanCRCstring[SIZEOFCRC]='\0';

while(!feof(fp)){

    fflush(stdout);
    n=fread(buf,1,SBUFFERSIZE,fp);
    fflush(stdout);
    /*Concatinating */
    buf[n]=SpartanCRCstring[1];
    buf[n+1]=SpartanCRCstring[0];
    buf[n+2]=SpartanCRCstring[2];
    union_SpartanCRC.SpartanCRCvalue=crc16(buf,strlen(buf));
    temp=union_SpartanCRC.SpartanCRCvalue;

    for(i=0;i<SIZEOFCRC;i++){
        SpartanCRCstring[i]=union_SpartanCRC.temp[i];
    }

    SpartanCRCstring[SIZEOFCRC]='\0';

} // end of while
    fflush(stdout);
fclose(fp);
return union_SpartanCRC.SpartanCRCvalue;

}
```

8.6 leecher.h

```
#include "dstruct.h"
#include "header.h"
#include<sys/types.h>
#include<sys/socket.h>
#include<sys/signal.h>
#include<sys/time.h>
#include<sys/resource.h>
#include<sys/wait.h>
#include<sys/errno.h>
#include<netinet/in.h>
#include<netdb.h>
#include<pthread.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>

#define BUFFSIZE 1024

#define DEBUG

static int seq_no=0; // msg seq number
```

```
int random_select(int max_pieces)
{
    /* static long a = 3003003;*/
    /* a = (a*250)%29763027;*/
    /* return ((a%max_pieces));*/
    srand(time(NULL));
    return (rand() % max_pieces);
}

void send_update2tracker(char * trackerIP,int trackerport,char *filename,int part_num,int peer_id)
{
    struct sockaddr_in trackersock_in;
    int sock, ssize;
    char send_buffer[SOCK_BUFF]={0};

    char receive_buffer[SOCK_BUFF]={0};
    ssize = sizeof(trackersock_in);
    memset(&trackersock_in, 0, sizeof(trackersock_in));
    trackersock_in.sin_family = AF_INET;

    inet_pton(AF_INET,trackerIP, &(trackersock_in.sin_addr));
    trackersock_in.sin_port=htons(trackerport);

    memset(&send_buffer,0,SOCK_BUFF);
    sprintf(send_buffer,"1$$$%d$%d$%d",filename,part_num,peer_id,SEEDERSERVERPORT); //
0x02 protocol field
#ifdef DEBUG
    printf("\nThe 0x01 Buffer is %s ",send_buffer);
    fflush(stdout);
#endif

    /*create socket*/
    if((sock=socket(PF_INET,SOCK_STREAM,0))<0){
        printf("\nFailed to create socket ... \n");
        exit(1);
    }

#ifdef DEBUG
    printf("\nConnecting to Tracker at IP: %s , Port %d ",trackerIP,trackerport);
    fflush(stdout);
#endif

    /*Connect to server*/
    if(connect(sock,(struct sockaddr *)&trackersock_in,sizeof(trackersock_in)) <0){
        printf("\nFailed to connect to tracker ... \n");
        exit(1);
    }
    /*Send protocol to Tracker*/

    send(sock,send_buffer,strlen(send_buffer),0);
#ifdef DEBUG
    printf("\n0x01 protocol Buffer Sent ...");
    fflush(stdout);
#endif
    close(sock);
}
```

```
}

void update_listds(struct file_part *head, char *receive_buffer)
{
    unsigned int part_num=0;
    struct file_part *temp=head;
    int seederindex=0;
    char * field,*fieldvalue,*subfield,*subfieldvalue;
    char seederip[30];
    int seeder_port;
    int i=0,node_number=0;
    int noseederflag=0;
    int peer_id;
    char filename[BUFF]={0};
    char tempbuffer[BUFFSIZE]={0};
    char tempbuffer2[BUFFSIZE]={0};

    fieldvalue=(char *)malloc(sizeof(char) * 300); // assuming 10 seeders
    subfield=(char *)malloc(sizeof(char) * 300); // assuming 10 seeders
    //incoming msg 4$1$frog_3.jpg$1$877:130.65.157.190:7508,

    field=strtok(receive_buffer,"$");// protocol

    fieldvalue=strtok(NULL,"$"); //seq_no
    fieldvalue=strtok(NULL,"$"); //filename
    strcpy(filename,fieldvalue);
    fieldvalue=strtok(NULL,"$"); //partnumber this msg is for..
    sscanf (fieldvalue, "%d", &part_num);
    printf("\nPart Number:%d",part_num);
    fieldvalue=strtok(NULL,"$"); //peer_id:IPaddr:port,peer_id:IPaddr:port ..... based on existing
no. of seeder for this part..
    printf("\nParsed Response of all seeders and put in field value:%s",fieldvalue);
    strcpy(tempbuffer,fieldvalue); // example : 877:130.65.157.190:7508,
    fflush(stdout);
    i=0;
    printf("\nEntering While..temp buffer is :%s",tempbuffer);
    fflush(stdout);
    fieldvalue=strtok(tempbuffer,":"); // put 1st seeder in field value

    while(fieldvalue!=NULL){

        printf("\nInside While.x.");
        fflush(stdout);
        //strcpy(subfield,fieldvalue);
        //printf("\nModified Fieldvalue:%s",fieldvalue);
        //printf("\nParsed fieldvaule response of one seeder info:%s",subfield); // example :
877:130.65.157.190:7508
        //fflush(stdout);
        //strcpy(tempbuffer2,subfield);

        printf("\nParsed value of %dth peer id:%s",i,fieldvalue);
        fflush(stdout);
        sscanf (fieldvalue, "%d", &peer_id); // converting in int

        fieldvalue=strtok(NULL,":"); // seeder ip 130.65.157.190
        //printf("\nModified subfield:%s",subfield);

        printf("\nParsed value of %dth seeder ip:%s",i,fieldvalue);
    }
}
```

```

fflush(stdout);

strcpy(seederip,fieldvalue);
fieldvalue= strtok(NULL, "."); // seeder port
//printf("\nModified subfield:%s",subfield);
printf("\nParsed value of %dthe seeder port::%s",i,fieldvalue);
fflush(stdout);
sscanf (fieldvalue, "%d", &seeder_port);

//subfieldvalue= strtok(NULL, "."); // seeder port

/*
printf("\nParsed Data:");
printf("\nPeer_id.%d",peer_id);
fflush(stdout);
*/
/*
*/
/*
*/
printf("\nSeeder Ip: %s",seederip);
fflush(stdout);
printf("\nSeeder Port:%d",seeder_port);
fflush(stdout);
*/
if(head!=NULL){
printf("\npreparing to populate..");
fflush(stdout);

while(temp->part_num!=part_num){
temp=temp->next;
}

strcpy(temp->filename,filename);
temp->part_num=part_num;

seederindex=0;
while((seederindex<MAX_SEEDS) && (temp-
>seeder_list[seederindex].seeder_id!=0)){

printf("\nSeeder Index is:%d and seeder id: %d peer id is:
%d",seederindex,temp->seeder_list[seederindex].seeder_id,peer_id);
fflush(stdout);
if(temp->seeder_list[seederindex].seeder_id==peer_id)
{
printf("\npeer id Match");
fflush(stdout);
break;
}
else{
printf("\nIncrementing seeder index");
fflush(stdout);
seederindex++;
}
}
if(seederindex<MAX_SEEDS){
temp->seeder_list[seederindex].seeder_id=peer_id;
inet_pton(AF_INET,seederip, &(temp-
>seeder_list[seederindex].sd_sin.sin_addr));
temp-
>seeder_list[seederindex].sd_sin.sin_port=htons(seeder_port);
}
}

```

```

                                else{
                                printf("\nNo Space Available to add extra seeder ... didnt
populate the node\n");
                                }
                                }

                                fieldvalue=strtok(NULL,":"); //next peer_id
                                if(fieldvalue==NULL)
                                {
                                break;
                                }
                                }

                                printf("\noutside While..");
                                fflush(stdout);
                                }

// function used to get latest information from tracker about the part.
void getinfo_part_populate(struct file_part * head,char *trackerIP,unsigned int trackerport,char * filename,int
part_num)
{
    struct sockaddr_in trackersock_in;
    int sock, sinsize;
    char send_buffer[SOCK_BUFF]={0};

    char receive_buffer[SOCK_BUFF]={0};
    sinsize = sizeof(trackersock_in);
    memset(&trackersock_in, 0, sizeof(trackersock_in));
    trackersock_in.sin_family = AF_INET;

    inet_pton(AF_INET,trackerIP, &(trackersock_in.sin_addr));
    trackersock_in.sin_port=htons(trackerport);

    memset(&send_buffer,0,SOCK_BUFF);
    sprintf(send_buffer,"2$%d$s$d$",seq_no,filename,part_num); // 0x02 protocol field
    seq_no++;
    if(seq_no>=65535)
        seq_no=0;
    #ifdef DEBUG
    printf("\nThe Buffer is %s ",send_buffer);
    fflush(stdout);
    #endif

    /*create socket*/
    if((sock=socket(PF_INET,SOCK_STREAM,0))<0){
        printf("\nFailed to create socket ... \n");
        exit(1);
    }

    #ifdef DEBUG
    printf("\nConnecting to Tracker at IP: %s , Port %d ",trackerIP,trackerport);
    fflush(stdout);
    #endif

    /*Connect to server*/
    if(connect(sock,(struct sockaddr *)&trackersock_in,sizeof(trackersock_in)) <0){
        printf("\nFailed to connect to tracker ... \n");
        exit(1);
    }
}
```

```
        /*Send protocol to Tracker*/

        send(sock,send_buffer,strlen(send_buffer),0);
#ifdef DEBUG
        printf("\nBuffer Sent ... Waiting for response...");
        fflush(stdout);
#endif
recv(sock,receive_buffer,sizeof(receive_buffer),0);

#ifdef DEBUG
        printf("\nReceived Buffer for piece..%d.%s",part_num,receive_buffer);
        fflush(stdout);
#endif

        //IF 0xFF is received there is no seeder in the list*****
        if(!strcmp(receive_buffer,"FF")){
            printf("\n Received FF Some error");
        }

        //send buffer to update node
        update_listds(head,receive_buffer);

        printf("\n node populated");
        fflush(stdout);
        close(sock);
    }

void info_req2track_populate(char *trackerIP,unsigned int trackerport,char * filename, int pieces,int
peer_id,struct file_part *head)
{
    struct sockaddr_in trackersock_in;
    int sock, sinsize;
    sinsize = sizeof(trackersock_in);
    char send_buffer[SOCK_BUFF]={0};
    char receive_buffer[SOCK_BUFF]={0};
    memset(&trackersock_in, 0, sizeof(trackersock_in));
    trackersock_in.sin_family = AF_INET;

    char * field,*fieldvalue,*subfield,*subfieldvalue;
    char seederip[30],ack='R';
    int seeder_port;
    int i=0,node_number=0;
    int noseederflag=0;
    inet_pton(AF_INET,trackerIP, &(trackersock_in.sin_addr));
    trackersock_in.sin_port=htons(trackerport);
#ifdef DEBUG
    printf("\nSending Retrieve file Info request to tracker ",trackerIP,trackerport);
    fflush(stdout);
#endif

    /*create socket*/
    if((sock=socket(PF_INET,SOCK_STREAM,0))<0){
        printf("\nFailed to create socket ... \n");
        exit(1);
    }
}
```



```
    }

#ifdef DEBUG
    printf("\nConnecting to Tracker at IP: %s , Port %d ",trackerIP,trackerport);
    fflush(stdout);
#endif

    /*Connect to server*/
    if(connect(sock,(struct sockaddr *)&trackersock_in,sizeof(trackersock_in)) <0){
        printf("\nFailed to connect to tracker ... \n");
        exit(1);
    }
    /*Send protocol to Tracker*/
    do{
        memset(&send_buffer,0,SOCK_BUFF);
        sprintf(send_buffer,"3%d%s$d",seq_no,filename,pieces); // 0x03 protocol field
        seq_no++;
        if(seq_no>=65535)
            seq_no=0;

#ifdef DEBUG
        printf("\nThe Buffer is %s ",send_buffer);
        fflush(stdout);
#endif

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

#ifdef DEBUG
        printf("\nBuffer Sent ... Waiting for response...");
        fflush(stdout);
#endif

        for(node_number=1;node_number<=pieces;node_number++){

            recv(sock,receive_buffer,sizeof(receive_buffer),0);

#ifdef DEBUG
            printf("\nReceived Buffer for piece %d...%s",node_number,receive_buffer);
            fflush(stdout);
#endif

            //IF 0xFNF is received there is no seeder in the list*****
            if(!strcmp(receive_buffer,"FNF")){
                noseederflag=1;
                break;
            }
            noseederflag=0; //seeder is present
            //send buffer to update node
            update_listds(head,receive_buffer);
            printf("\nThe ack Buffer is %c ",ack);
            fflush(stdout);
            send(sock,&ack,sizeof(ack),0);
        }
        print_list(head);
        printf("\n List populated");
        fflush(stdout);
        close(sock);

        if(noseederflag==1){
            sleep(WAITTIMEFORSEEDER);
        }
    }
```

```
    }while(noseederflag==1);
}

int joinfile(char * filename,long int fileSize,int noOfParts,int lpsize,crc filecrc_parsed)
{
    FILE *tempfp;
    FILE *fp;
    char newfilename[20];
    char appendedfile[20];
    char * buffer;
    size_t n;
    int i,Partno;
    crc crc_new;
    sprintf(appendedfile,"%s%s","New",filename);
    fp=fopen(appendedfile,"a+");
    buffer = (char*) malloc (sizeof(char)*PARTSIZE);
    if (buffer == NULL) {fputs ("Memory error",stderr); exit (2);}
    printf("\n****Appending parts and forming: %s****",appendedfile);

#ifdef DEBUG
    printf("\nreceived file size %ldand parts %d and lpsize%d****",fileSize,noOfParts,lpsize);
    fflush(stdout);
#endif

    if(lpsize!=0){
        for(i=1;i<noOfParts;i++){
            memset(buffer,0,sizeof(buffer));
            sprintf(newfilename,"%s.part.%d",filename,i);
            printf("\nAppending %s to file %s",newfilename,appendedfile);
            fflush(stdout);
            tempfp=fopen(newfilename,"r+");
            n=fread(buffer,1,PARTSIZE,tempfp);
            fwrite(buffer,1,n,fp);
            fclose(tempfp);
        }
        memset(buffer,0,sizeof(buffer));
        sprintf(newfilename,"%s.part.%d",filename,noOfParts);
        printf("\nAppending %s to file %s\n",newfilename,appendedfile);
        tempfp=fopen(newfilename,"r+");
        n=fread(buffer,1,lpsize,tempfp);
        fwrite(buffer,1,lpsize,fp);
        fclose(tempfp);
        fclose(fp);
    }
    else{
        sprintf(newfilename,"%s.part.%d",filename,noOfParts);
        printf("\nAppending %s to file %s\n",newfilename,appendedfile);
        tempfp=fopen(newfilename,"r+");
        n=fread(buffer,1,fileSize,tempfp);
        fwrite(buffer,1,fileSize,fp);
        fclose(tempfp);
        fclose(fp);
    }
}
```

```
    }
    free(buffer);

    printf("\nFile created, checking crc ...%s",filename);
    crc_new=calculatecrc(filename);
    if(crc_new==filecrc_parsed)
    {
        printf("\nComplete File is present\n");
        fflush(stdout);
        return SUCCESS;
    }
    else{
        printf("\nFILE CRC mismatch\n");
        return FAILURE;
    }
}

void delete_parts(char * filename,int noOfParts)
{
    int i;
    char newfilename[20];
    printf("no of parts = %d\n",noOfParts);
    for(i=1;i<=noOfParts;i++){
        sprintf(newfilename,"%s.part.%d",filename,i);
#ifdef DEBUG
        printf("Deleting %s...\n",newfilename);
        fflush(stdout);
#endif
        if(remove(newfilename) != 0)
            printf("Unable to delete %s\n",newfilename);
        else{
            //printf("Deleted %s\n",newfilename);
        }
    }
}

int leecher_client(struct file_part *head,char *filename,int part_num)
{
    struct file_part *temp = head;
    char newpart[BUFF]={0};
    char buffer[BUFFSIZE],tempbuff[BUFFSIZE];
    char msg[50];
    char p_ack = 'P', n_ack = 'N',s_ack = 'S',fpa_buff = 'S',fp_buff;
    unsigned long int ip_addr;
    unsigned short int port;
    int sock;
    int no_of_pieces=0,random_piece=0,part_size,no_of_seeders=0;
    int size_counter=0,write_count=0,i;
    size_t bytes_read;
    struct leecher sin;
    pthread_mutex_t safe_leech=PTHREAD_MUTEX_INITIALIZER;
    //calculating number of pieces
#ifdef DEBUG
```

```
printf("\n Filename received in head node :%s and filename %s, part num is%d", head-
>filename,filename, part_num);
fflush(stdout);
#endif
while(temp){
    no_of_pieces++;
    temp=temp->next;
}
#ifdef DEBUG
printf("\nNumber of pieces in file: %d",no_of_pieces);
fflush(stdout);
#endif
temp = head;
while(temp->part_num != part_num){
    temp=temp->next;
}
#ifdef DEBUG
printf("\nI have to download part: %d",temp->part_num);
fflush(stdout);
#endif
//Now temp is at the reqd node.
if(temp->part_num == no_of_pieces)
    part_size = temp->file_size - (no_of_pieces* PARTSIZE);
else
    part_size = PARTSIZE;

sprintf(newpart,"%s.part.%d",filename,part_num);
#ifdef DEBUG
printf("\nNew part name is: %s",newpart);
fflush(stdout);
#endif
pthread_mutex_unlock(&safe_leech);
pthread_mutex_lock(&safe_leech);

temp->data_ptr = fopen(newpart,"ab+");

if(!temp->data_ptr){
    printf("ERROR: file '%s' cannot be created.No space\n",newpart);
    exit(1);
}
pthread_mutex_unlock(&safe_leech);
for(i=0;i<MAX_SEEDS;i++){
    if(temp->seeder_list[i].seeder_id)
        no_of_seeders++;
}

#ifdef DEBUG
printf("\nNo of seeders in the seeder list are: %d\n",no_of_seeders);
fflush(stdout);
#endif

random_piece = random_select(no_of_seeders);

#ifdef DEBUG
printf("\nRandom piece to be selected is: %d\n",random_piece);
fflush(stdout);
#endif
#endif
```

```
memset(&sin.lc_sin,0,sizeof(sin.lc_sin));
sin.lc_sin.sin_family = AF_INET;
//get seeders address.
pthread_mutex_lock(&safe_leech);

/*
ip_addr = ntohl(temp->seeder_list[random_piece].sd_sin.sin_addr.s_addr);
if((sin.lc_sin.sin_addr.s_addr = htonl(ip_addr)) == INADDR_NONE){//update ip addr
    printf("Cant get IP address of seeder\n");
}
port = htons(temp->seeder_list[random_piece].sd_sin.sin_port);
if((sin.lc_sin.sin_port = htons(port)) == 0){//update port number.
    printf("Can't get Port nnumber of seeder\n");
}
*/
/***** Need to refine this logic *****/
sin.lc_sin.sin_addr.s_addr=temp->seeder_list[random_piece].sd_sin.sin_addr.s_addr;
sin.lc_sin.sin_port=temp->seeder_list[random_piece].sd_sin.sin_port;

//memcpy(sin.lc_sin,temp->seeder_list[0].sd_sin,sizeof(temp->seeder_list[random_piece].sd_sin));

inet_ntop(AF_INET, &(sin.lc_sin.sin_addr), tempbuff, BUFFSIZE);
#ifdef DEBUG
printf("Random Number generated is: %d\n",random_piece);
#endif
printf("\n Connecting to seeder on IP and Port :%s:%d\n",tempbuff,ntohs(sin.lc_sin.sin_port));
fflush(stdout);

//TODO : exit if no connection is formed

pthread_mutex_unlock(&safe_leech);
//create socket
if((sock = socket(PF_INET,SOCK_STREAM,0))<0){
    printf("ERROR:Socket not created\n");
    exit(1);
}
//connect socket
if(connect(sock,(struct sockaddr*)&sin.lc_sin,sizeof(sin.lc_sin))<0){
    printf("ERROR:Unable to connect to seeder\n");
    return CONNECTERROR;
}
else{
    printf("Connected to seeder with ID:%d\n",temp->seeder_list[0].seeder_id);
}

printf("Downloading %s of size %d ...",newpart,temp->file_size);
sprintf(msg,"%s,%d%c",filename,part_num,'$');
send(sock,msg,strlen(msg),0);//sending the filename and part_num to the seeder
#ifdef DEBUG
printf("\nSent filename and partnum to seeder\n");
fflush(stdout);
#endif
//    memset(&buffer,0,sizeof(buffer));

/*
bytes_read=recv(sock,buffer,BUFFSIZE,MSG_WAITALL);*/
*/
```

San José State University
Computer Engineering Department

```

/*      buffer[bytes_read]='\0';*/
/*      printf("\nReceived :%s\n",buffer);*/
/*      */
/*      //if(!strcmp(buffer,&p_ack)){*/
/*      printf("\nAck P Received");*/
/*      send(sock,&p_ack,sizeof(p_ack),0);*/
/*      */
/*      fflush(stdout);*/
/*      */
/*      //}*/
/*      //send(sock,msg,sizeof(msg),0);*/
/*      //sending the filename and part_num to the seeder

recv(sock,&fp_buff,sizeof(fp_buff),0);
if(fp_buff == 'P'){
    printf("\nReceived %c",fp_buff);
    fflush(stdout);
}
send(sock,&fpa_buff,sizeof(fpa_buff),0);

size_counter=0;

while(size_counter < temp->file_size){
    memset(&buffer,0,sizeof(buffer));
    bytes_read = recv(sock,buffer,BUFSIZE,0);

#ifdef DEBUG
    printf("bytes read %d ....\n",bytes_read);
    fflush(stdout);
#endif

    pthread_mutex_lock(&safe_leech);
    if(fwrite(buffer,sizeof(char),bytes_read,temp->data_ptr) == bytes_read){
        //fwrite(buffer,sizeof(char),bytes_read,temp->data_ptr);
        size_counter += bytes_read;

#ifdef DEBUG
        printf("Bytes written to file so far :%d\n",size_counter);
        fflush(stdout);
#endif

        write_count++;
        send(sock,&p_ack,sizeof(p_ack),0);
    }

    else {
        size_counter -= bytes_read;
        write_count--;
        printf("ERROR writing to file...\n");
        send(sock,&n_ack,sizeof(n_ack),0);
    }

    pthread_mutex_unlock(&safe_leech);
/*      if(bytes_read==0)
    {
        break;
        fclose(temp->data_ptr);
    }*/
}
fclose(temp->data_ptr);
close(sock);
printf("\n Socket closed.. Closing file ptr ... \n");
fflush(stdout);

```

```
pthread_mutex_lock(&safe_leech);

pthread_mutex_unlock(&safe_leech);

pthread_mutex_lock(&safe_leech);
temp->percent_downloaded = 100.00;
//temp->download_status = COMPLETE;
pthread_mutex_unlock(&safe_leech);
//CRC check
printf("\n Download Status Set to 100 ... \n");
fflush(stdout);
return 0;
}

int main_leecher(char *trackerIP,unsigned int trackerport,char * filename, int pieces,int lpsize, int fileSize,int
peer_id,struct file_part *head,crc filecrc_parsed)
{
    int node_number;
    int download_fail=0;
    int connectflag=0;
    int part_downloadflag=0;

#ifdef DEBUG
    printf("\n Main Leecher Started ...");
    fflush(stdout);
#endif
    print_list(head);

    printf("\n Peer ID is :%d",peer_id);
    info_req2track_populate(trackerIP,trackerport,filename,pieces,peer_id,head);

    for(node_number=1;node_number<=pieces;node_number++)
    {
        part_downloadflag=leecher_client(head,filename,node_number);
        printf("\n Download Flag for node_number %d is %d",node_number,part_downloadflag);
        fflush(stdout);
        if(part_downloadflag==CONNECTERROR)
        {
            part_downloadflag=leecher_client(head,filename,node_number);
            if(part_downloadflag==CONNECTERROR)
            {
                printf("\n Seeder not available... clean up the files");
                exit(1);
                fflush(stdout);
            }
        }
        if(part_downloadflag==SUCCESS){
            send_update2tracker(trackerIP,trackerport,filename, node_number,peer_id); //0x01
partial seeding
            if(node_number!=pieces){
                getinfo_part_populate(head,trackerIP,trackerport,filename,node_number+1); //
get latest DS of next part from tracker
            }
        }
    }
    //update complete list
}
```

```
        for(node_number=1;node_number<=pieces;node_number++)
        {
            getinfo_part_populate(head,trackerIP,trackerport,filename,node_number); // get latest DS of
next part from tracker
        }
#ifdef DEBUG
        print_list(head);
#endif
        if(part_downloadflag==SUCCESS){
            download_fail=joinfile(filename,fileSize,pieces,lpsize,filecrc_parsed);

            if(!download_fail){
                return SUCCESS;
            }
            else{
                return FAILURE;
            }
        }
    }
}
```

8.7 seeder.c

/******

Problems faced : passing multiple arguments to thread.
MSG_WAITin recv

*****/

/****** Iterative seeder *****/

```
#include "header.h"
#include "dstruct.h"
```

```
#include<sys/signal.h>
#include<sys/time.h>
#include<sys/resource.h>
#include<sys/wait.h>
#include<sys/errno.h>
```

```
#include<pthread.h>
#include<unistd.h>
```

```
#define SEND_BUFF 1024
```

```
#define BUF_SIZE 128
```

```
#define DEBUG
```

```
static void *argbuff;
```

```
void main_seeder(struct file_part *head,int partial)
{
```

```
    pthread_t    new_thread;
    pthread_attr_t thread_attr;
```

```
/******Doubt should i define it here or change this to take it from DS *****/
    struct sockaddr_in seed_server_sin; //seeder socketaddr struct
```



```
struct sockaddr_in leech_client_sin; // leecher socketaddr struct
int alen,t;
int msocket, ssocket; // master and slave sockets

memset(&seed_server_sin,0,sizeof(seed_server_sin) );
memset(&leech_client_sin,0,sizeof(leech_client_sin) );

seed_server_sin.sin_family = AF_INET;
seed_server_sin.sin_addr.s_addr = INADDR_ANY;
seed_server_sin.sin_port = htons(SEEDERSERVERPORT);

//Create server socket.
msocket = socket(PF_INET, SOCK_STREAM,0);
if (msocket < 0){
    printf("\n Cannot create the socket ");
    fflush(stdout);
    exit(1);
}
// Bind socket
if (bind(msocket, (struct sockaddr*)&seed_server_sin, sizeof(seed_server_sin)) < 0){
    printf("\n ERROR ! Cannot bind the socket ");
    fflush(stdout);
    exit(1);
}
if (listen(msocket, 3) < 0){
    printf("\n ERROR ! Cannot listen the socket ");
    fflush(stdout);
    exit(1);
}
printf("\n Waiting for leecher connections. ");
fflush(stdout);
while (1)
{
    (void) pthread_attr_init(&thread_attr);
    (void) pthread_attr_setdetachstate(&thread_attr, PTHREAD_CREATE_DETACHED);
#ifdef DEBUG
    printf("\n After setting attributes detach state\n");
    fflush(stdout);
#endif
    for(t=0;t<NUMBER_OF_THREADS;t++)
    {
        printf("\nWaiting for leecher requests...\n");
        fflush(stdout);
        alen = sizeof(leech_client_sin);
        ssocket = accept(msocket, (struct sockaddr*)&leech_client_sin, &alen);

        /*
            if (ssocket < 0)
            {
                if (errno == EINTR)
                    continue;
                printf("\naccept: %s\n", strerror(errno));
                fflush(stdout);
            }
        */
#ifdef DEBUG
        printf("\ncreating arguments for threading connections");
        fflush(stdout);
#endif
    }
}
```

```
#endif
/*  argbuff=malloc(sizeof ssocket + sizeof head);
    memcpy(argbuff,&ssocket,sizeof ssocket);
    memcpy(argbuff+sizeof ssocket,head, sizeof head);
*/
/*  seed_arguments[t].socket=ssocket;

    seed_arguments[t].head=head;
    // memcpy(seed_arguments->head,head,sizeof(head));
*/
#ifdef DEBUG
printf("\n Passing  slave Socket Number to function as: %d",ssocket);
fflush(stdout);
/*  printf("\n Copied values of socket: %d",seed_arguments[t].socket);
    fflush(stdout);
*/
#endif
/*  if (pthread_create(&new_thread, &thread_attr, (void * (*)(void *))seeder_server,(void *)
(seed_arguments)) < 0)
    {
        printf("\n ERROR ! Cannot Create new thread. ");
        fflush(stdout);
    }
*/

    switch (fork())//create a new child process to handle requests.
    {
        case 0:// CHILD process

            (void) close(msocket);//Close master socket
            if(partial == 0)
                seeder_server(ssocket,head);
            else
                partial_seeder_server(ssocket);
            (void) close(ssocket);
            exit(0);

        default: /* if parent process */
            (void) close(ssocket);
            break;
        case -1:
            printf("fork");
    }

}

}

}

void seeder_server(int sock,struct file_part *head)
{
    struct file_part *temp = head;
    char ack_buff;
    char p_ack='P';
    char fp_buff = 'P',fnp_buff = 'N';
    char msg[50],buffer[SEND_BUFF];
    char filename[30],part[20];
    int i =0,len=0,part_num,no_of_pieces=0;
```

```
    long int size_counter=0;
    int part_size,read_count=0;
    unsigned long int bytes_read=0;
    char * field1,*field2;
    pthread_mutex_t safe_seed=PTHREAD_MUTEX_INITIALIZER;
#ifdef DEBUG
    printf("\n Received Slave Socket Number as: %d",sock);
    printf("\n Filename in head node :%s", head->filename);
    printf("\nWaiting for file request from leecher.");
    fflush(stdout);
#endif
    recv(sock,msg,sizeof(msg),0); // receive the filename and part no from leecher.
    //parse the incoming message into tokens

#ifdef DEBUG
    printf("\nfile request from leecher received ... parsing %s.",msg);
#endif
    field1=strtok(msg,",");
    field2=strtok(NULL,"$");
    strcpy(filename,field1);
    strcpy(part,field2);
    part_num = atoi(part);
#ifdef DEBUG
    printf("\nfile part requested by leecher is part:%s filename %d.",filename,part_num);
    fflush(stdout);
#endif
    //calculate the number of pieces present in a file
    while(temp){
        no_of_pieces++;
        temp=temp->next;
    }
    temp = head;

    while(temp->part_num != part_num){
        temp=temp->next; // go to the requested file part node
    }
#ifdef DEBUG
    printf("\nfile part in list %s.%d, size %d",temp->filename,temp->part_num,temp->file_size);
    fflush(stdout);
#endif
    //part_size = temp->file_size - (no_of_pieces* PARTSIZE);

    pthread_mutex_unlock(&safe_seed);

    temp->data_ptr = fopen(filename, "rb");
    if(!temp->data_ptr){
        printf("ERROR: Unable to open %s.No such file or directory\n",filename);
        send(sock,&fnp_buff,sizeof(fnp_buff),0);
        close(sock);
        exit(1);
    }

    send(sock,&fp_buff,sizeof(fp_buff),0);
    recv(sock,&fp_buff,sizeof(fp_buff),0);
    if(fp_buff == 'S'){
        printf("\nReceived %c",fp_buff);
        fflush(stdout);
    }
}
```

```
#ifdef DEBUG

    printf("\nCalculating offset");
    fflush(stdout);
#endif
    fseek(temp->data_ptr,0,SEEK_END);
    part_size = (int)ftell(temp->data_ptr);
    rewind(temp->data_ptr);

    //pthread_mutex_lock(&safe_seed);
#ifdef DEBUG
    printf("\nSetting offset");
    fflush(stdout);
#endif
    //moving to the proper offset
    fseek(temp->data_ptr,((part_num-1)*PARTSIZE),SEEK_SET);
#ifdef DEBUG
    printf("\nOffset is :%d",((part_num-1)*PARTSIZE));
    fflush(stdout);
    printf("\nSize_counter initially is :%d and part size:%d",size_counter,temp->file_size);
    fflush(stdout);
#endif

//    send(sock,&p_ack,sizeof(p_ack),0); // Ack that i am strting seeding
    while(size_counter < temp->file_size){
        pthread_mutex_lock(&safe_seed);
#ifdef DEBUG
        printf("\nReading from file...");
        fflush(stdout);
#endif
        bytes_read=0;
        bytes_read = fread(buffer,sizeof(char),SEND_BUFF,temp->data_ptr);
#ifdef DEBUG
        printf("\nBytes read from file :%d",bytes_read);
        fflush(stdout);
#endif
        pthread_mutex_unlock(&safe_seed);

        size_counter += bytes_read;
#ifdef DEBUG
        printf("\nSize Couter is :%d",size_counter);
        fflush(stdout);
#endif
        read_count++;
#ifdef DEBUG
        printf("\nSending data");
        fflush(stdout);
#endif
        send(sock,buffer,bytes_read,0);
        /*if(size_counter == temp->file_size){
            break;
        }*/
        printf("\nwaiting for ack");
        fflush(stdout);
        recv(sock,&ack_buff,sizeof(ack_buff),MSG_WAITALL); // reason for &ack_buff: it is a char , but
        recv needs pointer
    }
```

```
        if(ack_buff == 'P'){
            printf("\nP ack received\n");
            continue;
        }
        else if(ack_buff == 'N'){
            printf("\nN received");
            fflush(stdout);
            fseek(temp->data_ptr,-bytes_read,SEEK_CUR);
            size_counter -= bytes_read;
            read_count--;
        }
    }

    pthread_mutex_lock(&safe_seed);
    fclose(temp->data_ptr);
    pthread_mutex_unlock(&safe_seed);
    close(sock);
    printf("\nSocket Closed\n");
    fflush(stdout);
}

void partial_seeder_server(int sock)//,struct file_part *head)
{
    //struct file_part *temp = head;
    FILE *fp;
    crc filecrc;
    char ack_buff,fp_buff = 'P',fnp_buff = 'N';
    char msg[50],buffer[SEND_BUFF];
    char filename[30],part[20],readfile[30];
    int i =0,len=0,part_num,no_of_pieces=0,size_counter=0;
    int part_size,read_count=0;
    size_t bytes_read=0;
    char * field1,*field2;
    pthread_mutex_t safe_seed=PTHREAD_MUTEX_INITIALIZER;
#ifdef DEBUG
    printf("\n Received Slave Socket Number as: %d",sock);
    //printf("\n Filename in head node :%s", head->filename);
    printf("\nWaiting for file request from leecher.");
    fflush(stdout);
#endif
    recv(sock,msg,sizeof(msg),0); // receive the filename and part no from leecher.
    //parse the incoming message into tokens

#ifdef DEBUG
    printf("\nfile request from leecher received ... parsing %s.",msg);
#endif
    field1=strtok(msg,"");
    field2=strtok(NULL,"$");
    strcpy(filename,field1);
    strcpy(part,field2);
    part_num = atoi(part);

    //calculate crc for this piece
    sprintf(readfile,"%s.part.%d",filename,part_num);
    // filecrc = calculatecrc(readfile);
#ifdef DEBUG
    printf("\nfile part requested by leecher is %s",readfile);
    fflush(stdout);
#endif
}
```

```
#endif
//calculate the number of pieces present in a file
/* while(temp){*/
/*     no_of_pieces++;*/
/*     temp=temp->next;*/
/* }*/
/* temp = head;*/
/* */
/* while(temp->part_num != part_num){*/
/*     temp=temp->next; // go to the requested file part node*/
/* }*/
#ifdef DEBUG
// printf("\nfile part in list %s.%d, size %d",temp->filename,temp->part_num,temp->file_size);
// fflush(stdout);
#endif
//part_size = temp->file_size - (no_of_pieces* PARTSIZE);

pthread_mutex_unlock(&safe_seed);

fp = fopen(readfile, "rb");
if(!fp){//file not present
    printf("ERROR: Unable to open %s.No such file or directory\n",readfile);
    send(sock,&fnp_buff,sizeof(fnp_buff),0);
    close(sock);
    exit(1);
}
send(sock,&fp_buff,sizeof(fp_buff),0);
recv(sock,&fp_buff,sizeof(fp_buff),0);
if(fp_buff == 'S'){
    printf("\nReceived %c",fp_buff);
    fflush(stdout);
}
#ifdef DEBUG
// printf("\nCalculating offset");
// fflush(stdout);
#endif

fseek(fp,0,SEEK_END);
part_size = (int)ftell(fp);
rewind(fp);

//pthread_mutex_lock(&safe_seed);
#ifdef DEBUG
printf("\nPartial Seeder : part size %d",part_size);
fflush(stdout);
#endif
//moving to the proper offset
// fseek(temp->data_ptr,((part_num-1)*PARTSIZE),SEEK_SET);
#ifdef DEBUG
// printf("\noffset is :%d",((part_num-1)*PARTSIZE));
// fflush(stdout);
// printf("\nSize_counter initially is :%d and part size:%d",size_counter,temp->file_size);
// fflush(stdout);
#endif
while(size_counter != part_size){
    pthread_mutex_lock(&safe_seed);
#ifdef DEBUG
printf("\nReading from file...");
```

```
        fflush(stdout);
#endif
        bytes_read = fread(buffer,sizeof(char),SEND_BUFF,fp);
#ifdef DEBUG
        printf("\nBytes read from file :%d",bytes_read);
        fflush(stdout);
#endif
        pthread_mutex_unlock(&safe_seed);

        size_counter += bytes_read;
#ifdef DEBUG
        printf("\nSize Couter is :%d",size_counter);
        fflush(stdout);
#endif
        read_count++;
#ifdef DEBUG
        printf("\nSending data");
        fflush(stdout);
#endif
        send(sock,buffer,bytes_read,0);
        /*if(size_counter == temp->file_size){
            break;
        }*/
        printf("\nwaiting for ack");
        fflush(stdout);
        recv(sock,&ack_buff,sizeof(ack_buff),MSG_WAITALL); // reason for &ack_buff: it is a char , but
recv needs pointer
        if(ack_buff == 'P'){
            printf("\nP ack received\n");
            //continue;
        }
        else if(ack_buff == 'N'){
            printf("\nN received");
            fflush(stdout);
            fseek(fp,-bytes_read,SEEK_CUR);
            size_counter -= bytes_read;
            read_count--;
        }
    }

    pthread_mutex_lock(&safe_seed);
    fclose(fp);
    pthread_mutex_unlock(&safe_seed);
    close(sock);
    printf("\nSocket Closed\n");
    fflush(stdout);
}
```

8.8 sparatorrent.c

```
/******
*spartatorrent.c is a bit torrent like file sharing program (Client Software)
*Input: filename
*Output: File Seeding or Download.
*Author:Ashwin Raut , Aaditya Singhvi , Neeraj Naik , Aditya Kumar Verma
*Date:24th Nov 2010
*SpartaTorrent.All Rights Reserved (c) 2010
```

```
*****/
#include "dstruct.h"
#define BUFF 30
#define DEBUG
#define DEBUG_TEST_VALUES
/*Function to remove \r*/
void rm_nline_char(char *in_string, int size)
{
    int j;

    for(j=0;j<size;j++){
        if(in_string[j]=='\r'){
            in_string[j]='\0';
        }
    }
}

//FREE the linked LIST.

void create_list(struct file_part **headref, char * filename ,int pieces, int piecesize, int lpsize)
{
    struct file_part * new_node,*temp;
    int node_number=0;
    int no_of_nodes=0;
    // For testing these values are hardcoded , normally they will be taken from tracker

    //----
    for(node_number=1;node_number<=pieces;node_number++)
    {
        new_node= (struct file_part *) malloc(sizeof (struct file_part));

        // populate filename
        strcpy(new_node->filename,filename);
        //sprintf(new_node->filename,"%s.part%d",filename,node_number);
        // populate partnumber
        new_node->part_num= node_number;
        // populate part size
        if(node_number==pieces)
        {
            new_node->file_size=lpsize;
        }
        else
        {
            new_node->file_size=piecesize;
        }
        // initialize seeder list and leecher list to zero.
        memset(&new_node->seeder_list,0,sizeof(new_node->seeder_list));
        memset(&new_node->leecher_list,0,sizeof(new_node->leecher_list));

        new_node->next=NULL;

        //set percent download
        new_node->percent_downloaded=00.00;
    }
}
```



```
        if(*headref==NULL)
        {
            *headref=new_node;
            no_of_nodes++;
        }
        else
        { // iterate to the end

            temp=*headref;
            while(temp->next!=NULL)
            {
                temp=temp->next;
            }
            temp->next=new_node;
            no_of_nodes++;
        }
    }

}

void print_list(struct file_part *temp)
{
    char pipbuff[BUFF]={0};
    int i=0;

    while(temp!=NULL){
        printf("\n***The Data in linked list is:***");
        printf("\nFilename: %s",temp->filename);
        printf("\nPart number: %d",temp->part_num);
        printf("\nPart Size: %d",temp->file_size);
        printf("\n Seeder Info:\n");
        for(i=0;i<MAX_SEEDS;i++)
        {
            //***** Change the formatting specifiers according to
length
            printf(" ID:%5d", temp->seeder_list[i].seeder_id);
            inet_ntop(AF_INET, &(temp->seeder_list[i].sd_sin.sin_addr), pipbuff, BUFF);
            printf(" Seeder ip is :%16s:%4d",pipbuff,ntohs(temp->seeder_list[i].sd_sin.sin_port));
            memset(&pipbuff,0,BUFF);
            printf(" | ");
            if((i%3)==2){
                printf("\n");
            }
        }
        printf("\n Leecher Info:\n");
        for(i=0;i<MAX_LEECHES;i++)
        {
            printf(" ID:%5d", temp->leecher_list[i].leecher_id);
            inet_ntop(AF_INET, &(temp->leecher_list[i].lc_sin.sin_addr), pipbuff, BUFF);
            printf(" Leecher ip is :%16s:%4d",pipbuff,ntohs(temp->seeder_list[i].sd_sin.sin_port));
            memset(&pipbuff,0,BUFF);
            printf(" | ");
        }
    }
}
```

```
        if((i%3)==2){
            printf("\n");
        }
        printf("\npercent download is: %f\n",temp->percent_downloaded);
        temp=temp->next;
        printf("*****\n");
    }
}
```

```
int manager(char * argument1)
{
    struct file_part *head=NULL;
    char filename[MAX_FILENAME_LEN]={0};
    char fname_argument[MAX_FILENAME_LEN]={0};
    char *extension;
    unsigned int extension_len=0,length,filename_len;
    char *torrentextension=".torrent";
    crc filecrc;
    int2c crc_union;
    ///////////////////////////////////
    int pid,partial,leech_ret,retry;

    // Variables used to store the parsed data.
    FILE * fp_storrent,*file_present;
    struct sockaddr_in trackersocketaddr;
    char filebuffer_storrent[100]={0};
    char filebuffer1_storrent[100]={0};
    unsigned int filesize_storrent=0;
    char trackerIP[BUFF];
    unsigned int trackerport;
    long int filesize;
    int pieces,piecesize;
    int lpsize=0;
    int peer_id;
    crc filecrc_parsed;
    char * field,*fieldvalue;

    trackersocketaddr.sin_family=AF_INET; // change this for IPV6.

    extension=strstr(argument1,torrentextension);
    strcpy(fname_argument,argument1);
    extension_len=strlen(extension);
    fflush(stdout);
    length=strlen(argument1);
    filename_len=length-extension_len;
    strncpy(filename,fname_argument,filename_len);
    printf("\n The filename parsed is :%s",filename);

    //PARSE THE .STORRENT FILE.
    fp_storrent = fopen(fname_argument , "r");
    if(fp_storrent==NULL)
    {
```

```
        printf("\n\n*****ERROR*****");
        printf("\nFile Not Present.\nPlease download *.torrent file in current folder\n");
        printf("*****\n");
        fflush(stdout);
        exit(1);
    }
    fseek(fp_storrent,0,SEEK_END);
    filesize_storrent=ftell(fp_storrent);
    rewind(fp_storrent);

    //File Parsing
    while(!feof(fp_storrent))
    {
        fgets(filebuffer_storrent,100,fp_storrent);
        rm_nline_char(filebuffer_storrent,strlen(filebuffer_storrent));
        field= strtok(filebuffer_storrent," ");
        fieldvalue= strtok(NULL," ");
/*
        printf("%s",field);
        printf("\n");
        printf("%s",fieldvalue);
        printf("\n");
*/
        if(!strcmp("TrackerIP",field))
        {
            //change this for IPv6
            inet_pton(AF_INET,fieldvalue, &(trackersocketaddr.sin_addr));
            inet_ntop(AF_INET, &(trackersocketaddr.sin_addr), trackerIP, BUFF);

#ifdef DEBUG
            printf("\nTracker ip is :%s",trackerIP);
            fflush(stdout);
#endif
        }
        else if (!strcmp("TrackerPort",field))
        {
            //change this for IPv6
            trackerport=atoi(fieldvalue);
            trackersocketaddr.sin_port=htons(trackerport);

#ifdef DEBUG
            printf("\nTracker port number is :%d",trackerport);
            fflush(stdout);
#endif
        }
        else if (!strcmp("FileSize",field))
        {
            filesize=atol(fieldvalue);

#ifdef DEBUG
            printf("\nFile size is :%ld",filesize);
            fflush(stdout);
#endif
        }
        else if (!strcmp("Pieces",field))
        {
            pieces=atoi(fieldvalue);

#ifdef DEBUG
            printf("\npieces :%d",pieces);
            fflush(stdout);
#endif
        }
    }
}
```

```
    }
    else if (!strcmp("PieceSize",field))
    {
        piecesize=atoi(fieldvalue);
#ifdef DEBUG
        printf("\npiecessize :%d",piecesize);
        fflush(stdout);
#endif
    }
    else if (!strcmp("LastPieceSize",field))
    {
        lpsize=atoi(fieldvalue);
#ifdef DEBUG
        printf("\nlpsize :%d",lpsize);
        fflush(stdout);
#endif
    }
    else if (!strcmp("Filecrc",field))
    {
        filecrc_parsed=atoi(fieldvalue);
#ifdef DEBUG
        printf("\nfilecrc_parsed :%X",filecrc_parsed);
        fflush(stdout);
#endif
    }
} //end of while(feof)
fclose(fp_storrent);
srand(time(NULL));
peer_id=rand()%1000+1;

pid = fork();
if(pid == 0){ //Child process, Leecher
    if((file_present = fopen(filename,"r")) == NULL){ //File not present
#ifdef DEBUG
        printf("\nChild process: file %s not present...Calling Leecher",filename);
        fflush(stdout);
#endif

        //fclose(file_present);
        //info_req2track_populate(trackerIP,trackerport,filename,pieces,peer_id,head);
        create_list(&head,filename,pieces,piecesize,lpsize);
        leech_ret =
        main_leecher(trackerIP,trackerport,filename,pieces,lpsize,filesize,peer_id,head,filecrc_parsed);
        if(!leech_ret){
            printf("\nFile Downloaded successfully!!!");
            fflush(stdout);
#ifdef DEBUG
            printf("\nClosing Child process");
            fflush(stdout);
#endif

            exit(0);
        }
        else{
#ifdef DEBUG
            printf("\nLeecher returned with error");
            fflush(stdout);
#endif
        }
    }
}
```

```

        printf("\nFile Download failed...Trying again");
        if(retry == 1){
            leech_ret =
main_leecher(trackerIP,trackerport,filename,pieces,lsize,filesize,peer_id,head,filecrc_parsed);
            retry = 2;
        }
        printf("\nRetry Failed:Cannot download file");
        fflush(stdout);
        exit(0);
    }
}
else{//File present
#ifdef DEBUG
    printf("\nChild process: file %s is already present...Closing Child process",filename);
    fflush(stdout);

    fclose(file_present);
    exit(0);
}
}
else if(pid > 0){//Parent process
    printf("\nParent process");
    fflush(stdout);
    if((file_present = fopen(filename,"r")) == NULL){//File not present
        partial = 1;

        //create_list(&head,filename,pieces,piecesize,lsize);
        #ifdef DEBUG
            // print_list(head);
        #endif
        //initial_tracker_update(trackerIP,trackerport,filename,pieces,peer_id);
        //fclose(file_present);
        printf("\nStarting partial seeder\n");
        fflush(stdout);
        main_seeder(head,partial);//Call partial seeder
    }
    else{//File present
        partial = 0; // do not start partial seeder but call main seeder
        printf("\nFile %s is present,checking crc ...",filename);
        fflush(stdout);
        filecrc=calculatecrc(filename);
        if(filecrc==filecrc_parsed){
            printf("\nComplete File is present, updating tracker and starting main seeder\n");
            fflush(stdout);
        }
        srand(time(NULL));
        peer_id=rand()%1000+1;
        create_list(&head,filename,pieces,piecesize,lsize);
        #ifdef DEBUG
            print_list(head);
        #endif
        initial_tracker_update(trackerIP,trackerport,filename,pieces,peer_id);
        fclose(file_present);
        main_seeder(head,partial);//Call seeder_server
    }
}
}
}
return 0;

```

```
}//end of int manager()
```

```
void initial_tracker_update(char *trackerIP,unsigned int trackerport,char * filename, int pieces,int peer_id)
{
```

```
    struct sockaddr_in trackersock_in;
    int sock, sinsize;
    sinsize = sizeof(trackersock_in);
    char send_buffer[SOCK_BUFF]={0};
```

```
    memset(&trackersock_in, 0, sizeof(trackersock_in));
    trackersock_in.sin_family = AF_INET;
```

```
    inet_pton(AF_INET,trackerIP, &(trackersock_in.sin_addr));
    trackersock_in.sin_port=htons(trackerport);
```

```
    sprintf(send_buffer,"0$%s$d$d$d",filename,pieces,peer_id,SEEDERSERVERPORT);
```

```
    #ifdef DEBUG
        printf("\nThe Buffer is %s ",send_buffer);
        fflush(stdout);
```

```
    #endif
        /*create socket*/
        if((sock=socket(PF_INET,SOCK_STREAM,0))<0){
            printf("\nFailed to create socket ... \n");
            exit(1);
        }
```

```
    #ifdef DEBUG
        printf("\nConnecting to Tracker at IP: %s , Port %d ",trackerIP,trackerport);
        fflush(stdout);
```

```
    #endif
        /*Connect to server*/
        if(connect(sock,(struct sockaddr *)&trackersock_in,sizeof(trackersock_in)) <0){
            printf("\nFailed to connect to tracker ... \n");
            exit(1);
        }
        /*Send protocol to Tracker*/
```

```
        send(sock,send_buffer,sizeof(send_buffer),0);
```

```
    #ifdef DEBUG
        printf("\nBuffer Sent ... Closing Socket...");
        fflush(stdout);
```

```
    #endif
        close(sock);
    }
```

```
int main(int argc, char * argv[])
{
```

```
    printf("\n**Starting Program**");
    printf("\n the no of arguments is :%d",argc) ;
```

```
    if(argc!=2){
        printf("\n\n*****ERROR*****");
        printf("\nUSAGE: ./sparta [option]\n");
    }
```

```
    printf("\t[option]: filename.extension.storrent\n")    ;
    printf("Example: ./sparta photo.jpg.storrent\n")      ;
    printf("*****\n");
    fflush(stdout);
    exit(1);
}
if(argc>0){
    manager(argv[1]);
}

return 0;
}
```

8.9 tracker.c

```
#define DEBUG
#include "tracker.h"

struct node *head=NULL;
static int Test=0;

//struct file_part *fhead=NULL;

/*****Functions definitions*****/

/* This function just makes Initial 1st time (peer_count = 1), DS popuate for each file,
 * remaining all updates for each part are handled by update function
 */

void append_file_part(struct node **new1, char *fname, int seed_id,int part_number, int myport, char
IP_addr[IPv4_SIZE]) // 6 parameters passed
{
    struct node *temp;
    struct file_part *new_temp, *r;

    //char * f_name;
    char f_name[BUF_SIZE];
    int peer_count = 0;
    temp = *new1 ;

    // f_name = temp->value;

    strcpy(f_name,temp->value);
    //printf("\n Inside Append file part with file name f_name: %s , and fname = %s\n\n ",f_name,
fname);fflush(stdout);

    if ( temp->next ==NULL){

        peer_count = 0;
        if( temp->node_part == NULL)
        {
            new_temp = (struct file_part *)malloc ( sizeof ( struct file_part) ) ;

            new_temp-> next = NULL;
```

```
new_temp-> part_num = part_number;

new_temp-> seeder_list[peer_count].seeder_id = seed_id;
new_temp-> seeder_list[peer_count].sd_sin.sin_port = htons(myport) ; // ???
    inet_pton(AF_INET,IP_addr,&(new_temp-> seeder_list[peer_count].sd_sin.sin_addr));
new_temp-> seeder_list[peer_count].sd_sin.sin_family = AF_INET ;

/*
    Few more fields to put.. like dwld status n all .....
*/

temp->node_part = new_temp;
//printf("\n Inside append_file_part : %s , of part number %d , seeder_IP %s and Port ",
temp->value, new_temp-> part_num,myport);
}

else
{
    new_temp = temp->node_part ;
    while ( new_temp -> next != NULL )
        new_temp = new_temp -> next ;

    r = (struct file_part *)malloc ( sizeof ( struct file_part ) ) ;
    r-> next = NULL;
    r-> part_num = part_number;
    r-> seeder_list[peer_count].seeder_id = seed_id;
    r-> seeder_list[peer_count].sd_sin.sin_port = htons(myport) ;
    inet_pton(AF_INET,IP_addr,&(r-> seeder_list[peer_count].sd_sin.sin_addr));
    r-> seeder_list[peer_count].sd_sin.sin_family = AF_INET ;

    /*
        Few more fields to put.. like dwld status n all .....
    */

    inet_ntop(AF_INET,&(r-> seeder_list[peer_count].sd_sin.sin_addr),IP_addr,IPv4_SIZE); //
presentation
    new_temp-> next = r;
    printf("\n Inside append_file_part : \t%s, \t%d , \t%s, \t%d", temp->value, r-> part_num, IP_addr,
    ntohs(r-> seeder_list[peer_count].sd_sin.sin_port));

}

}

else {
    while ( temp -> next != NULL )
        temp = temp->next;

    peer_count = 0;
    if( temp->node_part == NULL)
    {
        new_temp = (struct file_part *)malloc ( sizeof ( struct file_part ) ) ;

        new_temp-> next = NULL;
        new_temp-> part_num = part_number;

        new_temp-> seeder_list[peer_count].seeder_id = seed_id;
        new_temp-> seeder_list[peer_count].sd_sin.sin_port = htons(myport) ; // ???
```



```
        inet_pton(AF_INET,IP_addr,&(new_temp-> seeder_list[peer_count].sd_sin.sin_addr));
        new_temp-> seeder_list[peer_count].sd_sin.sin_family = AF_INET ;

        /*
            Few more fields to put.. like dwld status n all .....
        */

        temp->node_part = new_temp;
        //printf("\n Inside append_file_part : %s , of part number %d , seeder_IP %s and Port ",
temp->value, new_temp-> part_num,myport);
    }

    else
    {
        new_temp = temp->node_part ;
        while ( new_temp -> next != NULL )
            new_temp = new_temp -> next ;

        r = (struct file_part *)malloc ( sizeof ( struct file_part ) ) ;
        r-> next = NULL;
        r-> part_num = part_number;
        r-> seeder_list[peer_count].seeder_id = seed_id;
        r-> seeder_list[peer_count].sd_sin.sin_port = htons(myport) ;
        inet_pton(AF_INET,IP_addr,&(r-> seeder_list[peer_count].sd_sin.sin_addr));
        r-> seeder_list[peer_count].sd_sin.sin_family = AF_INET ;

        /*
            Few more fields to put.. like dwld status n all .....
        */

        inet_ntop(AF_INET,&(r-> seeder_list[peer_count].sd_sin.sin_addr),IP_addr,IPv4_SIZE); //
presentation
        new_temp-> next = r;
        printf("\n Inside append_file_part : \t%s, \t%d , \t%s, \t%d", temp->value, r-> part_num, IP_addr,
ntohs(r-> seeder_list[peer_count].sd_sin.sin_port));

    }

}

}

}

/***** append function *****/

int append_list ( struct node ** new2, char * name )
{
    struct node *temp, *r ;
    static int i=1;
    if ( *new2 == NULL ) // if the list is empty, create first node
    {
        printf("\n There is no file present , at start");
        fflush(stdout);
    }
}
```

```
temp = (struct node *)malloc ( sizeof ( struct node ) );
strcpy(temp->value,name); //temp -> value = name ;
temp -> next = NULL ;
temp -> node_part = NULL;
*new2 = temp ;
printf("\n Inside append_list ( 1st Node ): %s ", temp->value);
fflush(stdout);
i++;
return 0;
}
else
{
    unsigned int set =0;           // check for duplication of Key

    char *tempstr;
    temp = *new2;

    while ( temp != NULL )
    {
        set = 0;
        tempstr = temp->value;

        if ( strcmp(name,tempstr)==0)
        {
            printf("\n File info already present in the tracker \n");
            return 1;
            break;
        }
        else
        {
            temp = temp -> next ;
            set = 1;
        }
    }

    if( set ==1)
    {
        temp = *new2 ;

        while ( temp->next!= NULL ) // go to last node
        {
            temp = temp -> next ;
            printf("\nIncrementing node in appendix list..... %d",i);
            fflush(stdout);
            i++;
        }
        //printf("\n Appending ith %d at next location current location : %s ", i,temp->value);
        //fflush(stdout);
        r = (struct node *)malloc ( sizeof ( struct node ) ) ; // add node at the end
        strcpy(r->value, name ); // r -> value = name ;
        r -> next = NULL ;
        r -> node_part = NULL;
        temp->next = r ;
        printf("\n Inside append_list (other parts).....: %s ", r->value);
        fflush(stdout);
        return 0;
    }
}
```

```
    } // inner if ended

} // else ended.

}

/***** Update function *****/

void Update ( struct node **new1, char *fname, char * protocol, int p_id, int part_number, int myport, char
IP_addr[IPv4_SIZE]) // 7 parameters
{
    struct node *temp;
    struct file_part *new_temp, *r;
    char f_name[BUF_SIZE], protocol_code;
    int i= 0;

    int peer_count ;
    temp = *new1 ;

    protocol_code = protocol;

    strcpy(f_name,temp->value) ;//f_name = temp->value;

    printf("\n Inside update function .. \n");

    printf("\n f_value %s, temp->value %s, fname %s \n", f_name, temp->value, fname); fflush(stdout);
    printf("\n Planning for strcmp "); fflush(stdout);

    while ( strcmp((char *)fname, (const char*)f_name)!= 0)
    {
        temp = temp->next;
        strcpy(f_name,temp->value); // f_name = temp->value;
        printf("\n inside while ....");

        if(temp->next == NULL && (strcmp((char *)fname, (const char*)f_name) != 0 ))
        {
            printf("\n Sorry ! File not present in the file_list. Needs to be added ");
            //getch();
            exit(1);
        }
    }

    printf("\n Inside update f_name is matched temp->value = %s",temp->value); fflush(stdout);

    new_temp = temp->node_part ;
    printf("\n Debug -1\n"); fflush(stdout);
    if (part_number != 1)
    {
        for ( i = 1 ; i < part_number ; i++ )
            new_temp = new_temp -> next ;
    }
}
```

```
}

printf("\n new_temp->part_num = %d, part_number = %d", new_temp->part_num, part_number);
fflush(stdout);
printf("\n .... Debug -2\n"); fflush(stdout);

/* new_temp = temp->node_part ;

for ( i = 1 ; i < part_number ; i++ )
    new_temp = new_temp -> next ;

if ( new_temp == NULL )
{
    printf ( "\nThere are less than %d elements in list", part_number ) ;
    return ;
}

printf("\n.... new_temp->part_num = %d, part_number = %d ..... \n ", new_temp->part_num,
part_number ); */

if ( new_temp->part_num == part_number)
{

if ( protocol_code == 0x01) // seeder update
{
    for (peer_count=0;peer_count< MAX_PEERS;) // Reach
    {
        printf("\n peer id is %d", new_temp->seeder_list[peer_count].seeder_id); fflush(stdout);
        if((new_temp->seeder_list[peer_count].seeder_id == 0) ||( new_temp-
>seeder_list[peer_count].seeder_id == p_id) )
        {
            break;
        }
        peer_count++;
    }

    if (peer_count < MAX_PEERS)
    {
        printf("\n update .. protocol 0x01... \n peer count is %d \n", peer_count); fflush(stdout);
        new_temp-> seeder_list[peer_count].seeder_id = p_id;
        new_temp-> seeder_list[peer_count].sd_sin.sin_port = htons(myport) ;
        //new_temp-> seeder_list[peer_count].sd_sin.sin_addr.s_addr = IP_addr ;
        inet_pton(AF_INET,IP_addr,&(new_temp-> seeder_list[peer_count].sd_sin.sin_addr));
        new_temp-> seeder_list[peer_count].sd_sin.sin_family = AF_INET ;
    }
    else
        printf("\n Max Peer in the Swarm reached. No more Peers can be added ");
}

else // leecher update
{
```

```

        for (peer_count=0;peer_count< MAX_PEERS;peer_count++) // Reach
        {
            if(new_temp->leecher_list[peer_count].leecher_id == NULL ); // Initialise somewhere ,
this array as null. = {0};
            break ;
        }
        new_temp-> leecher_list[peer_count].leecher_id = p_id;
        new_temp-> leecher_list[peer_count].lc_sin.sin_port = htons(myport) ; // ??? or just
myport;
        //new_temp-> leecher_list[peer_count].lc_sin.sin_addr.s_addr = IP_addr ;
        inet_pton(AF_INET,IP_addr,&(new_temp-> leecher_list[peer_count].lc_sin.sin_addr));
        new_temp-> leecher_list[peer_count].lc_sin.sin_family = AF_INET ;
    }

    }

    //printf("\n\n Percentage downloaded for part number %d is %f ",part_number, new_temp-
>percent_downloaded);
    printf("\n\n Its peer id is : %d \n\n ",new_temp->seeder_list[peer_count].seeder_id);
}

void print_list(struct node *head)
{
    struct node * temp=head;
    struct file_part * horizontal;
    int i=0;

    char pipbuff[IPv4_SIZE]={0};

    if (temp != NULL)
    {
        //temp->value;
        //printf("\n Head value %s",temp->value);
        while(temp!=NULL)
        {
            printf("\n ***** Head value %s*****",temp->value);
            fflush(stdout);
            if(temp->node_part==NULL)
                printf("\n The horizontal node is NULL");
            if ( temp->node_part != NULL)
            {
                horizontal = temp->node_part;
                while ( horizontal != NULL)
                {

                    printf("\n***The Data in part list of file is:***");
                    printf("\nPart number: %d",horizontal->part_num);

                    printf("\n Seeder Info:\n");
                    for(i=0;i<MAX_SEEDS;i++)
                    {
                        //***** Change the formatting specifiers according to
length
                        printf(" ID:%5d", horizontal->seeder_list[i].seeder_id);
                        inet_ntop(AF_INET, &(horizontal->seeder_list[i].sd_sin.sin_addr), pipbuff, IPv4_SIZE);
                        printf(" Seeder ip is :%16s:%4d",pipbuff,htons(horizontal-
>seeder_list[i].sd_sin.sin_port));

```

```

        memset(&pipbuff,0,IPv4_SIZE);
        printf(" | ");
        if((i%3)==2)
        {
            printf("\n");
        }
    }
    printf("\n Leecher Info:\n");
    for(i=0;i<MAX_LEECHES;i++)
    {
        printf(" ID:%5d", horizontal->leecher_list[i].leecher_id);
        inet_ntop(AF_INET, &(horizontal->leecher_list[i].lc_sin.sin_addr), pipbuff, IPv4_SIZE);
        printf(" Leecher ip is :%16s:%4d",pipbuff,htons(horizontal-
>seeder_list[i].sd_sin.sin_port));
        memset(&pipbuff,0,IPv4_SIZE);
        printf(" | ");
        if((i%3)==2)
        {
            printf("\n");
        }
    }

    horizontal=horizontal->next; // go to next horizontal node (FILE pART)
    printf("*****\n");
} //end of while

}

temp=temp->next;
printf("\n ***** .....File Ended
..... *****\n");
fflush(stdout);
} //end of while vertical
}

}

/***** Read_DS_and_send *****/

char* Read_DS ( struct node **new1, char *fname, int seq_no, int part_number )
{
    struct node *temp;
    struct file_part *new_temp, *r;
    //char * f_name;
    char f_name[BUF_SIZE];
    int i= 0;
    char send_buffer[512]={0};
    char token3[100];
    int peer_id, port_no;
    char IP_addr[IPv4_SIZE];
    char* reply_protocol = "0x04";

    char send_this[512]= {0};
    char s_buff[64];

    int ack_no = seq_no +1;
    char ss_buff[10] = "FNF";
    temp = *new1 ;

```

```
        if(*new1==NULL){
            strcpy(send_buffer,ss_buff);
            return send_buffer;
        }
// printf("\n\n .... ack_no... = %d, fname = %s ", ack_no, fname);
printf("\n\n ***** PART NUMBER = %d *****", part_number);

printf("\n Inside Read DS .... \n"); fflush(stdout);
strcpy(f_name, temp->value); // f_name = temp->value;

printf("\n f_value %s, temp->value %s, fname %s \n", f_name, temp->value, fname); fflush(stdout);
printf("\n Planning for strcmp "); fflush(stdout);

while ( strcmp((char *)fname, (const char*)f_name)!= 0)
{
    temp = temp->next;
    strcpy(f_name,temp->value); // f_name = temp->value;
    printf("\n inside while ....");

    if(temp->next == NULL && (strcmp((char *)fname, (const char*)f_name) != 0 ))
    {
        printf("\n Sorry ! File not present in the file_list. Needs to be added ");

        sprintf(send_buffer,"%s",ss_buff); // IF file not found, send the ERROR CODE " FNF " to
tracker .

        printf("\n\n send buffer is : %s\n", send_buffer);
        return send_buffer;
        //exit(1);
    }
}

new_temp = temp->node_part ;
printf("\n Debug -1\n"); fflush(stdout);
if (part_number != 1)
{
    for ( i = 1 ; i < part_number ; i++ )
        new_temp = new_temp -> next ;
}

printf("\n Debug -3 ..... \n"); //fflush(stdout);
printf("\n new_temp->part_num = %d, part_number = %d", new_temp->part_num, part_number);
fflush(stdout);

strcpy((char *)send_this,(const char *) "");
printf("\n Debug -4\n"); fflush(stdout);

if ( new_temp->part_num == part_number) // Need to read always seeder info .. reqd for seeding.
{
```

```

    printf("\n Debug -5\n"); fflush(stdout);
    for( i=0;i< 10 ;i++)
    {
        if(new_temp->seeder_list[i].seeder_id == 0)
        {
            break;
        }

        printf("\n Peer_id read from DS is %d",new_temp->seeder_list[i].seeder_id);

        peer_id = new_temp->seeder_list[i].seeder_id;
        port_no = ntohs(new_temp->seeder_list[i].sd_sin.sin_port);
        // IP_addr = new_temp->seeder_list[i].sd_sin.sin_addr.s_addr;
        inet_ntop(AF_INET,&(new_temp->seeder_list[i].sd_sin.sin_addr),IP_addr,IPv4_SIZE); // presentation

        printf("\n\n Peer id [%d] = %d, port_no [%d] = %d , IP [%d] = %s ", i, new_temp->seeder_list[i].seeder_id, i,
        ntohs(new_temp->seeder_list[i].sd_sin.sin_port), i, IP_addr);
        fflush(stdout);
        sprintf(s_buff, "%d:%s:%d:",new_temp->seeder_list[i].seeder_id,IP_addr,ntohs(new_temp-
        >seeder_list[i].sd_sin.sin_port));
        strcat((char *)send_this,(const char *) s_buff);

    }

    }

    printf("\n\n .... Send_this = %s ", send_this);

    sprintf(send_buffer,"%s$d%s$d%s",reply_protocol,ack_no,fname,part_number,send_this); // is
    protocol code %s or %u or %d ?? also IP YES ??

    // sprintf(send_buffer,"%s$d%s$d",reply_protocol,ack_no,fname,peer_id); // FILL
    MORE ....

    printf("\n\n send buffer is : %s\n", send_buffer);
    return send_buffer;

}

int main(void)
{
    int alen,t;
    char IP_addr[IPv4_SIZE];

    pthread_t new_thread[NUMBER_OF_THREADS];
    pthread_attr_t thread_attr;

    /***** Proxy_Server data structures *****/

    struct sockaddr_in tracker_sin; /* Proxy server endpoint */
    struct sockaddr_in client_sin; /* an SMTP Client endpoint */
    struct servent *tracker_pse; /* pointer to port, currently not used in code */
    int msocket, ssocket; /* socket descriptor and socket type */

    /*****/

```



```
memset(&tracker_sin,0,sizeof(tracker_sin) );
memset(&client_sin,0,sizeof(client_sin) );
printf("\n Starting my Sparta Torrent Tracker ...\n");

/*****P_Server side *****/

tracker_sin.sin_family = AF_INET;
tracker_sin.sin_addr.s_addr = INADDR_ANY;
tracker_sin.sin_port = htons(TRACKER_PORT);

/*Create TCP master Socket for Proxy Server*/
msocket = socket(PF_INET, SOCK_STREAM,0);
if (msocket < 0)
    printf("\n Cannot create the socket ");

/* Bind the master socket */
if (bind(msocket, (struct sockaddr *)&tracker_sin, sizeof(tracker_sin)) < 0)
    printf("\n ERROR ! Cannot bind the socket ");

if (listen(msocket, 3) < 0)
    printf("\n ERROR ! Cannot listen the socket ");

while (1)
{
    //printf(" \n Inside while: Waiting for connections\n");
    printf(" \n Creating New Thread ..\n");

    (void) pthread_attr_init(&thread_attr);
    (void) pthread_attr_setdetachstate(&thread_attr, PTHREAD_CREATE_DETACHED);

    for(t=0;t<NUMBER_OF_THREADS;t++)
    {
        alen = sizeof(client_sin);
        printf("\n Waiting at accept call ..... \n\n");
        fflush(stdout);
        ssocket = accept(msocket, (struct sockaddr *)&client_sin, &alen);

        printf("\n Socket accepted .... \n\n ");
        fflush(stdout);

        if (ssocket < 0)
        {
            if (errno == EINTR)
                continue;
            printf("accept: %s\n", strerror(errno));
        }

        thread_data_array[t].ssocket = ssocket;
        // thread_data_array[t].seeder_list[t].sd_sin.sin_port = client_sin.sin_port ;
        thread_data_array[t].client_sin.sin_addr.s_addr = client_sin.sin_addr.s_addr ;
        thread_data_array[t].client_sin.sin_family = AF_INET ;

#ifdef DEBUG
```

```
    inet_ntop(AF_INET,&(thread_data_array[t].client_sin.sin_addr),IP_addr,IPv4_SIZE); // presentation
    printf("\n its IP is : %s ", IP_addr);
    fflush(stdout);
    #endif
    if(head!=NULL)
    {
        printf("\n Very Initial Value of Head is %s",head->value);

        fflush(stdout);
    }
    if (pthread_create(&new_thread[t], &thread_attr,communicate,(void *)&thread_data_array[t]) < 0)
        printf("\n ERROR ! Cannot Create new thread. ");

    printf("\n Before passing to function .... ssocket is : %d, ... %d", thread_data_array[t].ssocket,
ssocket);

    }

} // while end
}

void * communicate(void * thread_arguement)
{
    int r, i ,buff_len;
    char buff[BUF_SIZE]={0}, cp[BUF_SIZE]={0};
    char *filename;
    //char filename[BUF_SIZE];
    //unsigned char * protocol_feild;
    //unsigned char protocol_feild[BUF_SIZE];
    unsigned char *protocol_feild;
    unsigned int protocol=9;

    const char delimiter[] = "$";
    struct sockaddr_in cli_sin;

    char *no_of_parts, *peer_id, *port_num, *part_no_available, *seeder_id, *leecher_id , *seq_no, *send_buffer;

    //char no_of_parts[BUF_SIZE], peer_id[BUF_SIZE], port_num[BUF_SIZE], part_no_available[BUF_SIZE],
    seeder_id[BUF_SIZE], leecher_id[BUF_SIZE] , seq_no[BUF_SIZE], send_buffer[BUF_SIZE];

    int total_parts, seed_id, port_here, part_available, s_id, l_id, sequence_no ;

    struct thread_data *threaddata;

    char IP_address[IPv4_SIZE];
    int bufflen=BUF_SIZE;
    int file_present;

    int new_socket;
    char ack;
    threaddata = (struct thread_data*)thread_arguement;
    new_socket=threaddata->ssocket;

    cli_sin.sin_addr.s_addr=threaddata->client_sin.sin_addr.s_addr;
    //memcpy(&cli_sin,threaddata->client_sin,sizeof(threaddata->client_sin.sin_addr));
```

```
printf("\n Inside communicate function ");
printf("\n Inside function .... ssocket is : %d", new_socket);
fflush(stdout);
if(Test>0)
    {
        printf("\n The initial Value of Head is %s",head->value);

        fflush(stdout);
    }

memset(&buff,0,BUF_SIZE);
inet_ntop(AF_INET,&(cli_sin.sin_addr),IP_address,IPv4_SIZE); // presentation

#ifdef DEBUG
    printf("\n IP, passed inside function is : %s ", IP_address);
    fflush(stdout);
#endif

/* start reading buffer for Communication Protocol */
// printf("\n Now reading the buffer .....");
// fflush(stdout);

r = read(new_socket,buff, buflen-1);
if(r<0){
    printf("\n Error in Reading from Peers");
}
// printf("\n Read data ... r is:%d",r);
// fflush(stdout);
buff[r]='\0';
// printf("\nReceived Msg :%s",buff);
// printf("\n Received Buffer , Now copying input to other buffer => tokenizing..");
// fflush(stdout);
printf("\n buffer is %s", buff);
fflush(stdout);

strcpy((char*)cp,(const char *)buff);
printf("\n before protocol..... : %s",cp);
fflush(stdout);

protocol_feild = strtok(buff,delimiter);
// strncpy(&protocol, protocol_feild, 1);
protocol = atoi(protocol_feild);

//protocol = strtok(buff,delimiter);
//printf("\n Buffer %s copied.. received token: %x",buff,protocol);
//fflush(stdout);

switch (protocol)
{
    case 0x00: // 1st time populate DS. Msg format : [ protocol $ filename $ no_of_parts $ peer_id $
port_num ] // Always by Seeder
        // No Reply send from tracker
#ifdef DEBUG
            printf("\n Inside protocol 0x00.....\n\n");
            fflush(stdout);
#endif
}
```

```
filename = strtok(NULL, delimiter);
no_of_parts = strtok(NULL, delimiter);
//total_parts = atoi(no_of_parts);
sscanf (no_of_parts, "%d", &total_parts);

peer_id = strtok(NULL, delimiter);
sscanf (peer_id, "%d", &seed_id);

port_num = strtok(NULL, delimiter);
sscanf (port_num, "%d", &port_here);
if(Test>0)
{
    printf("\n The Start Value of Head(before append) is %s",head->value);

    fflush(stdout);
}

printf("\n..... Before append_list ..... \n"); fflush(stdout);
file_present = 0;
printf("\n\n\n file present value..... = %d", file_present);
file_present = append_list(&head,filename);
printf("\n\n\n Return file present value..... = %d", file_present);
printf("\n..... After append_list ..... \n"); fflush(stdout);

printf("\n The Start Value of Head is %s",head->value);
Test++;
fflush(stdout);

if(file_present == 0)
{
    printf("\n\n\n Inside ..... file present value..... = %d", file_present);

    for(i=1 ;i <= total_parts;i++)
    {
        append_file_part( &head, filename, seed_id, i , port_here, IP_address); //
        IP & port need to taken from struct client_sin ( 6 parameter pass)
    }

    printf("\n The Start Value of Head is..... after append had been
called %s",head->value);
    fflush(stdout);

    if(file_present == 0)
    {
        print_list(head);
    }
    else
    {
        printf("\n\n File Information for %s already exists on Tracker, Updating Tracker
",filename);
        for(i=1 ;i <= total_parts;i++)
        {
            Update( &head, filename, 1, seed_id, i , port_here, IP_address);
        }
    }
}
```

```
    }

    printf("\n The Start Value of Head is %s",head->value);
    fflush(stdout);
    break;

case 0x01: // Seeder Update. Msg format : [ protocol $ filename $ part_no_available $ seeder_id $
port_num ]
    // No Reply send from tracker

#ifdef DEBUG
    printf("\n Inside protocol 0x01");
    fflush(stdout);
#endif
    filename = strtok(NULL, delimiter);
    part_no_available = strtok(NULL, delimiter);
    sscanf (part_no_available, "%d", &part_available);

    seeder_id = strtok(NULL, delimiter);
    sscanf (seeder_id, "%d", &s_id);

    port_num = strtok(NULL, delimiter);
    sscanf (port_num, "%d", &port_here);
    printf("\n ..... Before update called.....\n");

    Update( &head, filename, protocol, s_id, part_available, port_here, IP_address); // 7
parameters passed
    //break;
    printf("\n ..... After update called.....\n");
    print_list(head);
    break;

case 0x02: // Get Update for that file part. Msg format : [ protocol $ seq_num $ filename $ part_num ]
    // No reply send from tracker

/
*****
*****

C) 0x02 get info on file part request (This is request from leecher, requesting the latest seeder info on specific
part)

(Assuming 2 seeders are present)

2$seq_no$filename$part_num$

REPLY from tracker to leecher:
2$seq_no$filename$part_num$peer_id:seeder1IPaddr:port,peer_id:seeder2IPaddr:port$

ACTION on tracker : read fields for file part from DS and send reply.
*****
*****/

#ifdef DEBUG
```

```
        printf("\n Inside protocol 0x02.. actually");
        fflush(stdout);
#endif

    seq_no = strtok(NULL, delimiter);
    sscanf (seq_no, "%d", &sequence_no);

    filename = strtok(NULL, delimiter);

    part_no_available = strtok(NULL, delimiter);
    sscanf (part_no_available, "%d", &part_available);

    /*
        leecher_id = strtok(NULL, delimiter);
        sscanf (leecher_id, "%d", &l_id);

        port_num = strtok(NULL, delimiter);
        sscanf (port_num, "%d", &port_here); */

    // Update( &head, filename, protocol, s_id, part_available, port_here, IP_address); // 7 parameters
passed

        send_buffer = Read_DS(&head,filename, sequence_no,part_available);

    r = strlen(send_buffer);
    printf("\n Size of buffer to be send is : %d",r);

    if ( write(new_socket,send_buffer,r)<0)
    {
        printf("\n Error ! Writng to peer Server socket failed ");
        break;
    }

    break;

    case 0x03: // Reterive info request. Msg format : [ protocol $ seq_no $ filename $ no_of_parts ] // Always
by leecher

        // Reply from the tracker. Reply format : [ protocol $ ack_no [seq_no + 1] $ filename $ part_num $
seeder_id-1:port_no-1:IP-1, seeder_id-2:port_no-2:IP-2,seeder_id-3:port_no-3:IP-3, ..... ] here ip n port will be
read from DS. This reply is send for each part ( its for loop)

#ifdef DEBUG
        printf("\n Inside protocol 0x03");
        fflush(stdout);
#endif

        seq_no = strtok(NULL, delimiter);
        sscanf (seq_no, "%d", &sequence_no);
        //printf("\n debug -1 ... %d",sequence_no); fflush(stdout);

        filename = strtok(NULL, delimiter);
        //printf("\n debug -1 ... %s",filename); fflush(stdout);

        no_of_parts = strtok(NULL, delimiter);
        sscanf (no_of_parts, "%d", &total_parts);
        printf("\n Before calling Read ... total parts %d", total_parts); fflush(stdout);
```

```
for(i=1;i<=total_parts;i++)
{
    //printf("\n debug -3"); fflush(stdout);
    send_buffer = Read_DS(&head,filename, sequence_no,i);

    //send_buffer='\0';

    r = strlen(send_buffer);
    printf("\n Size of buffer to be send is : %d",r);

    if ( send(new_socket,send_buffer,r,0)<0)
    {
        printf("\n Error ! Writng to peer Server socket failed ");
        break;
    }
    printf("\n Waiting for ACK...");
    recv(new_socket,&ack,sizeof(ack),0);
    printf("\n Ack Received is ..... %c",ack);
    memset(&ack,0,sizeof(ack));

}

} // Switch ends

printf("\n\n Exiting Thread Now ..... \n\n");
fflush(stdout);
pthread_exit(NULL);
}
```

