Presentation

OurTorrent

PRESENTED TO

Prof Sandeep Kumar Garg

PRESENTED BY

Group-6

CSN-361 Project

CONTENTS

TEAM DETAILS AND CONTRIBUTION

MEET SINDHAV
22114053
meet_js@cs.iitr.ac.in
+91-7984221079

Contribution:

Worked on implementation of client part of the project. Worked on establishing communications between peers and eventual downloading. Also worked on managing file pieces.

MOHAMMED HAAZIQ JAMAL
22114055
mohammed_hj@cs.iitr.ac.in
+91-8126061554

Contribution:

Worked on handling of connect and announce requests by the peer.Also on writing functions related to storing data about leechers and seeders in database using the sqlite3 database.

MUNDADA ADITYA MAHESH
22114058
mundada_am@cs.iitr.ac.in
+91-7720079142

Contribution:

Worked on implementing UDP tracker: handling connection requests and announce requests, establishing client-tracker communication. Also, involved in overall code clarification.

NAYAN RATAN KAKADE
22114060
nayan_rk@cs.iitr.ac.in
+91-7420936972

Contribution:

Worked on parsing of the torrent file using Bencoding Library. Also worked on connection of peer and tracker which included sending connect request and announce request to tracker and retrieving list of peers as response.

SARVESH RUPESH BAHETI
22114087
sarvesh_rb@cs.iitr.ac.in
+91-9767547468

Contribution:

Worked on socket utilities that are required between tracker and peer communication. Also, on setting up the UDP tracker.

ROOPAM TANEJA
22125030
roopam_t@cs.iitr.ac.in
+91-9810788629

Contribution:

Implemented cutom bencoding library. Also worked on implementing client part. Managed communication with multiple peers and succesful downloading of files.

INTRODUCTION

Presenting **OurTorrent-** a Bittorrent inspired P2P file sharing system to share files between a set of peers without the need of any central server.

We have built this project in two phases:

1. Building a **Bittorrent client** that can read any torrent file and request the tracker to give information of peers that hold pieces of the actual file we wish to download.
2. A lightweight **UDP Tracker** to store necessary attributes of peers and dynamically updates them as they enter and exit the network.

FEATURES

P2P File Sharing

The system enables sharing files among peers over a torrent network

Tracker

Peers can request the tracker for details of where to find required files

Bencoding Library

A small, single-header C++ serialization library for bencoded data.

Client Tracker connection

Client Tracker interaction enabled using UDP protocol

CODE SNIPPETS

Client

Peer Class

Manages information about peers participating in file sharing within a BitTorrent network. It initializes peer-related attributes like IP, port, torrent instance etc

peer.py

bittorrent > src > peer.py > peer

```python
# peer.py handles all info about peers participating in file sharing
# and manages communication with them

import time
import struct
import hashlib
from threading import *
from copy import deepcopy

from torrent_logger import *
from peer_messages import *
from peer_socket import *
from peer_state import *

class peer():

    def __init__(self, peer_IP, peer_port, torrent, init_peer_socket = None):
        # peer IP, port and torrent instance
        self.IP          = peer_IP
        self.port        = peer_port
        self.torrent     = deepcopy(torrent)

        self.state = peer_state()
        self.unique_id  = '(' + self.IP + ' : ' + str(self.port) + ')'
        self.peer_id = None
        self.max_block_length = torrent.block_length

        # handshake flag with peer
        self.handshake_flag = False

        # bitfield representing which data file pieces peer has
        self.bitfield_pieces = set([])

        # peer socket for communication
        self.peer_sock = peer_socket(self.IP, self.port, init_peer_socket)

```

Peer-to-Peer Communication

peer_wire_message class manages the creation and representation of peer messages

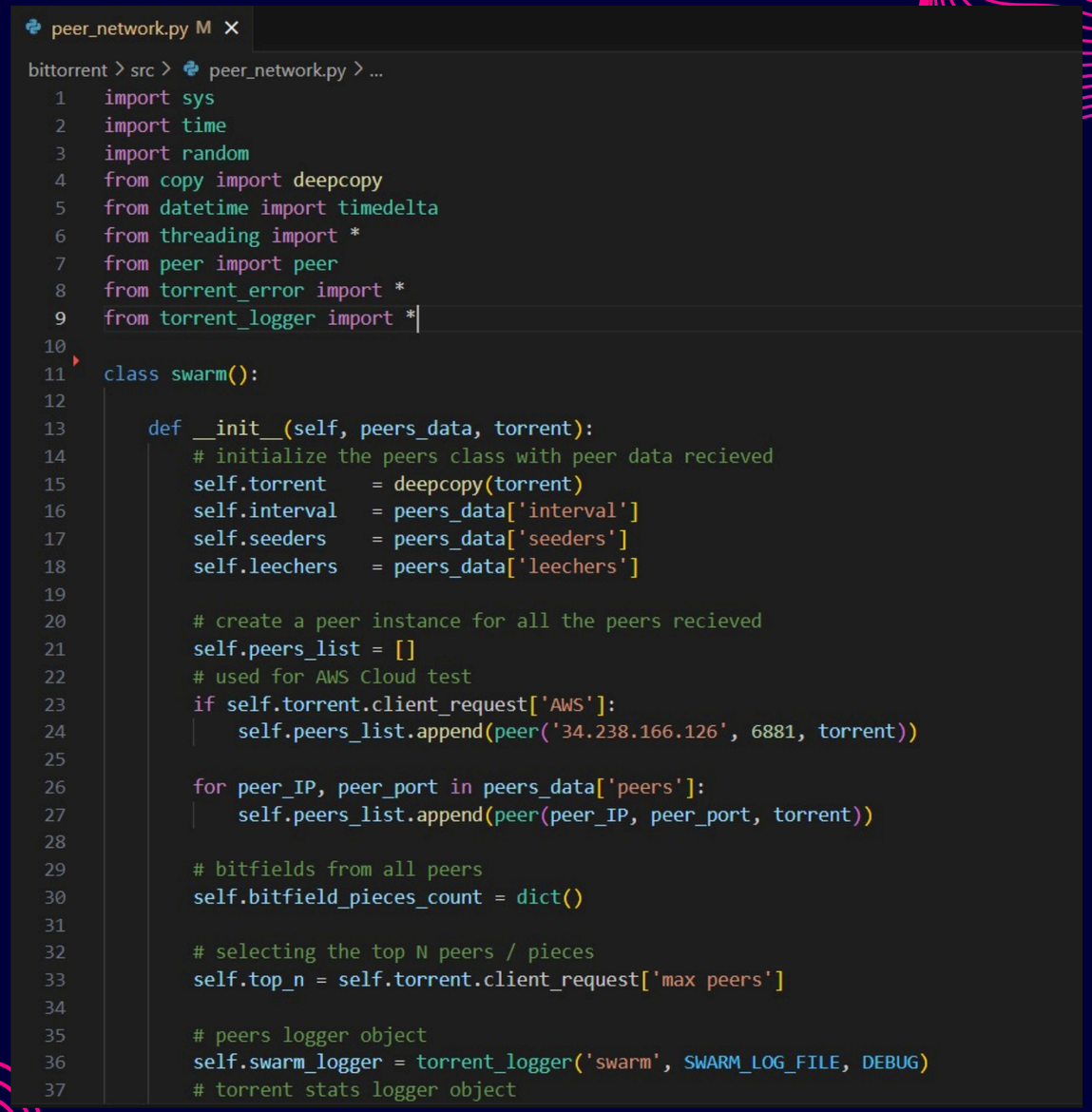
peer_messages.py M

bittorrent > src > peer_messages.py > peer_wire_message > message

```python
# class for general peer message part of peer wire protocol
class peer_wire_message():
    def __init__(self, message_length, message_id, payload):
        self.message_length = message_length
        self.message_id     = message_id
        self.payload        = payload

    # returns raw bytes as peer message
    def message(self):
        message  = struct.pack("!I", self.message_length)
        if self.message_id != None:
            message += struct.pack("!B", self.message_id)
        if self.payload != None:
            message += self.payload
        return message

    # printing the peer wire message
    def __str__(self):
        message  = 'PEER WIRE MESSAGE : '
        message += '(message length : ' +  str(self.message_length) + '), '
        if self.message_id is None:
            message += '(message id : None), '
        else:
            message += '(message id : ' +  str(self.message_id)       + '), '
        if self.payload is None:
            message += '(protocol length : None)'
        else:
            message += '(payload length : ' +  str(len(self.payload)) + ')'
        return message



# class for creating handshake messages between the peers
class handshake():
    # initialize the handshake with the paylaod
    def __init__(self, info_hash, client_peer_id):
        self.protocol name = "BitTorrent protocol"
```

Handling Multiple Peers

Manages a group of peers in a BitTorrent network, tracking data such as seeders, leechers, and peer bitfields



Serialization Library

Bencoder

The serialization library is responsible for encoding metadata for torrent files

```cpp
namespace bencoding
{
    struct string_subs
    {
    public:
        std::string str;
        std::string::const_iterator citer;
        string_subs(std::string input) : str{input}, citer{str.begin()} {}
        void refresh() { citer = str.begin(); }
    };

    class bencode_base
    {
    public:
        static const char delimiter_token = ':';
        static const char end_token = 'e';
        static const char integer_token = 'i';
        static const char list_token = 'l';
        static const char dict_token = 'd';

        bencode_base() = default;
        bencode_base(const bencode_base &) = default;
        bencode_base &operator=(const bencode_base &) = default;
        bencode_base(bencode_base &&other) noexcept = default;
        bencode_base &operator=(bencode_base &&other) noexcept = default;

        virtual ~bencode_base() = default;

        virtual std::string encode() const = 0;
        virtual void encode_n_dump(std::ostream &out) const = 0;
        virtual void decode(const std::string &in, std::string::const_iterator &start) = 0;
        virtual std::string get_as_str() const = 0;
    };
```

```cpp
class bencode_integer : public bencode_base
{
public:
    using value_type = int64_t;

    bencode_integer() = default;
    bencode_integer(value_type value) : integer_value_{value} {}

    bencode_integer &operator=(value_type value)
    {
        integer_value_ = value;
        return *this;
    }
    value_type get() const { return integer_value_; }

    std::string encode() const override
    {
        return std::string(1, integer_token) + std::to_string(integer_value_) + std::string(1, end_token);
    }
    void encode_n_dump(std::ostream &out) const override
    {
        out << integer_token << integer_value_ << end_token;
    }
    void decode(const std::string &in, std::string::const_iterator &start) override
    {
        start++; // Assuming *start == 'i'
        value_type value = 0;
        bool neg = false;
        if (*start == '-')
            neg = true, start++;
        while (start != in.end() && *start != end_token)
            value = value * 10 + (*start - '0'), start++;
        integer_value_ = neg ? -value : value;
        if (start != in.end())
            start++; // assuming it ends with 'e'
    }
    std::string get_as_str() const override
    {
        return std::to_string(integer_value_);
    }
private:
    value_type integer_value_{0};
};
```

```cpp
std::unique_ptr<bencode_base> make_value(const std::string &in, std::string::const_iterator &start);

template <typename T>
concept bencodeDerived = std::is_base_of<bencode_base, T>::value;

class bencode_list : public bencode_base
{
public:
    using value_ptr_type = std::unique_ptr<bencode_base>;
    using container_type = std::vector<value_ptr_type>;
    using iterator = container_type::iterator;
    using const_iterator = container_type::const_iterator;

    bencode_list() = default;
    bencode_list(size_t size_) : bencode_list() { list_.resize(size_); }

    iterator begin() { return list_.begin(); }
    const_iterator begin() const { return list_.begin(); }
    const_iterator cbegin() const { return list_.cbegin(); }
    iterator end() { return list_.end(); }
    const_iterator end() const { return list_.end(); }
    const_iterator cend() const { return list_.cend(); }

    size_t size() const { return list_.size(); }
    value_ptr_type &operator[](std::size_t index) { return list_[index]; }

    template <bencodeDerived T, typename... Args>
    void push_back(Args &&...args)
    {
        list_.emplace_back(std::make_unique<T>(std::forward<Args>(args)...));
    }
    void pop_back()
    {
        if (!list_.empty())
            list_.pop_back();
    }

    std::string encode() const override
    {
        std::string enc_str(1, list_token);
        for (const value_ptr_type &ptr_ : list_)
            if (ptr_)
                enc_str += (ptr_->encode());
        enc_str += end_token;
```

```cpp
class bencode_string : public bencode_base
{
public:
    using string_type = std::string;
    using iterator = string_type::iterator;
    using const_iterator = string_type::const_iterator;
    using CharT = char;

    bencode_string() = default;
    bencode_string(string_type str) : str_{str} {}
    bencode_string(const CharT *cstr) : str_{cstr} {}

    iterator begin() { return str_.begin(); }
    const_iterator begin() const { return str_.begin(); }
    const_iterator cbegin() const { return str_.cbegin(); }
    iterator end() { return str_.end(); }
    const_iterator end() const { return str_.end(); }
    const_iterator cend() const { return str_.cend(); }

    bencode_string &operator=(string_type str)
    {
        str_ = str;
        return *this;
    }
    bencode_string &operator=(const CharT *cstr)
    {
        str_ = cstr;
        return *this;
    }
    size_t size() const { return str_.length(); }
    string_type get() const { return str_; }
    std::string encode() const override
    {
        return std::to_string(str_.length()) + std::string(1, delimiter_token) + str_;
    }
    void encode_n_dump(std::ostream &out) const override
    {
        out << str_.length() << delimiter_token << str_;
    }
    void decode(const std::string &in, std::string::const_iterator &start) override
    {
        size_t len = 0; // Assuming it starts with length
        while (start != in.end() && *start != delimiter_token)
```

```cpp
class bencode_dict : public bencode_base
{
private:
    // custom comparator
    template <typename T_>
    struct lexicographical_compare
    {
        bool operator()(const T_ &lhs, const T_ &rhs) const
        {
            return std::lexicographical_compare(lhs.cbegin(), lhs.cend(), rhs.cbegin(), rhs.cend());
        }
    };

public:
    using key_type = bencode_string;
    using mapped_type = std::unique_ptr<bencode_base>;
    using value_type = std::pair<const key_type, mapped_type>;
    using comparator_type = lexicographical_compare<key_type>;
    using container_type = std::map<key_type, mapped_type, comparator_type>;
    using iterator = container_type::iterator;
    using const_iterator = container_type::const_iterator;

    bencode_dict() = default;

    iterator begin() { return dict_.begin(); }
    const_iterator begin() const { return dict_.begin(); }
    const_iterator cbegin() const { return dict_.cbegin(); }
    iterator end() { return dict_.end(); }
    const_iterator end() const { return dict_.end(); }
```

```cpp
    std::string encode() const override
    {
        std::string enc_str(1, dict_token);
        for (const auto &[key_, ptr_] : dict_)
        {
            if (ptr_)
            {
                enc_str += key_.encode();
                enc_str += ptr_->encode();
            }
        }
        enc_str += end_token;
        return enc_str;
    }
    void encode_n_dump(std::ostream &out) const override
    {
        out << dict_token;
        for (const auto &[key_, ptr_] : dict_)
        {
            if (ptr_)
            {
                key_.encode_n_dump(out);
                ptr_->encode_n_dump(out);
            }
        }
        out << end_token;
    }
    void decode(const std::string &in, std::string::const_iterator &start) override
    {
        dict_.clear();
        start++; // assuming *start = 'd'
        while (start != in.end() && *start != end_token)
        {
            key_type key_;
            key_.decode(in, start);
            dict_[key_] = make_value(in, start);
        }
        start++; // assuming *start = 'e'
    }
    std::string get_as_str() const override
    {
        return this->encode();
    }
```

Tracker

UDP Tracker

The code initialises UDP server, listening for incoming client messages on a specified port, handling connection and announcement requests, and logging client messages in hexadecimal format.

Header files socket_util.h is used for implementation of Tracker

```cpp
#include "socket_util.h"
#include "peer_manager.h"
using namespace std;

void startUDPServer()
{
    srand(time(0));
    cout << "Server" << endl;
    int listenSocket=createUDPIPv4Socket();
    if (listenSocket < 0) {
        cout << "Socket creation failed" << endl;
        return ;
    }
    sockaddr_in trackeraddr=createIPv4Address("0.0.0.0",PORT);

    if (bind(listenSocket, reinterpret_cast<sockaddr*>(&trackeraddr), sizeof(trackeraddr)) < 0) {
        cout << "Bind error" << endl;
        close(listenSocket);
        return;
    }
        char buffer[4096];
        sockaddr_in clientAddr;
        socklen_t addrLen=sizeof(clientAddr);
        while (true) {
        int bytesRcvd = recvfrom(listenSocket, buffer, sizeof(buffer), 0,(sockaddr*)&clientAddr,&addrLen)

        cout << "Message from client: " << endl;
        for(size_t i=0; i<bytesRcvd;i++)
        {
            printf("%02x",(unsigned char)buffer[i]);
        }
```

BEP-15

The following reference has been used for the format of connect request and response

Connect

Before announcing or scraping, you have to obtain a connection ID.

1. Choose a random transaction ID.
2. Fill the connect request structure.
3. Send the packet.

connect request:

Offset	Size	Name	Value
0	64-bit integer	protocol_id	0x41727101980 // magic constant
8	32-bit integer	action	0 // connect
12	32-bit integer	transaction_id	
16			

1. Receive the packet.
2. Check whether the packet is at least 16 bytes.
3. Check whether the transaction ID is equal to the one you chose.
4. Check whether the action is connect.
5. Store the connection ID for future use.

connect response:

Offset	Size	Name	Value
0	32-bit integer	action	0 // connect
4	32-bit integer	transaction_id	
8	64-bit integer	connection_id	
16			

Announce request

This is Bittorent Extension proposal and is made for UDP tracker.It can also be updated to IPv6

Announce

1. Choose a random transaction ID.
2. Fill the announce request structure.
3. Send the packet.

IPv4 announce request:

Offset	Size	Name	Value
0	64-bit integer	connection_id	
8	32-bit integer	action	1 // announce
12	32-bit integer	transaction_id	
16	20-byte string	info_hash	
36	20-byte string	peer_id	
56	64-bit integer	downloaded	
64	64-bit integer	left	
72	64-bit integer	uploaded	
80	32-bit integer	event	0 // 0: none; 1: completed; 2: started; 3: stopped
84	32-bit integer	IP address	0 // default
88	32-bit integer	key	
92	32-bit integer	num_want	-1 // default
96	16-bit integer	port	
98			

1. Receive the packet.
2. Check whether the packet is at least 20 bytes.
3. Check whether the transaction ID is equal to the one you chose.
4. Check whether the action is announce.
5. Do not announce again until interval seconds have passed or an event has occurred.

Do note that most trackers will only honor the `IP address` field under limited circumstances.

Retrieving peers from database

sqlite3 library of C++ is utilised to achieve this functionality

```cpp
std::vector<Peer> retrievePeersFromDatabase(const std::string &infoHash)
{
    std::vector<Peer> peers;

    const char *selectSQL = "SELECT peer_id, ip, port FROM peers WHERE info_hash = ?";
    sqlite3_stmt *stmt;

    if (sqlite3_prepare_v2(db, selectSQL, -1, &stmt, 0) != SQLITE_OK)
    {
        std::cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << std::endl;
        return peers;
    }

    sqlite3_bind_text(stmt, 1, infoHash.c_str(), -1, SQLITE_STATIC);

    while (sqlite3_step(stmt) == SQLITE_ROW)
    {
        Peer peer;
        peer.peer_id = reinterpret_cast<const char *>(sqlite3_column_text(stmt, 0));
        peer.ip = reinterpret_cast<const char *>(sqlite3_column_text(stmt, 1));
        peer.port = sqlite3_column_int(stmt, 2);
        peers.push_back(peer);
    }

    sqlite3_finalize(stmt);
    return peers;
}
```

Announce Request

The peer generates an announce request and registers its presence with the tracker

```cpp
#include <iostream>
#include <cstring>
#include <vector>
#include <arpa/inet.h> // For inet_addr

// Define htonll if not available
#ifndef htonll
uint64_t htonll(uint64_t value) {···
#endif
#include <cstdint>     // For fixed-width integer types like uint32_t, uint16_t

// Function to create an announce request
std::vector<uint8_t> createAnnounceRequest() {
    std::vector<uint8_t> request;

    // 8-byte Connection ID (for example, using a known constant)
    uint64_t connection_id = htonll(0x41727101980); // For a real tracker
    request.insert(request.end(), reinterpret_cast<uint8_t*>(&connection_id),
                   reinterpret_cast<uint8_t*>(&connection_id) + sizeof(connection_id));

    // 4-byte Action (1 for announce)
    uint32_t action = htonl(1);
    request.insert(request.end(), reinterpret_cast<uint8_t*>(&action),
                   reinterpret_cast<uint8_t*>(&action) + sizeof(action));

    // 4-byte Transaction ID (example value)
    uint32_t transaction_id = htonl(123456); // Arbitrary transaction ID
    request.insert(request.end(), reinterpret_cast<uint8_t*>(&transaction_id),
                   reinterpret_cast<uint8_t*>(&transaction_id) + sizeof(transaction_id));
```

socket_util.cpp

The file contains important functions that deal with creation of socket and IPV4 addresses

```cpp
#include "socket_util.h"

int createUDPIPv4Socket()
{
    int s=socket(AF_INET,SOCK_DGRAM,0);
    return s;
}
int createTCPIPv4Socket()
{
    int s=socket(AF_INET,SOCK_STREAM,0);
    return s;
}
sockaddr_in createIPv4Address(const char* ip, int port) {
    sockaddr_in address;
    address.sin_family = AF_INET;
    address.sin_port = htons(port);

    if (strlen(ip) == 0) {
        address.sin_addr.s_addr = INADDR_ANY;
    } else {
        inet_pton(AF_INET, ip, &address.sin_addr);
    }

    return address;
}

uint64_t ntohll(uint64_t value)···

uint64_t htonll(uint64_t value) {···
```

db_manager.cpp

The file is responsible for maintaining peers' information in the database using functions such as store peer in database. It can also count number of seeders and leechers.

```cpp
std::vector<Peer> retrievePeersFromDatabase(const std::string& infoHash) {···

void initializeDatabase() {
    if (sqlite3_open("tracker.db", &db)) {
        std::cerr << "Can't open database: " << sqlite3_errmsg(db) << std::endl;
        exit(1);
    }

    const char* createTableSQL = R"(
        CREATE TABLE IF NOT EXISTS peers (
            info_hash TEXT NOT NULL,
            peer_id TEXT NOT NULL,
            ip TEXT NOT NULL,
            port INTEGER NOT NULL,
            uploaded INTEGER,
            downloaded INTEGER,
            left INTEGER
        );
    )";

    char* errMsg = 0;
    if (sqlite3_exec(db, createTableSQL, NULL, 0, &errMsg)) {
        std::cerr << "SQL error: " << errMsg << std::endl;
        sqlite3_free(errMsg);
    }
}

void storePeerInDatabase(const std::string& infoHash, const std::string& peerId, const std::string& ip, int
port, int uploaded, int downloaded, int left) {···

void countLeechersAndSeeders(const std::string& infoHash, int& leechers, int& seeders) {···
```

peer_manager.cpp

Important functions:

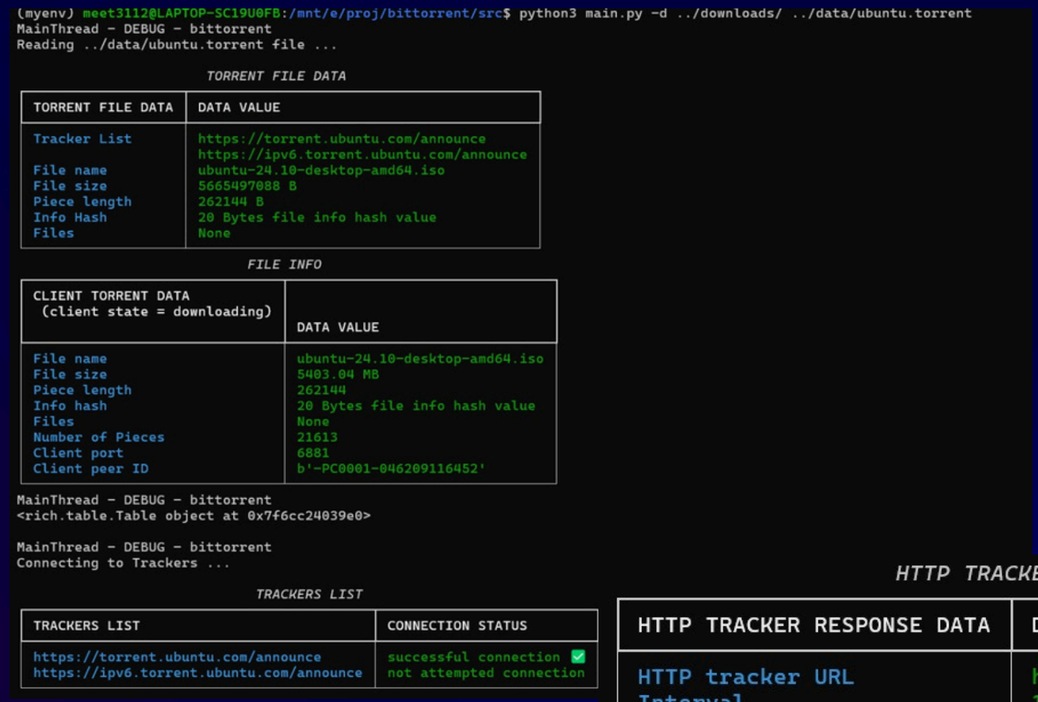
- generateConnectionID()
- handleConnectRequest()
- processAnnounceRequest()

```cpp
#include "peer_manager.h"
#include "db_manager.h"

using namespace std;
set<uint64_t> valid_connection_ids;

uint64_t generateConnectionID()···
void handleConnectRequest(char *buffer, int bytesRcvd, int listenSocket, sockaddr_in &clientAddr,
addrLen)···
void processAnnounceRequest(char *buffer, int length, int sockfd, sockaddr_in &clientAddr)
{
    try
    {
        // Check if buffer length matches the expected size
        if (length < sizeof(AnnounceRequest))
        {
            throw std::runtime_error("Invalid announce request length");
        }

        AnnounceRequest req;
        memcpy(&req, buffer, sizeof(AnnounceRequest));
        // sample Annnouncement:<8-byte Connection ID><4-byte Action (1 for announce)><4-byte Tran
        ID><20-byte Info_hash><20-byte Peer ID><8-byte Downloaded><8-byte Left><8-byte Uploaded><4
        Event><4-byte IP address><4-byte Key><4-byte Num Want><2-byte Port>
        //  Convert network byte order to host byte order where necessary
        req.connection_id = ntohl(req.connection_id);

        if (valid_connection_ids.find(req.connection_id) == valid_connection_ids.end())
```

Application Snapshot

Peer file download interface

The screenshot shows interface of how Torrent file, trackers list and other data is provided to the client



```
(myenv) meet3112@LAPTOP-SC19U0FB:/mnt/e/proj/bittorrent/src$ python3 main.py -d ../downloads/ ../data/ubuntu.torrent
MainThread - DEBUG - bittorrent
Reading ../data/ubuntu.torrent file ...
```

TORRENT FILE DATA

TORRENT FILE DATA	DATA VALUE
Tracker List	https://torrent.ubuntu.com/announce https://ipv6.torrent.ubuntu.com/announce
File name	ubuntu-24.10-desktop-amd64.iso
File size	5665497088 B
Piece length	262144 B
Info Hash	20 Bytes file info hash value
Files	None

FILE INFO

CLIENT TORRENT DATA (client state = downloading)	DATA VALUE
File name	ubuntu-24.10-desktop-amd64.iso
File size	5403.04 MB
Piece length	262144
Info hash	20 Bytes file info hash value
Files	None
Number of Pieces	21613
Client port	6881
Client peer ID	b'-PC0001-046209116452'

```
MainThread - DEBUG - bittorrent
<rich.table.Table object at 0x7f6cc24039e0>

MainThread - DEBUG - bittorrent
Connecting to Trackers ...
```

TRACKERS LIST

TRACKERS LIST	CONNECTION STATUS
https://torrent.ubuntu.com/announce	successful connection ✅
https://ipv6.torrent.ubuntu.com/announce	not attempted connection

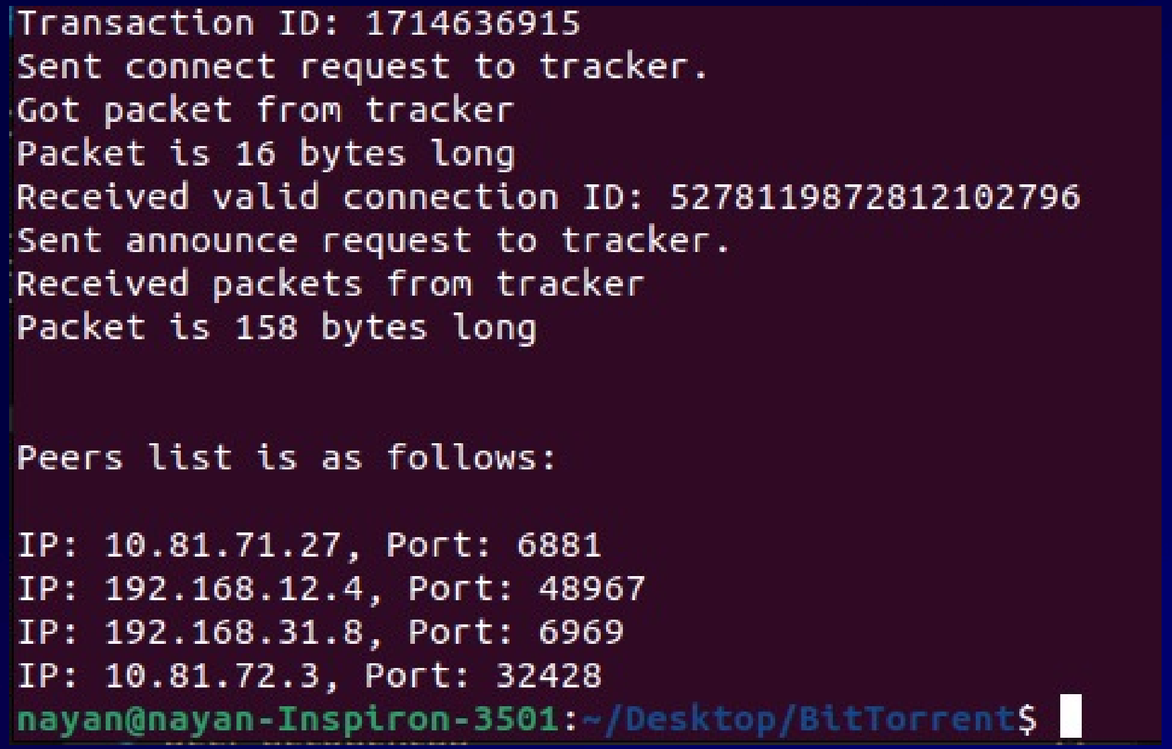
HTTP TRACKER RESPONSE DATA

HTTP TRACKER RESPONSE DATA	DATA VALUE
HTTP tracker URL	https://torrent.ubuntu.com/announce
Interval	1800
Number of leechers	82
Number of seeders	853
Peers in swarm	(185.125.190.59 : 6894) ... 0 more peers

```
MainThread - DEBUG - bittorrent
<rich.table.Table object at 0x7f6cc254fc80>

MainThread - DEBUG - bittorrent
Initializing the swarm of peers ...

MainThread - DEBUG - bittorrent
Initializing the file handler for peers in swarm ...

MainThread - DEBUG - bittorrent
Client started downloading (check torrent statistics) ...
```

Tester’s Terminal

The following is the screenshot of tester’s terminal output and the details of the requests it sent and responses it recieved.



Tracker’s interface

The screenshot below shows the handling of requests and responses by the tracker

```
haaziq@Haaziq-Asus:~/bittorrent-tracker$ g++ main.cpp db_manager.cpp peer_manager.cpp udp_server.cpp socket_util.cpp -l sqlite3 -o bittorrent_tracker
haaziq@Haaziq-Asus:~/bittorrent-tracker$ ./bittorrent_tracker
Server
Message from client:
80191027170400000000000073483366
Client connected
Received a valid connect request
Generated connnection id: 5278119872812102796
Sent connect response to client
Message from client:
8c34e5440fa63f490000000173483366117e3a6665e8ff1b157e5ec37823578adb8a712b304d6a526f7852334b757a5865636a4f0000000000000000000000000000000000001e54a0000000000000000000000000000000643c9869000000001ae1000000000000
Client connected
Announce request received
8c 34 e5 44 0f a6 3f 49 00 00 00 01 73 48 33 66 11 7e 3a 66 65 e8 ff 1b 15 7e 5e c3 78 23 57 8a db 8a 71 2b 30 4d 6a 52 6f 78 52 33 4b 75 7a 58 65 63 6a 4f 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 01 e5 4a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 64 3c 98 69 00 00 00 00 1a e1 00 00 00 00 00 00
ip address of peer:10.81.34.228
response sent by tracker:
1 1714636915 0 124234 0 0 10.81.34.228 1681692777 0 6881
```

ACKNOWLEDGMENT

We would like to express our immense gratitude to all the individuals who have helped and supported us throughout the project. We are thankful to our course instructor, Prof. Sandeep Kumar for his support from initial advice, and encouragement, till the completion of this project. We would also like to thank all the Teaching Assistants (TAs) who were always present in our practical sessions for assistance. A special acknowledgement goes to our classmates who helped us by exchanging interesting ideas and sharing their valuable experiences.

Thank You

Thank You