National Institute of Technology Silchar

COMPUTER NETWORKS PROJECT

Analysis and Implementation of bitTorrent Protocol

TUSHAR SINGH

Prof. Dr Anupam Biswas

Computer Science and Engineering Department

December 2020

ABSTRACT

Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes.

These are widely used for file sharing, and one such is the BitTorrent Protocol.

Each BitTorrent client is an agent that interacts with other agents and reacts autonomously, following the same decision algorithms as a real client. Their goals are downloading contents in the shortest time possible and sharing them with other agents.

CONTENTS

INTRODUCTION

The main purpose of this project is to analyse and implementations of BitTorrent protocol. Ever since the development and subsequent release of bit-torrent Protocol by its inventor Bram Cohen in July, 2001 , it has remained a very popular file sharing protocol. In November 2004, BitTorrent accounted for a huge 35 percent of all the traffic on the Internet and in 2006 the BitTorrent protocol has risen to over 60 percent of all Internet traffic according to British Web analysis firm CacheLogic and 78 % of all the global p2p traffic . As per a recent article in Jan., 2012, it is claimed that BitTorrent is accounting for an unbelievable 150 million monthly users . Furthermore, it is estimated that BitTorrent, at any given point of time, has on average more active users than YouTube and Facebook combined . One key aspect of performance in BitTorrent is the download rate that is achieved by participating peers. In this project we try to analyse BitTorrent protocol in bit details.

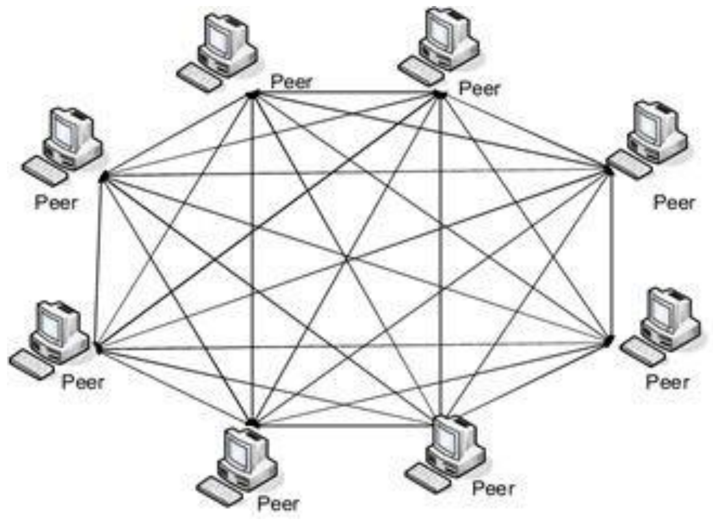
In this project we will learn about p2p networking. We will analyze the bittorrent protocols and learn about them too.
In the end we will go through the implementation part of the project in which we will try to mimic a bittorrent client.

p2p NETWORKING

A peer-to-peer network is a technology that allows you to connect two or more computers to one system. This connection allows you to easily share data without having to use a separate server for file-sharing. Each end-computer that connects to this network becomes a 'peer' and is allowed to receive or send files to other computers in its network. This enables you to work collaboratively to perform certain tasks that need group attention, and it also allows you to provide services to another peer.

p2p Architecture

The peer to peer computing architecture contains nodes that are equal participants in data sharing. All the tasks are equally divided between all the nodes. The nodes interact with each other as required as shared resources.



Characteristics of Peer to Peer Computing

The different characteristics of peer to peer networks are as follows −

- Peer to peer networks are usually formed by groups of a dozen or less computers. These computers all store their data using individual security but also share data with all the other nodes.

- The nodes in peer to peer networks both use resources and provide resources. So, if the nodes increase, then the resource sharing capacity of the peer to peer network increases. This is different from client server networks where the server gets overwhelmed if the nodes increase.
- Since nodes in peer to peer networks act as both clients and servers, it is difficult to provide adequate security for the nodes. This can lead to denial of service attacks.

Advantages & Disadvantages of Peer to Peer Computing

Some advantages of peer to peer computing are as follows −

- Each computer in the peer to peer network manages itself. So, the network is quite easy to set up and maintain.
- In the client server network, the server handles all the requests of the clients. This provision is not required in peer to peer computing and the cost of the server is saved.
- It is easy to scale the peer to peer network and add more nodes. This only increases the data sharing capacity of the system.
- None of the nodes in the peer to peer network are dependent on the others for their functioning.

Some disadvantages of peer to peer computing are as follows −

- It is difficult to backup the data as it is stored in different computer systems and there is no central server.
- It is difficult to provide overall security in the peer to peer network as each system is independent and contains its own data.

INTRODUCTION TO BIT-TORRENT

What is BitTorrent ?

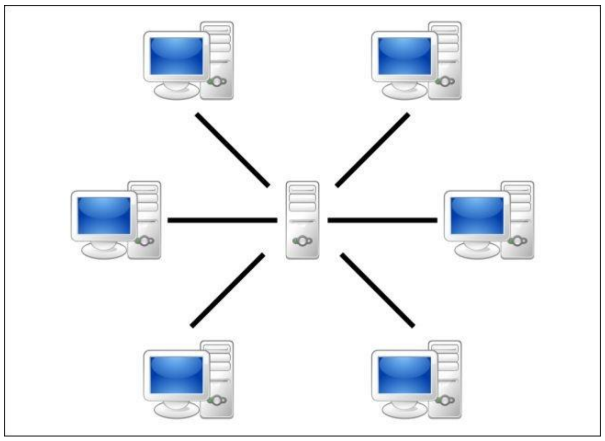
BitTorrent is a peer-to-peer file sharing protocol used to distribute large amounts of data. BitTorrent is one of the most common protocols for transferring large files. Its main usage is for the transfer of large sized files. A user can obtain multiple files simultaneously without any considerable loss of the transfer rate.

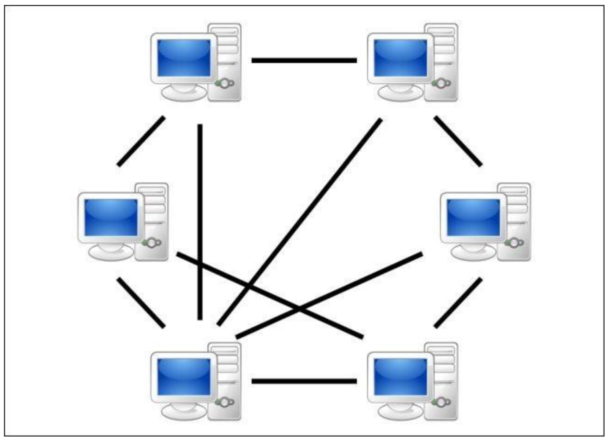
BitTorrent is based on the notion of a torrent, which is a smallish file that contains metadata about a host, the tracker, that coordinates the file distribution and files that are shared A peer that wishes to make data available must first find a tracker for the data, create a torrent, and then distribute the torrent file.

Other peers can then use information contained in the torrent file to assist each other in downloading the file The download is coordinated by the tracker. In BitTorrent terminology, peers that provide a complete file with all of its pieces are called seeders

How BitTorrent Works?

When you download a web page, your computer connects to the web server and downloads the data directly from that server. Each computer that downloads the data downloads it from the web page’s central server. This is how much of the traffic on the web works.





BitTorrent is a peer-to-peer protocol, which means that the computers in a BitTorrent “swarm” (a group of computers downloading and uploading the same torrent) transfer data between each other without the need for a central server.

Traditionally, a computer joins a BitTorrent swarm by loading a .torrent file into a BitTorrent client. The BitTorrent client contacts a “tracker” specified in the .torrent file. The tracker is a special server that keeps track of the connected computers. The tracker shares their IP addresses with other BitTorrent clients in the swarm, allowing them to connect to each other.

Once connected, a BitTorrent client downloads bits of the files in the torrent in small pieces, downloading all the data it can get. Once the BitTorrent client has some data, it can then begin to upload that data to other BitTorrent clients in the swarm. In this way, everyone downloading a torrent is also uploading the same torrent. This speeds up everyone’s download speed. If 10,000 people are downloading the same file, it doesn’t put a lot of stress on a central server. Instead, each downloader contributes upload bandwidth to other downloaders, ensuring the torrent stays fast.

Importantly, BitTorrent clients never actually download files from the tracker itself. The tracker participates in the torrent only by keeping track of the BitTorrent clients connected to the swarm, not actually by downloading or uploading data.
Users downloading from a BitTorrent swarm are commonly referred to as “leechers” or “peers”. Users that remain connected to a BitTorrent swarm even after they’ve downloaded the complete file, contributing more of their upload bandwidth so other people can continue to download the file, are referred to as “seeders”. For a torrent to be downloadable, one seeder – who has a complete copy of all the files in the torrent – must initially join the swarm so other users can download the data. If a torrent has no seeders, it won’t be possible to download – no connected user has the complete file.
BitTorrent clients reward other clients who upload, preferring to send data to clients who contribute more upload bandwidth rather than sending data to clients who upload at a very slow speed. This speeds up download times for the swarm as a whole and rewards users who contribute more upload bandwidth.

BIT-TORRENT ARCHITECTURE

BitTorrent distributes a file by partitioning it into ‘pieces’ and distributing the pieces amongst its peers.
The architecture normally consists of the following entities:

- a ‘tracker’
- a static metainfo file (a torrent file)
- an original downloader (seeder)
- the end user downloader (leecher)

Tracker

A BitTorrent tracker is server software that centrally coordinates the transfer of files among users, the tracker does not contain a copy of the file and only helps peers discover each other.
The tracker and the client exchange information using a simple protocol on top of HTTP. The clients inform the tracker regarding the file they want to download, their IP and port and the tracker respond with a list of peers downloading the same file and their contact information. This list of peers that all share the same torrent represents a ‘swarm’. The tracker is necessary for peers to find each other and transfer data, because of the presence of this central entity, BitTorrent protocol is considered as a Hybrid P2P implementation.

Metainfo file

The metainfo file is also called as a torrent file and has a .torrent extension.
This file mainly contains encoded information regarding the url of the tracker, name of the file, and hashes of the pieces of the file for verifying downloaded pieces of the file.
These torrent files are generally created using client software. A list of trackers and the original file are required to create this torrent file. Once the file is created it can be shared using the regular methods such as email, file sharing websites etc.

Seeders

The original downloader is a peer with the whole file. Also known as seeder. The seeder must keep uploading the file until a complete copy has been distributed among the downloaders. As long as there is a complete copy collectively present among the peers the download will continue for all.
The developers providing the linux iso image who have the full file will be called as seeders.

Leechers
The peers without a complete copy of the file are known as leechers. Leechers will get the list of peers from the tracker which have the pieces that the leecher requires. The leecher then downloads the required piece from one of those peers. A leecher can also distribute the pieces that it has completed downloading even before it completes downloading the whole file. Once a Leecher has all the pieces it is called a seeder. As a leecher receives the 'pieces' it validates them against the hashes present in the meta info file.
Any user downloading the file through BitTorrent will be called as a leecher, once they have a full file they can be called as seeders.

BitTorrent uses a piece selection algorithm to decide which piece to download with the goal of achieving maximum piece replication.

Piece Selection Algorithm

Random First Piece
When downloading first begins, as the peer has nothing to upload, a piece is selected at random to get the download started. Random pieces are then chosen until the first piece is completed and checked. Once this happens, the 'rarest first' strategy begins.

Rarest First
When a peer selects which piece to download next, the rarest piece will be chosen from the current swarm, i.e. the piece held by the lowest number of peers. This means that the most common pieces are left until later, and focus goes to replication of rarer pieces.
At the beginning of a torrent, there will be only one seed with the complete file. There would be a possible bottleneck if multiple downloaders were trying to access the same piece. rarest first avoids this because different peers have different pieces. As more peers connect, rarest first will then load off of the tracker, as peers begin to download from one another.
Eventually the original seed will disappear from a torrent.If the original seed goes before at least one other peer has the complete file, then no one will reach completion, unless a seed re-connects.

When a download nears completion, and waiting for a piece from a peer with slow transfer rates, completion may be delayed. To prevent this, the remaining sub-pieces are requested from all peers in the current swarm.

BIT-TORRENT CLIENT

A BitTorrent client is an executable program which implements the BitTorrent protocol. It runs together with the operating system on a user's machine, and handles interactions with the tracker and peers. The client sits on the operating system and is responsible for controlling the reading / writing of files, opening sockets etc.

Understanding .torrent file

A .torrent file describes the contents of a torrented file and information for connecting to a tracker. It's all we need in order to kickstart the process of downloading a torrent. A simple .torrent file looks like this:

```
d8:announce41:http://bttracker.debian.org:6969/announce7:comment35:"Debian CD from
cdimage.debian.org"13:creation
datei1573903810e9:httpseedsl145:https://cdimage.debian.org/cdimage/release/10.2.0//
srv/cdbuilder.debian.org/dst/deb-cd/weekly-builds/amd64/iso-cd/debian-10.2.0-amd64-
netinst.iso145:https://cdimage.debian.org/cdimage/archive/10.2.0//srv/cdbuilder.deb
ian.org/dst/deb-cd/weekly-builds/amd64/iso-cd/debian-10.2.0-amd64-netinst.isoe4:inf
od6:lengthi351272960e4:name31:debian-10.2.0-amd64-netinst.iso12:piece
lengthi262144e6:pieces26800:88888PS8^88 (binary blob of the hashes of each
piece)ee
```

That is encoded in a format called **Bencode**, and we'll need to decode it.
Which is just this(below). It contains all the necessary details for the file to be downloaded.

```
d
  8:announce
    41:http://bttracker.debian.org:6969/announce
  7:comment
    35:"Debian CD from cdimage.debian.org"
  13:creation date
    i1573903810e
  4:info
    d
      6:length
        i351272960e
      4:name
        31:debian-10.2.0-amd64-netinst.iso
      12:piece length
        i262144e
      6:pieces
        26800:88888PS8^88(binary blob of the hashes of each piece)
    e
e
```

Getting peers via the tracker

Udp is often a good choice for trackers because they send small messages, and we use tcp for when we actually transfer files between peers because those files tend to be larger and must arrive intact.
The tracker is an HTTP(S) service that holds information about the torrent and peers.

UDP tracker protocol and message format

1. Send a connect request
2. Get the connect response and extract the connection id
3. Use the connection id to send an announce request - this is where we tell the tracker which files we're interested in
4. Get the announce response and extract the peers list

Connect message

```
function buildConnReq() {
  const buf = Buffer.allocUnsafe(16);
  buf.writeUInt32BE(0x417, 0);
  buf.writeUInt32BE(0x27101980, 4);
  buf.writeUInt32BE(0, 8);
  crypto.randomBytes(4).copy(buf, 12);

  return buf;
}
```

Announce message

Now that we have information about the file and its tracker, let's talk to the tracker to announce our presence as a peer and to retrieve a list of other peers. We just need to make a GET request to the announce URL supplied in the .torrent file, with a few query parameters:

```
function buildAnnounceReq(connId, torrent, port = 6881) {
  const buf = Buffer.allocUnsafe(98);

  // connection id
  connId.copy(buf, 0);
  buf.writeUInt32BE(1, 8);
  crypto.randomBytes(4).copy(buf, 12);
  // info hash
  torrentParser.infoHash(torrent).copy(buf, 16);
  // peerId
  util.genId().copy(buf, 36);
  // downloaded
  Buffer.alloc(8).copy(buf, 56);
  // left
  torrentParser.size(torrent).copy(buf, 64);
  // uploaded
  Buffer.alloc(8).copy(buf, 72);
  // event
  buf.writeUInt32BE(0, 80);
  // ip address
  buf.writeUInt32BE(0, 80);
  // key
  crypto.randomBytes(4).copy(buf, 88);
  // num want
  .. ..
  return buf;
}
```

Important Note :

- **info_hash**: Identifies the file we're trying to download. It's the infohash we calculated earlier from the bencoded info dict. The tracker will use this to figure out which peers to show us.
- **peer_id**: A 20 byte name to identify ourselves to trackers and peers.

Downloading from peers

1. First you'll want to create a tcp connection with all the peers in your list.The more peers you can get connected to the faster you can download your files.
2. After exchanging some messages with the peer as setup, you should start requesting pieces of the files you want. As we'll see shortly, a torrent's shared files are broken up into pieces so that you can download different parts of the files from different peers simultaneously.
3. Most likely there will be more pieces than peers, so once we're done receiving a piece from a peer we'll want to request the next piece we need from them. Ideally you want all the connections to be requesting different and new pieces so you'll need to keep track of which pieces you already have and which ones you still need.
4. Finally, when you receive the pieces they'll be stored in memory so you'll need to write the data to your hard disk. Hopefully at this point you'll be done!

Tcp connect to peers

```
function downloadFromPeer(peer, torrent, pieces, file) {
  const socket = new net.Socket();

  socket.on('error', console.log);
  socket.connect(peer.port, peer.ip, () => {
    console.log('connected to peer !', peer);
    socket.write(message.buildHandshake(torrent));
  });
```

Handshaking

Once you have a connection to your peer(s).The first message you send should be a Handshake and it's made up of five parts:

1. The length of the protocol identifier, which is always 19 (0x13 in hex)
2. The protocol identifier, called the pstr which is always BitTorrent protocol
3. Eight reserved bytes, all set to 0. We'd flip some of them to 1 to indicate that we support certain extensions. But we don't, so we'll keep them at 0.
4. The infohash that we calculated earlier to identify which file we want
5. The Peer ID that we made up to identify ourselves

```
module.exports.buildHandshake = torrent => {
    const buf = Buffer.allocUnsafe(68);
    // pstrlen
    buf.writeUInt8(19, 0);
    // pstr
    buf.write('BitTorrent protocol', 1);
    // reserved
    buf.writeUInt32BE(0, 20);
    buf.writeUInt32BE(0, 24);
    // info hash
    torrentParser.infoHash(torrent).copy(buf, 28);
    // peer id
    util.genId().copy(buf, 48);

    return buf;
  };
```

After you establish the handshake your peers should tell you which pieces they have.

Message Passing

Now we get to the meat of the Bittorrent protocol. Types of messages that bittorrent supports: keep-alive, choke, unchoke, interested, not-interested, have, bitfield, request, piece, cancel, and port.

```
  module.exports.buildChoke = () => {
    const buf = Buffer.allocUnsafe(5);
    // length
    buf.writeUInt32BE(1, 0);
    // id
    buf.writeUInt8(0, 4);
    return buf;
  };
  module.exports.buildUnchoke = () => {
    //
  };
  module.exports.buildInterested = () => {
    //
  };
  module.exports.buildUninterested = () => {
    //
  };
  module.exports.buildHave = payload => {
    //
  };
  module.exports.buildBitfield = bitfield => {
    //
  };
 .. so on...
```

Note :

Piece: Pieces are the different pieces of the file we are requesting .

Bitfield : It is a data structure that peers use to efficiently encode which pieces they are able to send us. A bitfield looks like a byte array, and to check which pieces they have, we just need to look at the positions of the bits set to 1.

Piece response handler

This is the final section before we can start downloading torrents! We just need to write out the pieceHandler which is for when we receive the actual bytes of the piece we requested.

The main things we want to do in this function are:

1. Add the piece to the list of received pieces.
2. Write the bytes to file.
3. Request the next needed piece. Or...
4. If there are no more pieces to download in the queue we close the socket connection.

```
function pieceHandler(socket, pieces, queue, torrent, file, pieceResp) {
  pieces.addReceived(pieceResp);

  const offset = pieceResp.index * torrent.info['piece length'] + pieceResp.begin;
  fs.write(file, pieceResp.block, 0, pieceResp.block.length, offset, () => {
    if (pieces.isDone()) {
      socket.end();
      console.log('DONE!');
      fs.close(file);
    } else {
      requestPiece(socket, pieces, queue);
    }
  });
}
```

```
function requestPiece(socket, pieces, queue) {
  if (queue.choked) return null;
  while (queue.length()) {
    const pieceBlock = queue.deque();
    if (pieces.needed(pieceBlock)) {
      socket.write(message.buildRequest(pieceBlock));
      pieces.addRequested(pieceBlock);
      break;
    }
  }
}
```

Running the client

We have used javaScript to build the bittorrent client. It's a simple client which can :

- parse torrent file
- use http or udp method according to torrent to get peers(ip, port) from tracker
- establish tcp connection from other peers and download single file

To run the program we need to first install some packages(bencode etc) which we have used here for that just run:

```
npm install
```

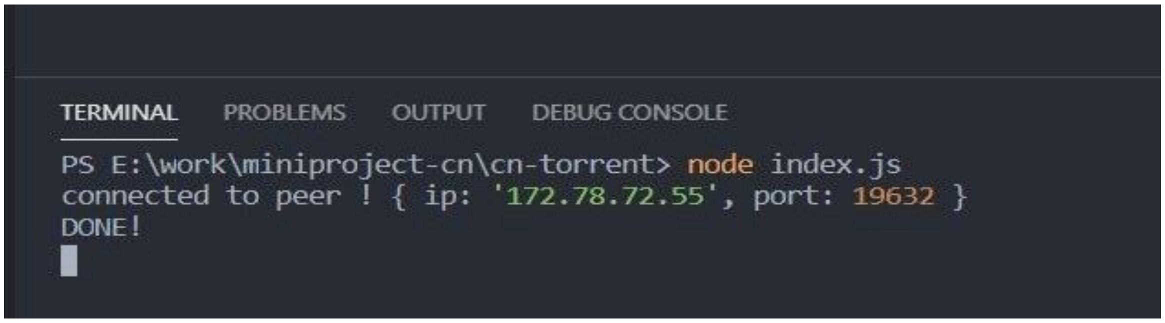
It will install all the necessary packages we have used in the project.
Now to download a torrent we need a .torrent file, and we can give the file location in the index.js file :

```
const torrentParser = require('./src/torrent-parser');
const handleTorrent = require('./src/handle-torrent');

const torrent = torrentParser.open('here_add_the_torrent_location'); //

handleTorrent(torrent, torrent.info.name);
```

Now to download the file just navigate to the root location of the project run the index.js :

```
node index.js
```

Output



```
PS E:\work\miniproject-cn\cn-torrent> node index.js
connected to peer ! { ip: '172.78.72.55', port: 19632 }
DONE!
```

CONCLUSION

This project presented an analysis and implementation of BitTorrent protocol.

First we reviewed the p2p Networking including architecture, characteristics and advantages and disadvantages . We noted that BitTorrent has been influenced by p2p file sharing applications. Thus we see the study of BitTorrent and its architecture including Trackers , metainfo file(Torrent file), Seeders, Leechers and then we describe BitTorrent client which implements the BitTorrent protocol and summarizes it by help of codes. Thus this mini project is very helpful for us and we learnt about p2p networking and BitTorrent and its implementations.

REFERENCES

[1]. Wikipedia - Bit-Torrents
https://en.wikipedia.org/wiki/BitTorrent

[2]. Official Site - Bit-Torrents Introductions
https://www.bittorrent.org/introduction.html

[3]. Official Site - Bit-Torrents Specifications
https://www.bittorrent.org/beps/bep_0003.html

[4]. Wikipedia - Peer to Peer
https://en.wikipedia.org/wiki/Peer-to-peer

[5]. Other references
https://blog.jse.li/posts/torrent/
https://www.howtogeek.com/141257/htg-explains-how-does-bittorrent-work
http://jonas.nitro.dk/bittorrent/bittorrent-rfc.html