

TJHSST Senior Research Project
End-to-End Publication Using the Bittorrent
P2P Filesharing Protocol
2006-2007

Andrew Wang

June 12, 2007

Abstract

Bittorrent is a promising peer-to-peer network that always allows for fast download speeds despite the number of peers downloading the file. Currently, there exist tools to make .torrent files, tools to "track" the peers downloading the file, tools to host .torrent files, and tools to initially upload the file. This project aims to unify this process by making an end-to-end software suite that simplifies the process of publishing a file on the Bittorrent network for download. The key to this will be automating and streamlining the process from the perspective of the user. It will involve a complete implementation of the Bittorrent protocol, including encoding torrent files, peer-to-tracker and peer-to-peer communication, and a greater understanding of the benefits and detriments of the Bittorrent protocol.

Keywords: Bittorrent, Peer-to-peer, Linux, Publishing, Download, Tracker

1 Introduction

1.1 Rationale

The scope of the project is broad, because it aims to be a complete solution to publishing files using Bittorrent. It will have to handle all aspects of the

Bittorrent protocol, from processing the file to make the .torrent metadata file to hosting and tracking the .torrent file and possibly a download client.

A new way of publishing using Bittorrent is important because the current system of publishing is much more complicated and is inaccessible to the normal computer user. Bittorrent has clear advantages over traditional methods of publishing via the Internet, such as HTTP or email, because it can handle a far larger number of users concurrently and thus allows for the publishing of far larger files, such as indie HD movies, podcasts, or other content that would otherwise be unfeasible because of bandwidth constraints [2] [3]. This system would also be superior to other forms of P2P though the use of a "everseed" that would keep the torrent from "dying" (a state where there are no peers with a complete copy of the file).

1.2 Purpose

This project started out as writing a better download client, but there is already a plethora of download clients available and finding an easy to use and mature Bittorrent development library proved difficult. It would also be difficult to surpass the quality and features of other download clients developing by myself, and users would be unlikely to switch their choice in download clients unless there was a very good reason. Thus, the project has been redirected toward streamlining the currently convoluted process of distributing a file using the Bittorrent protocol.

1.3 Expected Results

The research of this project involves the Bittorrent protocol. Given the specification of the protocol given on the Bittorrent website, I aim to implement the server side aspect. This means encoding and dencoding of .torrent metadata files, hosting of .torrent metadata files, and subsequent tracking and peer-to-tracker communication of download peers by the tracker. It will involve the subject areas of large scale networks, encoding and decoding algorithms, and peer-to-peer communication.

1.4 Type of Research

This project will be use-inspired basic research, because the underlying goal is to gain an understanding of the benefits and limitations of the Bittorrent

protocol. There are great practical implications for the end product of this research, but ultimately the project was started to gain an introduction to networking and peer-to-peer technology.

2 Background

Bittorrent is an up and coming filesharing protocol that has emerged in the wake of illegal services such as Kazaa, Napster, or Bearshare that have since been shutdown or forced to end their copyright violations [5]. Bittorrent is a much more legally feasible filesharing protocol than previous attempts, because there is no copyrighted content to be stored on centralized servers that can be subpoenaed or seized, and it has become extremely popular for independent movie makers and other people that need to distribute their legal content without buying an expensive server. A movie distribution method using Bittorrent is also being developed by major movie companies, as they too see the benefits of peer-to-peer technology.

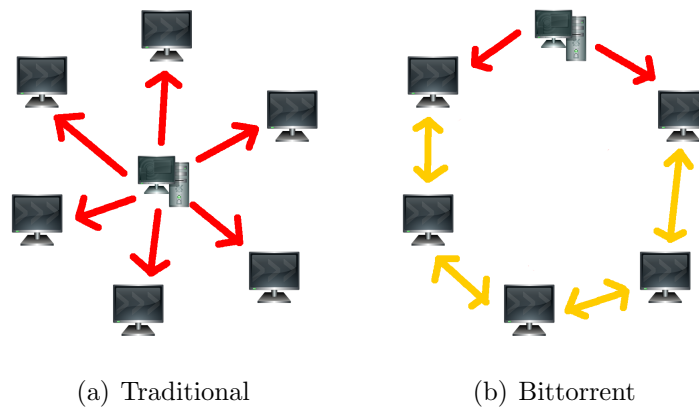


Figure 1: Comparison of traditional and Bittorrent methods of file distribution

Traditional file distribution (Figure 1(a)) uses a single centralized server to provide a copy of the file to each downloader. This means that a popular file can quickly exceed the hardware and network limitations of the centralized server, leading to slow download speeds, increased wait times, and general user dissatisfaction.

Bittorrent (Figure 1(b)) solves this problem by making use of peer-to-peer technology to scale available upload bandwidth according to demand. This allows for the distribution of large files without paying for increasingly expensive dedicated hosting and large internet connections.

The whitepaper written by the creator of Bittorrent, Bram Cohen, available on the official Bittorrent website is the most useful reference for this project [1]. Additionally, there is a semi-official Bittorrent Specification available at theory.org written by the major Bittorrent developers [4]. This document expands upon the official protocol specification in areas the official specification is lacking. These two documents give a total description of all aspects of Bittorrent, and will be the only necessary references throughout the project's extent.

3 Procedure and Methodology

3.1 Planning

The languages used in this project will be Python, for all parts of the project. Performance is not an issue because the processor and bandwidth requirements are low. A webserver of some sort will be needed to host the .torrent files, but this can be done with a third-party solution, or a basic server can be written if needed. The stages of this project can be split up into a number of clearly defined steps:

1. Study and implementation of encoding .torrent metadata files. These files are "bencoded," which is a translated form of dictionaries, lists, strings, and integers. This will also coincide with studying of the various kinds of metadata stored in .torrent files as well as an interface for creating these .torrent files.
2. Implementation of a .torrent metadata file generator, for distribution to clients.
3. Study and implementation of a Bittorrent tracker. The tracker must process the .torrent metadata value, store it into a database, and then handle processing "announce" and "scrape" requests from the clients that wish to download the file. It will also make use of the "bencoding" algorithm to send data from the tracker to the client.

A typical "announce" request from a client consists of status and unique identifying information. The tracker stores this information in a database, and the tracker then sends the client identifying information and a list of peers for the client to connect to for downloading and uploading purposes.

A typical "scrape" request asks the tracker for status information about a single or all torrents that the tracker is tracking. The difference between an "announce" and "scrape" request is determined by the URL used to query the tracker. The tracker will respond with information like the number of peers and seeds connected, and the number of downloads completed. This data will be fetched from the database.

4. Making a web interface that takes a file, prompts the user for the minimum amount of information regarding the file through the use of automation and intelligent defaults, make a .torrent metadata file for it, add it to the tracker, and put the file up for download via HTTP.
5. The final part of the project is the addition of the "everseed." The "everseed" is the initial and permanent uploader for the file that prevents the torrent from "dying" (a state where a complete copy of the file does not exist among the peers in the swarm, preventing the peers from ever completing downloading the file). This is a problem when a torrent has been around for a long time or is not that popular.

3.2 Testing and Analysis

3.2.1 .torrent File Testing

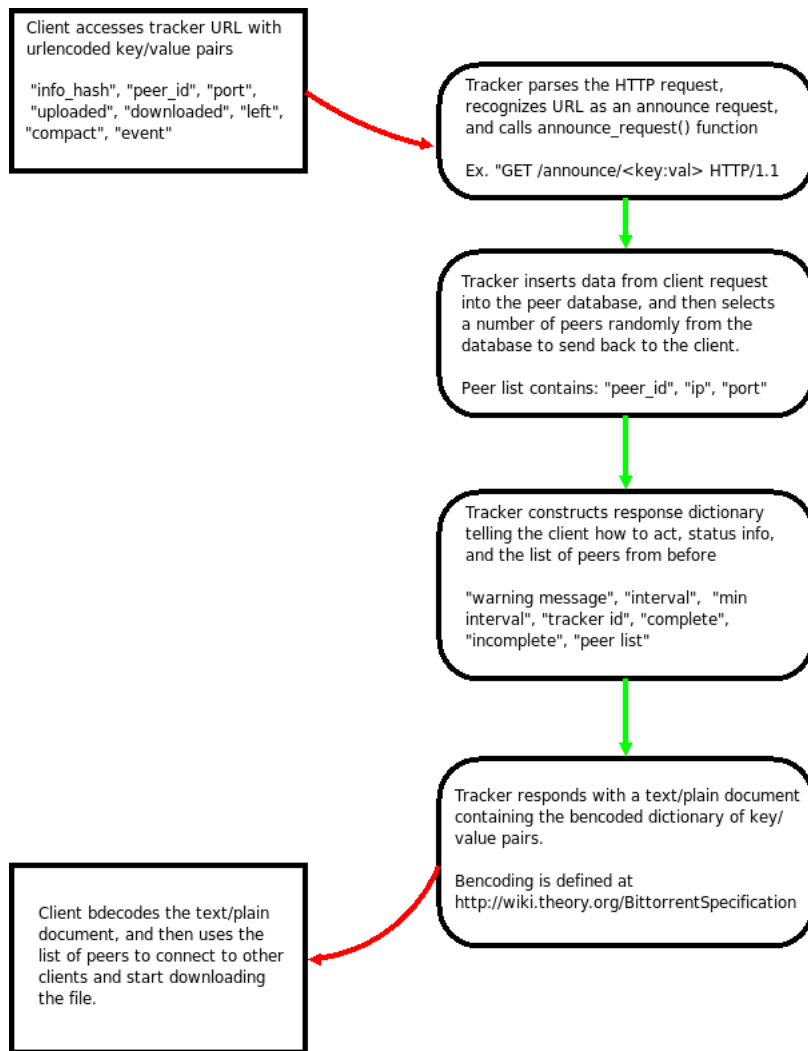
Testing of encoding .torrent files is done using examples on the Bittorrent website and others. The program will transparently handle errors because it will simply treat the invalid input as a string. This will result in an incorrect .torrent file though, so I will build in checking when I make the frontend for making .torrents. Performance is also not an issue for the encoding program because it takes minimal time even with the use of Python. The torrent files have also been verified using the official Bittorrent client as well as popular clients such as Azureus or Shad0w's.

```
[03:42:11] awang::hermit $ ./torrentfile.py
```

```
torrentfile.py 0.2
Enter a string: hello world!
Becoded string: 12:hello world!
Enter an integer: 12345
Becoded integer: i12345e
Enter a list: apple,orange,pear,grape
Becoded list: l5:apple6:orange4:pear5:grapee
```

```
Example bencoded dictionary
Dictionary: {'myname': ['andrew', 'wang'],
            'dozen': 12, 'apple': 'red', 'banana': 'yellow'}
Becoded: d6:myname16:andrew4:wange5:dozeni12e5:
apple3:red6:banana6:yellowe
```

3.2.2 Diagram of a Typical Announce Request



3.2.3 Analysis of a Typical Announce Request

The following is the step-by-step process through which the tracker handles an announce request from a client. The client passes data to the tracker in the form of urlencoded key-value pairs found in the the GET arguments of the URL.

```
GET /announce/?uploaded=314159&compact=YES%21&numwant=3&  
ip=127.0.0.1&info_hash=abcdefghijklmnopqrstuvwxy&event=started&
```

```
downloaded=951413&trackerid=&key=AWANG
&peer_id=evertestclient000000&port=6881&left=1 HTTP/1.0
Host: localhost:6969
User-agent: Python-urllib/1.16
```

The tracker then urldecodes this text, and turns it into a more useful Python dictionary:

```
{'uploaded': '314159', 'compact': 'YES!', 'numwant': '3', 'ip': '127.0.0.1',
 'info_hash': 'abcdefghijklmnopqrstuvwxy', 'event': 'started', 'downloaded':
 '951413', 'key': 'AWANG', 'peer_id': 'evertestclient000000', 'port': '6881',
 'left': '1'}
```

The data in this dictionary is then used to form an appropriate response. Various parts of this, such as the status, downloaded, key, and peer_id, are stored in the database for later use. For example, the tracker honors the numwant optional variable from the client, which is of the value 3, and forms the following dictionary (please note that the peers in this scenario are generated randomly):

```
{'peers': [{'ip': u'68.150.132.78', 'peer_id': u'EVERCLIENT1111111111',
 'port': 12454}, {'ip': u'106.190.185.236', 'peer_id': u'EVERCLIENT8888888888',
 'port': 41582}, {'ip': u'8.56.239.117', 'peer_id': u'EVERCLIENT8888888888',
 'port': 64733}], 'min interval': 240, 'complete': 1, 'interval': 720,
 'warning message': '', 'tracker id': 'EVERTRACKER', 'incomplete': 0}
```

This is then bencoded, and sent to the client as a text/plain document, which is seen as follows:

```
d8:completei1e10:incompletei0e8:intervali720e12:min interval
i240e5:peersld2:ip13:250.192.86.977:peer_id20:EVERCLIENT2222222222
4:porti18252eed2:ip13:156.51.108.137:peer_id20:EVERCLIENT7777777777
4:porti1171eed2:ip13:64.166.125.537:peer_id20:EVERCLIENT4444444444
4:porti25819eee10:tracker id11:EVERTRACKER15:warning message0:e
```

The client then bdecodes it, yielding the following dictionary:

```
{'peers': [{'ip': '250.192.86.97', 'peer_id': 'EVERCLIENT2222222222',
 'port': 18252}, {'ip': '156.51.108.13', 'peer_id': 'EVERCLIENT7777777777',
 'port': 1171}, {'ip': '64.166.125.53', 'peer_id': 'EVERCLIENT4444444444',
 'port': 25819}], 'interval': 720, 'complete': 1, 'min interval': 240,
 'warning message': '', 'tracker id': 'EVERTRACKER', 'incomplete': 0}
```


This data will then be enough such that the client can try to connect to other peers and actually start downloading the file.

3.2.4 Intelligent Announce Peer List Response

When a peer finishes downloading a file and becomes a seed, it no longer needs to be connected to other seeds. Most clients will at this point close and refuse connections to other seeds to further overall performance of the swarm.

A similar feat can be achieved on the tracker-side by recognizing the state of clients as seeds when they send announce requests. The announce response handling method of the tracker was improved such that if a seeding client requests a new list of peers, the tracker will return a list of peers containing only peers without complete copies of the file. This helps increase download performance as well.

3.2.5 Announce Peer List Compression

An optional extension to the announce response functionality of the tracker is to compress the list of peer IP addresses and ports by sending them in binary form instead of ASCII. The ability of the client to support compressed peer lists is identified by the presence of an optional key-value pair in the announce request.

The compressed peer list format ignores the ID component of the uncompressed peer list, and turns the IP address into 4 bytes and the port into 2 bytes. This means huge bandwidth savings, as an IP address can take as many as 15 bytes when ASCII encoded and a port number 4 bytes in ASCII, with the peer id is another 20 ASCII encoded bytes. In addition, the peer list is normally bencoded, which adds on about another 10 bytes or so to each peer. This means that the binary encoded compressed peer list is far smaller than the ASCII encoded uncompressed peer list. Both formats are detailed in the Bittorrent Specification page.

By greatly reducing the size of the peer list for announce requests, the performance of the tracker should scale according to the size of the response size. Since the peer list makes up the vast part of the response, the response time should be much faster than before in non-local situations.

3.3 Goals and Requirements

The goal of this project is a complete solution to publishing files using BitTorrent. The requirements for this project are as follows:

1. Easy to use, automated front end for the user
2. Bencoded .torrent file creation and parsing.
3. Correct implementation of tracker software and tracker-peer communication
 - Announce requests
 - Scrape requests
 - Commonly implemented features present in trackers
4. Implementation of an automatic, permanent "everseed" that prevents the torrent from dying

4 Results

Due to restarting my project second quarter, time constraints prevented me from finishing all aspects of my project. Of the requirements of my project, the .torrent file metadata generator, the bencoding functionality, and all parts of the tracker are functional and tested. The "everseed" client and the web interface have not been completed. The testing Bittorrent client I wrote is able to send both announce and scrape, and the test client and the tracker have been fully documented with the Python API.

References

- [1] Bram Cohen. Incentives build robustness in bittorrent. Technical report, Bittorrent, May 2003. Written by the creator of Bittorrent, it goes into far more detail about the inner workings than the TechRepublic report without giving implementation details. Talks about Pareto efficiency and the prisoner's dilemma that regulate download and upload rates to and from a given client, which insures that there aren't any peers that only download and don't contribute back through uploading.

- [2] P. Garbacki, J. A. Pouwelse, D. H. Epema, and H. J. Sips. The bittorrent p2p filesharing system: Measurements and analysis. Technical report, Delft University of Technology, Department of Computer Science, 2006. Analyzes the traffic of the high-usage Suprnova.com Bittorrent tracker and .torrent file server. It showed the benefits of Bittorrent’s peer-to-peer nature in distributing files, as the content was distributed much faster through Bittorrent compared to additional methods. However, because of the low number of seeds that continue uploading the data indefinitely, the torrent file slows down and dies soon after the initial burst of download traffic.
- [3] D. Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. Technical report, University of Illinois at Urbana-Champaign, Coordinated Science Laboratory, 2006. Goes into great depth analyzing the performance of a generic Bittorrent swarm of clients and uploaders. Uses a simple fluid model to predict how upload and download rate are affected, the overall effectiveness of the Bittorrent protocol as a distribution method, and how well the protocol functions under conditions like selfish peers, low request arrival rate, and bottlenecked upload and download rate.
- [4] Theory.org. *Bittorrent Specification*, 2006. Unofficial specification written by the developers of Bittorrent applications such as clients and trackers. Contains much more implementation detail than the official specification written by Bram Cohen and is a useful engineering reference. Describes the tracker-client communication protocol in great depth and also contains additional test cases for bencode and bdecode.
- [5] R. Tyagi. The technology behind bittorrent means smarter p2p for all. Technical report, TechRepublic, 2005. Explains the major differences between Bittorrent and services such as Napster or Kazaa that make Bittorrent a second generation P2P distribution system. Also has a brief overview of how Bittorrent works, and some of the algorithms that make it efficient and fast, such as choking, rarest first, and strict order.