Project 4: BitTorrent Client

Due: 11:59 PM, Dec 12, 2016

Contents

1 Introduction **2**

2 Warning **2**

3 The Pieces **3**

3.1 Small Essential Components 3

3.2 Tracker Requests 3

3.3 Peering - Downloading 4

3.4 Peering - Seeding 5

3.5 User Interface 5

3.6 Extra Credit 5

4 Getting Started **6**

4.1 Read the Specifications 6

4.2 Download a Client and Wireshark 6

4.3 Finding Suitable Torrents 6

4.4 Get your EC2 instance Running 6

4.5 Design Strategy 7

4.6 Reference Node 7

4.7 Reference Torrents 7

5 Grading **8**

5.1 Tracker Protocol - 15% 8

5.2 Torrent Download - 50% 8

5.3 Torrent Upload - 15% 8

5.4 User Interface - 10% 8

5.5 Readme - 10% 8

5.6 Extra Credit - up to 10% 8

6 Handing In and Interactive Grading **9**

1 Introduction

In this project you will implement a simple BitTorrent Client that will be able to seed and download files from other clients on the internet. The requirements for this assignment will follow closely to the relatively brief de-facto BitTorrent v1.0 Specifications [1]. The BitTorrent official website [2] also has a list of some official specifications that you might find interesting.

In this assignment, you will be able to use any programming language, and any packages that help with SHA-1/MD5 hashing, Bencoding parsing, and filesystem management. Everything related to the BitTorrent protocol, including Bitmap management, must be done by hand. You can use the available HTTP and TCP modules provided by your programming language's standard library. If you ever have a question about whether some package or feature is allowed, please post a public question on Piazza.

This is a partner project, since there is a fair amount of code that needs to be written in the time period available. You can decide to work individually if you want, but the minimum requirements will remain the same, so you will end up having to do about twice the work.

There are many BitTorrent implementations on the web, but you are required to write your own code. We will run a check on your code to verify it is your own. We also will be quizzing you about your implementation during final grading.

Lastly, in order to circumvent Brown's firewall (which is needed for peers to connect you your client), one would need to implement some elaborate NAT traversal algorithms. This is not required. Instead, you will be given an AWS EC2 instance, with port range 6881-6889 opened up for your client to listen on. Actually, this allows up to 9 of your clients to run simultaneously on the AWS machine, and they all can connect to each other. One should appreciate that all mainstream clients handle the difficult NAT problem quite effectively. [3]

2 Warning

BitTorrent is widely known to be used for illegal activity, and in doing this assignment, you might carelessly start using an illegal torrent found on ThePirateBay, KickassTorrents, etc - probably due to a high seed count. **Be very careful what torrents you use**.

To avoid this problem, you can stick excusively with The Internet Archive, [4] which provides BitTorrent alternatives for all its downloads (which runs in the millions). The trackers for the torrents are also run by The Archive. Overall, this seems to be a safe choice for publicly available torrent downloads.

Another alternative is to create your own torrent files, and run your own Tracker for them. This way, you can just serve them privately to yourself.

[1] https://wiki.theory.org/BitTorrentSpecification
[2] https://www.bittorrent.org/beps/bep_0000.html
[3] https://en.wikipedia.org/wiki/Interactive_Connectivity_Establishment
[4] https://archive.org/details/bittorrent

3 The Pieces

Everything discussed in this handout can be found at wiki.theory.org/BitTorrentSpecification, which is concise and comprehensive. If something is not clear in this handout, re-read the wiki. If something is still not clear, post on Piazza.

3.1 Small Essential Components

Before starting with communication, you need to know something about all these small essential components.

- **Bencode**:

  Bencode is similar to XML or JSON - it is just a data format, but is simpler and used by BitTorrent. [5] You can find a plethora of Bencoding parsers for every programming language, and we will allow you to use these parser libraries, since parsing is not really related to Networks. Bencoding is used by the MetainfoFile and is used in the response bodies of the Tracker server.

- **Metainfo File**:

  The MetainfoFile (aka the .torrent file), contains all the information about your torrent, including where the Tracker URL is, what the piece-length, piece-count, and all the piece SHA-1's are, and a bunch of other meta-data. All peers use the same Metainfo file, at least for the SHA-1 hashes. The first thing you will need to do when you want to download a torrent is parse the Metainfo File.

- **Pieces**:

  Pieces are specified in the Metainfo File. Pieces are equally-sized sections of the download. Piece sizes are typically around 2^{19} Bytes (except for the last piece, which may be smaller). Each piece has a SHA-1, which is the hash of the file(s) underyling that portion of the download. You will need to keep track of all your pieces by using a bitmap (1 bit per piece), asserting a bit when the corresponding piece is completely downloaded, which is found out by computing the SHA-1. Peers communicate their piece bitmaps with each other, which allows them to know which pieces to request from which peers.

- **Files**:

  Files are also specified in the Metainfo File. Each torrent can contain multiple files. This means that you might have N pieces and M files, and you need to map the piece address space to the file address space. There is plenty of room for simple math errors here, but you MUST implement all the address mapping by hand. Optionally, each file has an md5, but this seems redundant with piece SHA-1 calculations, so you'll probably never use them.

3.2 Tracker Requests

The first point of communication after adding a torrent to your client is to contact the Tracker to download the list of peers in the 'swarm'. The Tracker's HTTP URL is listed in the Metainfo file.

[5] https://en.wikipedia.org/wiki/Bencode

You are required to contact the Tracker on startup, sending all the approprate fields in the URL query string, and correctly parse the list of peers it sends back. You should include the `compact` flag in your request, but be able to parse both compact and not-compact responses. See the specs for the rest of the fields to include.

You are NOT required to issue `scrape` requests to the Tracker server, since this is just to give you stats, and not essential to your client's operation.

Lastly, you will need to send `started`, `stopped` and `completed` requests to the Tracker server when appropriate. When you send the `started` request, you need to include how many bytes you have downloaded already (at piece-granularity). This means you need to do some SHA-1 calculations before contacting the Tracker.

3.3 Peering - Downloading

After you get the list of peers from the Tracker, you now must connect to as many of them as you can and start downloading from them. These are simple TCP connections with a 2-way handshake to enter the established state. After the handshake, you should send a `bitmap` message, and a `interested` message if you still have remaining pieces that the peer has (which you find out after you receive a `bitmap` message).

After you receive an `unchoke` command from the peer, you can start downloading pieces. You download pieces by requesting blocks, which are usually 2^{14} bytes, so 32-64 blocks per piece is normal. Actually, if you try to request a block larger than 2^{14}, the peer might drop your connection. Also, in addition to a piece bitmap for the whole torrent, you will need block bitmaps for each piece, to see which blocks have been requested and downloaded.

To maximally utilize network bandwidth, you want to have a request queue, which keeps a list of all blocks you have sent a `request` message to the peer for. You want this queue to be 10-20 requests deep, but you can experiment with different queue sizes to see which gets the best download rates. The peer will eventually respond with a `piece` message corresponding to a `request` you previously sent, at which point you need to update your request queue, and your piece and block bitmaps.

If you ever receive a `choke` message from a peer in the middle of downloading, you must immediatly halt all requests, and wait until you get an `unchoke` message. Make sure you issue `keep-alive` messages periodically (120 seconds) and check that all your peers are not timing-out.

Since you will have multiple peers for the same torrent, you will need some way to keep track of which blocks have been requested to all the peers, so you don't send requests for the same block to multiple peers. You algorithm for choosing which blocks to request from which peer is completely up to you, with the minimum requirements that you don't send a request for a piece the peer does not have, and *you don't send a request for a piece you already have.*

To build off of the previous italicized point: if you start up your client application, and the torrent you specify for download partially or wholly exists on disk in the download location, you MUST NOT download the pieces you already have! This means that you need to check the SHA-1 when your client starts (or adds the torrent). So if your torrent is fully downloaded, the next time you start your application, it should immediately start seeding

Finally, the last requirement for downloading is that you send `have` messages each time you finish downloading a piece. You figure this out by periodically running a SHA-1 on the piece's data on disk (which might span multiple files, remember), and if the hash matches the one in the Metainfo

File, you can mark that piece complete in the bitmap, and send the `have` message to all the peers.

3.4 Peering - Seeding

Without seeders, there would be no way to download files using your BitTorrent client. Therefore, after you download the file, your application is required to start seeding it to other peers interested in downloading it. This requires that you implement the `piece`, `choke` and `unchoke`, and handle `request`, `interested` and `not-interested` commands from your peers. If you receive a `cancel` message, you MUST NOT send the corresponding `piece` message if you have it queued up to send.

3.5 User Interface

This part of the assignment leaves a lot of room for creativity: you can implement any style of client you want, as long as it allows you to meet the minimum requirements of downloading and seeding a single torrent to multiple peers, and gives us ability to see all the upload/download statistics. There are 2 main terminal-style interfaces that seem simple:

- **transmission-cli**:

  `transmission-cli` [6] allows you to specify the torrent file, port, and download location via command line switches, and then it prints statistics to the screen every second. The program just downloads your torrent, and then seeds it indefinitely until you kill it.

- **reference client**

  The reference client `btclient`, found in the pub dir, has a command line –debug and –port switches, and then lets you add, remove, or list torrents interactively from a REPL. This is a sort of hybrid approach between transmission-cli and a full fledged GUI like uTorrent.

If you have lots of extra time, you can even implement a GUI. This will most likely not result in extra credit, but might earn you some github stars.

3.6 Extra Credit

You may choose to implement any or all of the following items (or something even not listed here, as long as it is relevant to BitTorrent AND networking):

- **Distributed Hash Tables (DHT)**: BitTorrent without Centralized Tracker. You can usually find more peers here than by using the torrent's Tracker server.
- **Simultaneous Torrents**: Have your BitTorrent client demonstrate the ability to download and upload multiple torrents simultaneously, and a way to view the statistics via your user interface
- **Tracker Scraping**: implement the scraping protocol and provide a way to visualize the results in your UI

[6] https://help.ubuntu.com/community/TransmissionHowTo

- **Download/Seeding Strategies**: A simple download strategy is required for your client, but if you decide to implement something more fancy (like Super Seeding, Rarest-First, End Game), please document it.
- **Optimisting Unchoking**: a simple choking/unchoking strategy is required for your client, meaning you can just unchoke all your peers. If you decide to implement the actual specs from the wiki (of only 4+1 unchoked peers at a time), please document it

4 Getting Started

The following lists, in order, the steps we recommend you follow to implement your client.

4.1 Read the Specifications

If you find the Specifications wiki lacking, there is a decent amount of "How To's", Github Repos, and documentation sites related to BitTorrent.

4.2 Download a Client and Wireshark

You will most surely run into small hashing, indexing, or encoding errors that will cause the Trackers and Peers to close your connections down. You are strongly encouraged to download uTorrent (or another mainstream client of your choosing), and Wireshark to analyse any differences between your client and the mainstream client for the same .torrent file. By this time, most of you should be quite comfortable with using Wireshark, which has native BitTorrent support!

4.3 Finding Suitable Torrents

As mentioned earlier, there are many different Trackers (they are easy to set up and run), but we recommend you use the Internet Archive's BitTorrent trackers for this assignment. You can download everything on the Internet Archive via BitTorrent, and as far as we are aware, all the tracked content is legal. You will probably have the most luck finding a non-zero swarm, and not run into legal issues if you use Archive to find your .torrent files. Additionally, we have provided a select handful of .torrent files that you may use if you wish as well.

Another thing to keep in mind with your torrents is that downloads come in two flavors: single files and multiple-files. Multi-file torrents might be slightly more complicated to work with so you could choose to only work with single-file torrents as you start out, and add support for the multi-file downloads later. Both are required to be supported, however.

It may be useful here to use your mainstream BitTorrent clients with the torrents you downloaded, and analyse them with wireshark.

4.4 Get your EC2 instance Running

The next thing you should do is get your EC2 instance working, and make sure you can run transmission-cli and the reference client on your machine, and all the ports are opened up. See if

you can connect a client on your laptop to a client on the EC2 instance.

4.5 Design Strategy

One possible implementation would implement the design in the following pieces: If you are on a team, the 2nd and 3rd pieces could be done in parallel, since they are similar amounts of work.

- First start out with parsing the Metainfo File, then connect to the tracker. These two things are relatively straightforward, and can probably be done in a short amount of time.
- Then you have to implement the peer protocol, with all the message parsing and message construction. Here you will also need to implement keep-alives and timeouts, uploading and downloading.
- Finally, you need to implement the Piece and File mappings, the Bitmaps used to represent your pieces and blocks, the algorithm for selecting which block to request next, and the File I/O code.

4.6 Reference Node

We have implemented a simple BitTorrent client that meets this project's minimum requirements, and you may use it for testing as well as a guide for what we are looking for. The exact console output does not need to be matched by your client, but it should be close. The client is located in

`/course/cs168/pub/bittorrent/bin/btclient`

and you can also found a version be tested on Windows 10, amd64:

`/course/cs168/pub/bittorrent/bin/btclient_win64`

a version for Mac OS X:

`/course/cs168/pub/bittorrent/bin/btclient_darwin`

You might find it useful to run your client against `btclient -debug=1` which will provide verbose debugging output. You should be able to get most of this information from Wireshark anyways though.

4.7 Reference Torrents

We provide an example list of torrent files in:

`/course/cs168/pub/bittorrent/torrents`

These were just some random torrents of decent size selected from the Internet Archive that we tested the reference client against. You will most likely want to grab your own torrents from the Internet Archive, so you can debug your client using a torrent your classmates are also not debugging with (unless you want to have some fun).

5 Grading

The main requirements for this assigment are high-level:

- Download a multi-file torrent from multiple peers concurrently
- Upload a multi-file torrent to multiple peers concurrently
- Show useful statistics in your console app that lets us see download/upload statistics, currently used torrents, and peer info

You grade breakdown will be as follows:

5.1 Tracker Protocol - 15%

Successfully parse the Metainfo File, send a request to the tracker, have it respond to you with the list of peers to connect to, and parse the list of peers into useful 'ip' and 'port' pairs.

5.2 Torrent Download - 50%

Connect to multiple peers, download the torrent from all of them simultaneously, and appropriately use bitmaps to determine which blocks to request. You must not send requests for pieces you have stored on disk, and you must handle multi-file downloads correctly. You should follow the Peer Wire protocol correctly, but do not need to implement any specific algorithm with regards to choking, and piece selection for download.

5.3 Torrent Upload - 15%

Let multiple peers connect to you, and service multiple streams of requests simultaneously. Here, you need to implement the other half of the peer wire protocol.

5.4 User Interface - 10%

A suitible command-line driver or GUI is implemented that allows for specifying which torrent to download, where to put it, and shows statistics for the active torrents.

5.5 Readme - 10%

Provide documentation on your design, as well as any bugs or features.

5.6 Extra Credit - up to 10%

See the above list for possible ideas - remember that anything for extra credit must be BitTorrent AND Networks related.

6 Handing In and Interactive Grading

Run the handin script before the due date, and set up a time with your mentor for interactive grading by the end of the semester:

```
cs168_handin bittorrent
```

Even if your client does not run on linux, we want you to scp a tarball in and run the handin script. If your having handin script issues, just email the TAs a tarball of your code.

Interactive grading will be done within the week of the handin due date. Your mentor TA will set up a time to meet to demonstrate your client's functionality. We will be testing that you can download a torrent (of our choice) from another one of your clients, from the reference client, and from a mainstream client like uTorrent or transmission. Then we will see if you seed to the reference client. Then, we will have you download from many peers at the same time, and see what your speedup is. Any extra features implemented here can be demonstrated as well. All testing will be done on the TA's AWS instance. So make sure your client works on the AWS instance we provide you, and not just your laptop.

Finally, we will be quizzing you on your design. If you actually did the assignment, this should be very easy.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS168 document by filling out the anonymous feedback form:

`https://piazza.com/class/isqj37mfnyz26r?cid=6`.