BitTorrent Final Project Report

Andre Amor, Dominick Cardone, Feileen Li, Mohammad Durrani

July 7, 2025

1 Supported Features

Our client fully supports the following features:

- HTTP/HTTPS tracker with compact format
- Downloading files from:
  - Official BitTorrent Clients (Transmission Client)
  - Other instances of our BitTorrent clients
  - UMD cerf Server
- Downloading from official BitTorrent clients
- Single-File Torrent Parsing and Downloading
- Basic seeding after download
- Peer connection management
  - Supports periodic tracker re-announcing to discover new peers
- Progress bar to show real-time download progress
- Choking and Unchoking based on highest download speed
- Peer connection management
  - Receiving and sending messages
  - Cleanup for stale or disconnected peers
- Support for Tracker scraping
  - Determines peers with the entire file and number of leechers

2 Overview

Our client implementation follows an asynchronous architecture built in Python using the asyncio library. The TorrentClient acts as the main coordinator, managing peer connections and orchestrating the download process. The PeerManager handles peers, peer states, and block requests. The PieceManager manages the downloading and verification of file pieces. Individual Peer objects handle connections and message exchanges with remote peers. The Tracker component communicates with BitTorrent trackers to discover peers, and the Message class implements the BitTorrent protocol messages. The program uses asynchronous I/O operations throughout to efficiently handle multiple peer connections simultaneously, with tasks for peer communication, piece requesting, and choking algorithm updates running concurrently.

2.1 Torrent/Metainfo Parsing

We use the `bencodepy` library to parse the torrent file and extract information on the tracker URL, file length, and piece length and pieces. We also keep each piece's hash to validate the data after each block is fully downloaded.

2.2 Peer and Connection Management

The client creates a Peer object for each connected peer, and every peer is managed by the PeerManager. Peers are created from sending out initial handshakes or accepting handshake requests. We also have handshake validation to check that the peers have a matching info_hash. Each Peer object has fields to keep track of their states. The primary states needed for our algorithms are interest, choke status, and requests, as well as metrics like upload and download speed. The TorrentClient uses these fields for algorithms such as unchoking the top 4 uploaders.

2.3 Messaging Protocol

We built the Message class with subclasses for each Message type to handle decoding and encoding for sending and receiving. We used Python's `struct` library to ensure that all the messages were encoded Big Endian and decoded correctly after receiving.

2.4 Piece and Block Management

Each file is divided into several pieces, which itself is split up into 16 KB blocks. The PieceManager handles tracking the downloaded and pending blocks, as well as validating the blocks using SHA-1 hashing. The disk writes are done asynchronously, and the PieceManager manages what blocks to request to avoid redundant block downloads.

2.5 Piece Selection Strategy

Our piece selection strategy was relatively simple: we choose the first available non-complete piece that a peer has. Further improvements can be made by requesting for the rarest piece strategy. However, we verified that our piece selection strategy is correct and fully downloads the complete file.

3 Testing & Measurements

To evaluate our client's performance, we ran performance tests for various torrent files from small to larger sizes. We aimed to test for both stability and speed relative to the reference client, for which we used Transmission, which can be found at https://transmissionbt.com/.

3.1 Benchmark Setup

- System: 2019 16-inch MacbookPro, 2.3 GHz 8-Core Intel Core i9
- OS: Sequoia 15.4.1 (24E263)
- Tx (Link Speed): 600 Mbps

3.2 Results

Table 1. Performance for 400KB Flatland text file

Client	Trial	Memory (MB)	User CPU (s)	System CPU (s)	Time to Download (s)
Ours	1	22.49	0.11	0.04	7.2
	2	22.69	0.11	0.02	7.2
	3	22.73	0.11	0.02	7.1
Transmission	1	Unreported	Unreported	Unreported	11
	2	Unreported	Unreported	Unreported	11
	3	Unreported	Unreported	Unreported	11

Table 2. Performance for 1.34 GB Unigine_Superposition-1.1.exe.torrent

Client	Trial	Memory (MB)	User CPU (s)	System CPU (s)	Time to Download (s)
Ours	1	467.86	18.16	12.45	53
	2	414.07	19.23	8.62	59
	3	433.07	18.89	8.39	53
Transmission	1	Unreported	Unreported	Unreported	87
	2	Unreported	Unreported	Unreported	73
	3	Unreported	Unreported	Unreported	73

Table 3. Performance for 2.91GB Linuxmint-22-mate-cinnamon-64bit.iso.torrent

Client	Trial	Memory (MB)	User CPU (s)	System CPU (s)	Time to Download (s)
Ours	1	1933.04	35.22	16.94	139
	2	1390.09	43.42	19.81	164
	3	1862.26	37.86	22.26	146
Transmission	1	Unreported	Unreported	Unreported	139
	2	Unreported	Unreported	Unreported	159
	3	Unreported	Unreported	Unreported	199

Our client performs well compared to Transmission, one of the reference clients. In every test, our client outperformed Transmission on average, though our client uses more memory during downloading than Transmission. Our client also had relatively stable and consistent download speeds. The average difference between the time to download for any given trial and the average time to download for each test did not exceed 6% (the last test and largest file), and was as low as 0.04% for the smallest file.

3.3 Collecting the Metrics

We used the `psutil` library in Python and created another script that keeps track of the specific process id that the BitTorrent client takes. The Python script collects data on total download time, upload and download throughput, CPU time (both user and system), and memory usage.

4 Problems Encountered

4.1 Peer Disconnecting Mid Transfer

If a Peer dies during download, it could cause the pending blocks to be marked incorrectly. We avoided this issue by cleaning up the pending block states as well as cleaning up stale requests and peers.

4.2 Over Requesting

We initially didn't have a cap for the amount of requests our client can make to a peer. This can cause issues with the Peer dropping us for too many requests. We fixed this by introducing a cap with the variable `max_peer_requests` to limit the block requests per connection.

4.3 Tracker Compatibility

Some trackers don't support certain features like scraping, and others require the client to support compact mode. We added fallbacks and exception handling for various different cases to try to be compatible with the tracker.

4.4 Duplicate/Stale Sockets

Initially, we had duplicate sockets for outgoing and incoming connections. In order to adhere to the "bidirectional" nature of BitTorrent, we made sure to track all connections and close any duplicates. Additionally, we added callbacks to our peer message handlings that removed the connection when done.

4.5 Unresponsive HTTP Trackers

With some trackers, the response that we would get, and the headers expected would differ from tracker to tracker. Some needed the compact flag in order to send back a valid request, some required HTTP 1.1 to be specified, some would send a response back in a different format than the others. This required a lot of special casing to ensure that we don't run into errors.

5 Known Bugs or Issues

- Peer ID collisions causing handshake failures.
- We have no support for multi-file torrents.
- We are making some assumptions about how the peers are supposed to behave. We expect them to respond with a bitfield immediately, and we are using the string "BitTorrent protocol" for the initial Handshake.
- When a peer dies, we would expect asyncio to cleanly handle this cleanup, but on some runs, it will report a coroutine error. This does not cause issues, but points to improper configuration with how we are cancelling a task after a peer fails.
- Ubuntu https tracker only reveals one peer to us. This peer does not send us a bitfield and as such, we are unable to download since we don't handle building bitfields. Possible explanation for only seeing one peer is that we can't see IPv6 peers.
- As seen in the metrics, our client has very high memory usage. We are unsure if this is marked due to the file being buffered in memory unintentionally, but when looking at the heap size, it does not grow unboundedly. However, the resident memory size grows to about the size of the file.

6 Contributions

- **Andre Amor (Rodz Andrie Amor):** Wrote the Message classes and subclasses. Wrote parts of peer and implemented download metrics in torrent client.
- **Dominick Cardone:** Wrote Torrent class, implemented background tasks in main, worked on torrent client, peer.
- **Feileen Li:** Worked on Peer, TorrentClient, added HTTPS modification of tracker.
- **Mohammad Durrani:** Worked on PieceManager, HTTP portion of tracker, and speed optimization / refactoring to support request pipelining.