etracker: Healthy BitTorrent Swarms

Daniel Moerner

2025/02/04

Introduction: TLDR

- BitTorrent, like many other decentralized peer-to-peer algorithms, has a free-riding problem.
- We can incentivize fair behavior by manipulating the centralized system for peer discovery: We punish clients behaving unfairly by giving them fewer peers to try to download from.

Outline

- BitTorrent Background
- Live Demo (probably won't work)
- Technical Details
- Future Development

BitTorrent: Background & Jargon

- BitTorrent: Peer-to-peer data transfer algorithm
- Peers with content (possible uploaders) are called “seeders”, peers seeking content (possible downloaders) are called “leechers”.
- A particular collection of peers is called a “swarm”.
- Peer discovery: Either with a decentralized distributed hash table, or with a centralized tracker (our topic).

BitTorrent: Challenges

- Data transfer is zero sum.
- Downloading content has its own reward: You get the data you download.
- But, uploading content has no reward.
- Therefore, we have a free-riding problem: Leechers are free to exit the swarm after they download, and never seed to anyone else.

A New Solution at the Tracker Level

The tracker distributes other peer IP addresses and ports in response response to requests ("announces") by a peer. From the BitTorrent spec:

> *[T]he tracker randomly selects peers to include in the response. The tracker may choose to implement a more intelligent mechanism for peer selection when responding to a request.*

My idea: Seeders with a good reputation get more peers, and therefore better speeds on average, when they try to download new content.

(Failed?) Live Demo

Why did it probably fail?

- Don't sign up for a presentation on the same day as the presentation when you haven't yet tested a live demo.
- I *think* everything on my end is correct, but since my testing is all running from the same IP address, rtorrent is not trying to connect to multiple peers on the same IP.

Technical Details: System Design

- Written in Go.
- Clean and sufficiently rich standard library for this kind of server project.
- Very easy-to-use `net/http/httptest` package for mocking requests.
- All state is stored in a PostgreSQL database.

Client Announces

Each client announces for each torrent in its client:

- left: Whether it has any data left to download.
- downloaded: Total amount downloaded this session.
- uploaded: Total amount uploaded this session.
- numwant: How many peers it wants to receive.

To identify clients, each client has to use a unique announce URL.

Peer Distribution Algorithms (I)

- Calculate a *peerScore*, which is a function of how many torrents a client is seeding and whether they have a good “ratio” of uploaded to downloaded on those torrents.

$$peerScore = seededTorrents * (1 + ratioPercentage))$$

- The higher a downloader’s peerScore, the more peers it receives.

Peer Distribution Algorithms (II)

- It would be cool to not just give a raw count of more peers, but to try to give back "better" peers for higher peerScores.
- Complication: The best peers *to give* are not peers with their own high peerScores. E.g., a high seededTorrents count might be a congested home connection on spinning rust.
- The best peers *to give* are peers with the fastest raw upload speeds and with open ports.
- Too much noise for the tracker to measure this.

Smoothing the Peer Distribution (I)

- You can't be too punishing for new peers and give them too few peers. (Limitations of live demo)
- Current smoothing function:

$$peers = minPeers + (numRequested - minPeers) * tanh(k * peerScore)$$

- minPeers is currently set at 5, but this needs more data.
- *k*, the steepness quotient, is calculated such that peers $\sim=$ numRequested when the torrentSeeded count is one standard deviation above the mean.

Technical Details: Smoothing the Peer Distribution (III)

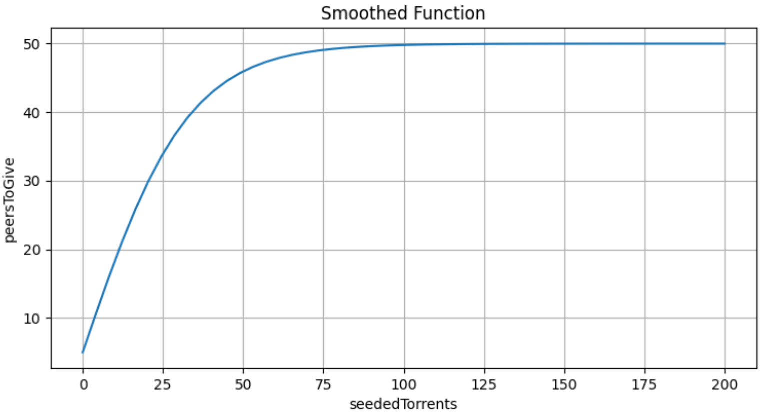

Figure 1: Sample peer distribution

Future Development: Cheater Detection

Once you start to track statistics, there's also an incentive to cheat. Three ways to cheat:

1. Not uploading, but falsely reporting upload to the tracker (plain old "cheating").
    - Reporting a consistent, small amount of fake upload along with real upload.
    - Reporting only fake upload.
2. Announcing to the tracker that you have torrents, but refusing connections to all peers ("fake seeding").
3. Downloading, but not reporting download to the tracker ("ghostleeching").

Catching Cheaters: (I) Faking upload

Heuristics:

- Multiple announces reporting new upload from a single peer, while no other peers report new download.
- Announces with upload amounts which are orders of magnitude greater than the size of the torrent.
- Announcing upload when left $>=$ torrent size
- Multiple announces reporting exactly the same upload amounts.

Catching Cheaters: (II) Fake seeding

- Announcing many torrents with left $= 0$, but never reporting any change in upload for them.
- How do you distinguish poor peering/seeding unpopular content from fake seeding?
- I may need to rethink prioritizing seededTorrents more than ratioPercentage.

Deployment Questions

- Frontend development: Go html templates are awful.
- Bootstrapping swarms: Why would anyone want to start using a tracker which “punishes” people who are starting to use the tracker?

Thanks

- Thanks to Deep Roy, Sherry Bai, Farid Rener, Nikolay Karadzhov, and Satyajit Sarangdhar for pairing with me on this project.
- Thanks to Ian Whitlock for a generous code review on Zulip.
- Thanks to Robbie Lyman for giving me moral support when everything was broken two hours ago!