# P3 Proposal: ByteTorrent

Connor Brem
Billy Wood

April 15, 2014

## Description

ByteTorrent will be a BitTorrent-esque file-sharing application. To run ByteTorrent, we will implement two types of nodes: Clients and Trackers. As with BitTorrent, Clients will share files chunk-by-chunk. Trackers will inform clients which other clients possess chunks of that file.

In order to increase reliability, Trackers will be implemented as clusters, which use Paxos for replication. A Tracker cluster should always be "up".

Clients, on the other hand, can go on- and off-line as they please without losing the "right" to serve file chunks to other Clients when they are online. To ensure this behavior, one Client will never ask a Tracker to remove another unresponsive Client from the Tracker's record. Instead, if one Client thinks that another Client is unresponsive, the first Client will back off.

We ultimately intend to implement graphical interfaces for both the Client and the Tracker. Users will be able to use the Client's interface to manage downloads in progress and construct .torrent files. They will be able to use the Tracker's interface to upload or browse for .torrent files.

## Structure

### Client

The Client nodes will be implemented in Go. They will have the following abilities:

- Create new .torrent files and send them to a Tracker
- Given a .torrent file, connect to a Tracker identified in the .torrent file to determine which other Clients hold chunks of the corresponding data file and fetch these chunks.
- Serve chunks of a file to other Clients on request
- Inform a Tracker when they have a chunk of a file
- Report themselves when they do *not* have a chunk that other clients think they do
- Detect when a Tracker node with which this client is communicating becomes unresponsive, and resume communication with the new master in this Tracker's cluster.

- Expose some of these functions (create .torrent files, fetch file chunks) through a GUI. This GUI will be implemented in Go (using go-gtk: mattn.github.io/go-gtk), since the client UI should be on the same machine as the rest of the client.

### Tracker Server

The Tracker nodes will need to be highly distributed. As such, we've decided to implement Trackers in Go. The Trackers will be able to:

- Receive new .torrent files from Clients
- Send existing .torrent files to Clients
- Keep record of which Clients have which chunks of files
- Maintain state within a cluster despite node failures (via Paxos).
- Expose some of these functions (add .torrent files, fetch/browser .torrent files) via a GUI. This GUI will be implemented in Javascript/HTML, since it has low network requirements and users will likely not be on the same machine as the tracker.

## Distributed Systems Concepts

### Clients

The Client nodes will be in constant contact with each other. They will need to be robust to neighbors dying, and Trackers dying. However, they must also be able to move on- and off-line without other clients reporting them as unresponsive. We plan to use exponential back-off to disengage from unresponsive connections.

### Trackers

Tracker servers will be organized into clusters. Within any cluster, all data will be replicated with Paxos.

## Test Plan

### Tracker Tests

- Kill a Tracker node in a Tracker cluster, and make sure that the affected Tracker cluster is still useful.
- Kill sufficiently many Trackers to stop progress, and make sure that all progress halts.
- Create slow Trackers, and make sure that the Tracker cluster stays consistent.

### Client Tests

- Kill a peer which has a chunk that Client $A$ wants, and ensure that client $A$ backs off and retrieves the chunk from another peer.
- If some Tracker in a Tracker cluster becomes unresponsive, ensure that Clients find and resume communication with the new leader in the cluster.
- Ask a Client for a chunk that it no longer has, and ensure that the asked Client reports that it does not have the chunk.
- Test with a slow Tracker cluster on start-up.

## Development Tiers

### Minimum Testable Product

Our primary goal is to create a Paxos-replicated Tracker. As such, our minimum product will have most of the same Tracker functionality, with a simplified Client program. In addition, our minimum product will only support a single .torrent file per Tracker cluster.

- Client:
  - Get chunks from other clients
  - Serve chunks to other clients
  - Can fail without disrupting things
  - Cannot create new .torrent files.
  - Will not report when Client does not have a chunk
- Tracker:
  - Maintains data for a single .torrent file
  - Paxos-replicated cluster
  - Record when new clients get chunks of files
  - Does not need to receive new .torrent files

### Target Deliverable

Our target deliverable will allow for multiple .torrent files to exist on each Tracker cluster. At this stage, we also make the Client program slightly more complicated.

- Client:
  - Self-report to tracker when Client does not have a chunk that other Clients expect it to
  - Compute/create checksums on torrents to ensure security
  - Maintain target seed-to-leech ratio
- Tracker:
  - Record new .torrent files

### Stretch Goal

For our final development stage, we want to add user interfaces to the Client and Tracker, for ease of use.

- Client:
  - User Interface
- Tracker:
  - Maintain users/manage logins
  - User Interface

## Development Schedule

### Week 1: April 14-20

For Week 1, our goal will be to have the Minimum Testable Product prepared. We will primarily spend our time writing a Paxos-replicated Tracker cluster, and allowing Clients to send and receive chunks.

### Week 2: April 21-28

For Week 2, we will focus on getting our project to the Target Deliverable state. For this, we will mostly be working on the Client, to implement various security and consistency features. The Tracker will need minor updates, to allow the receipt of new .torrent files.

### Week 3: April 29-May 1

In Week 3, we will work on the Stretch Goals. This primarily means implementing GUIs for the Tracker and Client, but we will also need to maintain users and manage logins.