λTorrent

Rahul Jha, Akshat Singh Tiwari

Nov' 24

Contents

Introduction

1. Bandwidth is a scarce resource, and CDNs are expensive.
2. BitTorrent is a way to transfer large files fast* using a peer-to-peer model.
3. We implement a substantial subset of the protocol from scratch in Haskell.

19:but what is bencode

1. **Bencoding** is a way to organize data in a terse format. It supports the following types: byte strings, integers, lists, and dictionaries.
2. Torrent files, among other things, employ this scheme to serialize information.
3. The spec is very simple, and can be found in the unofficial documentation.

The Parser

1. The parser is available in `src/Bencode.hs`. We **do not** use any parsing libraries like **Parsec**.
2. To play with the parser, just load the file (or better, just use `cabal repl` to load all modules) and then

```
ghci> run_parser bencode_parser $ BC.pack "19:but what is bencode"
Just ("",BBString "but what is bencode")
ghci> bencode_deparser (BBString (BC.pack "but what is bencode"))
"19:but what is bencode"
```

Figure: Examples

3. We verified that the parser worked correctly by parsing the torrent file, re-encoding the `info` dictionary and matching its SHA1 hash with the expected hash.

Tracker Protocol

1. After parsing the torrent file, the next step is to obtain a list of *peers* (other clients running the protocol that have the files we need).
2. This is done using the **Tracker** protocol implemented in `src/Tracker.hs`.
3. To see this in action, load the file and run `(a,b,c) ← get_announce_result <filename>`.
4. This assigns the hash of the `info` dictionary to `a`, the actual dictionary to `b` (explained later), and the list of peers to `c`.

```
ghci> (a,b,c) <- get_announce_result "examples/manjaro.torrent"
ghci> c
[("14.139.38.127",2000),("202.3.77.205",2000)]
ghci> (a,b,c) <- get_announce_result "examples/sample.torrent"
ghci> c
[("202.3.77.205",2000)]
```

Figure: Examples

1. Once we have a list of peers, we need to follow the peer wire protocol to actually download the pieces we need. This code is implemented in `src/Peers.hs`.
2. The detailed specification is in the unoffical spec.
3. We only implement a subset of the messages, and run the demo using a local Python tcp server (it works with actual peers but they refuse to connect to random Haskell sockets).
4. We demo this in class.

1. This is not a commercially viable implementation because Haskell's web libraries are, for lack of a better word, primitive (had to hardcode custom trackers).
2. Eventually, the goal is to complete PWP implementation, and add parallelism to increase viability.
3. A more ambitious goal is to probably run a full DHT node.

References

1. We primarily referred to the unofficial spec at TheoryOrg.
2. The code is available here. Just clone the repository and follow the README.