Torrent

Mechiel Lukkien

mechiel@xs4all.nl

ABSTRACT

The *torrent** project contains a *BitTorrent* program and tools for creating *.torrent* files and *''tracking''* a torrent. It is written in Limbo, for Inferno. *Torrent/peer* connects to many peers and exchanges data blocks with them. It serves a styx/9p interface from which progress can be read and its behaviour influenced. This interface is used by *wm/torrent*, a Tk program that visualizes progress and allows stopping/starting and setting bandwidth limits.

Introduction

This report briefly introduces the BitTorrent protocol†, explains the functionality and styx interface of *torrent/peer*, the user interface provided by *wm/torrent*, implementation details, and concludes with a discussion and future work.

BitTorrent

BitTorrent is a popular peer to peer protocol for file exchange over the internet. A *.torrent* file references a *tracker*, SHA-1 hashes of *pieces* of the data that are exchanged, and file names belonging to the data. An *info hash* (SHA-1 again) can be derived from this information and is the unique identifier of the torrent file. Peers use the *info hash* to determine whether they are talking to a peer that exchanges the same data. The tracker helps peers find each other, returning lists of peers interested in the same data. The tracker is the only centralized component used during data exchange, though decentralized trackers also exist nowadays. The SHA-1 hashes in the torrent file allow verification of the received data. The file names in the torrent have no role in the protocol: multiple files are treated as a sequential stream of bytes during data exchange. All pieces (except the last) are of the same fixed size, typically between 64KB and a few MB. Smaller *blocks* of a piece, of typically 16kb, are exchanged at a time. Once all blocks for a piece have been received, the piece is verified and from then on exchanged with other peers. The *torrent* file is encoded in the *''bee''* format, a simple BitTorrent-specific format that can encode lists, dictionaries, integers and (octet) strings.

An implementation connects to the tracker periodically to fetch a list of peers, and then dials those peers (unless enough peers are already connected). It also listens for incoming connections from other peers. It keeps track of the pieces each peer has, and keeps all peers informed of all the pieces it has itself. A connection to a peer has two bits of state on both the *local* side of the connection and the *remote* side: whether each side is *interested* (i.e. wants a piece the other side has), and whether it has *choked* the

* *Torrent*, `http://www.ueber.net/code/r/torrent`

† *The BitTorrent Protocol Specification*, `http://www.bittorrent.org/beps/bep_0003.html`

connection (i.e. is not willing to send blocks). If the local peer is *interested* in the remote peer, and the remote peer has not *choked* the local peer, *requests* for blocks are sent to the remote which the remote peer answers with blocks.

To keep TCP working reasonably (with slow-start, back-off, etc.), only a limited number of peers are selected for sending data to, i.e. *unchoked*. The set of peers to send data to is evaluated periodically. The best performing peers are (kept) unchoked, all others are choked. Performance is measured by the peer's contributed bandwidth. A random peer is unchoked once in a while, hoping it will appreciate our bandwidth and reciprocate. This simple mechanism finds good peers to exchange data with.

The pool of connected peers is kept healthy too. In torrents with many peers (large *''swarms''*), replacing existing connections with new peers ensures good piece distribution and gives new peers a chance to get data.

These are all standard BitTorrent mechanisms. There are many details an implementation has to care of. For example, it has to defend against freeloading peers, or peers that send blocks with wrong data (whether deliberate or not).

Torrent/peer

Over the years, various extensions have been added to the protocol. Not all have been implemented. The feature that makes *peer* different from most implementations (but not *btfs*!) is its styx interface. This interface is probably not generally useful, but it does give a nice separation of the protocol details and controlling the process and showing its progress. Perhaps a *web interface* will be implemented in the future though.

The following example illustrates the current styx interface. Be warned that it will likely change.

```
% mount {torrent/peer glenda.torrent} /mnt/torrent
% cd /mnt/torrent
% ls -l
--rw-rw-rw- M 4 torrent torrent 0 Jan 01  1970 ctl
--r--r--r-- M 4 torrent torrent 0 Jan 01  1970 files
--r--r--r-- M 4 torrent torrent 0 Jan 01  1970 info
--r--r--r-- M 4 torrent torrent 0 Jan 01  1970 peerevents
--r--r--r-- M 4 torrent torrent 0 Jan 01  1970 peers
--r--r--r-- M 4 torrent torrent 0 Jan 01  1970 peersbad
--r--r--r-- M 4 torrent torrent 0 Jan 01  1970 peerstracker
--r--r--r-- M 4 torrent torrent 0 Jan 01  1970 progress
--r--r--r-- M 4 torrent torrent 0 Jan 01  1970 state
% cat info
fs 0
torrentpath glenda.torrent
infohash f52fe0191737e1c3e6e86f0081fa52d182e12a70
announce http://localhost/announce
piecelen 65536
piececount 10
length 654030
% cat files
spaceglenda300.jpg spaceglenda300.jpg 654030 0 9
% echo start >ctl
%
```

Commands can be written to the `ctl` file, e.g. to start/stop data exchange, or to set bandwidth limits. A read on `ctl` returns the values of configurable parameters. `Info` returns properties from the torrent file. `State` returns most of the progress (bandwidth rates and totals of the transfer) and e.g. the number of connected peers. `Files` lists the files described by the torrent file. Each line consists of a path (sanitized by default, so no spaces and other shell and text-selection unfriendly characters), total size in bytes, and first and last piece that has bytes for this file. `Peers`, `peersbad` and `peerstracker` give information about the connected peers, a list of misbehaving peers, and addresses of peers that are known but not necessarily connected. `Peerevents` returns events about peers, one line per event. For example for newly connected peers, or a change of interestedness or chokedness, or when peers say they completed a piece. `Progress` returns events about progress *peer* itself is making, e.g. when a new piece is complete, or when checking the hash for a piece failed.

Wm/torrent

The styx interface exported by *peer* is used by *wm/torrent* to keep track of progress and allow setting of controls. *Torrent* shows information such as percentage of pieces completed, current upload and download rates, total number of bytes uploaded, downloaded and remaining, the number of connected peers. Two ‘‘piece bars’’ visually indicate which pieces have been downloaded and to what extend pieces are available at other peers. Another view shows information per peer, including their progress, network address, software version identifier, and upload/download rates and totals. A third view shows a list of ‘‘faulty’’ peers, those that did something wrong such as sending bogus BitTorrent messages or invalid data.

Torrent/track, torrent/create and torrent/verify

Track is a very simplistic tracker. It can be configured to serve a preset list of *info hashes*, or any *info hash* that comes along. It runs as an *scgi* program.

Create creates a *.torrent* file for a list of files that are to be exchanged. The tracker must be specified as well. *Create* logically divides the files into pieces and calculates their SHA-1 hashes for inclusion in the torrent file.

Verify calculates the SHA-1 hashes of files specified in a torrent file and compares them with the hashes in the torrent file. It prints which pieces are complete.

Implementation

The obligatory line counts:

```
    2824    9173   70618 ./appl/cmd/torrent/peer.b
     139     444    3022 ./appl/cmd/torrent/create.b
     100     256    1922 ./appl/cmd/torrent/verify.b
     719    2317   16644 ./appl/cmd/torrent/track.b
     214     583    3432 ./appl/lib/bitarray.b
    1007    3080   20351 ./appl/lib/bittorrentpeer.b
     346    1316    8852 ./appl/lib/bittorrentpeer.m
    1076    3889   25111 ./appl/lib/bittorrent.b
    1305    4267   30196 ./appl/wm/torrent.b
      39     166    1002 ./module/bitarray.m
     132     531    3443 ./module/bittorrent.m
    7901   26022  184593 total
```

Future work

The BitTorrent protocol has only ten very simple protocol messages. The file format of the torrent files is simple too, and the responses from the tracker are in the same format. Most of the work consists of managing all the connections, making sure all peers that are willing to transfer data receive requests, in a pipelined fashion. For each peer we have to keep track of the pieces they have, which of those we still want, which of those have not yet requested, etc. Preventing abuse plays an important role too. Thus, the most complicated part is all the accounting, keeping all the information in a consistent state and quickly accessible (at low cpu cost).

Decisions are made continuously: which piece to request next, which peers to unchoke. These decisions can be made with ''smart'' algorithms, e.g. based on previous actions by the peer. However, that greatly complicates the accounting and is susceptible to abuse. A simple and robust approach is to pick one of the options at random. It is cheap to execute, requires little bookkeeping and typically less prone to abuse.

There are many things that need improvement, listing them here would be too much (and too detailed). Some of the immediate or larger items on the list:

- Quality of peers should be taken into account more when requesting blocks. This provides robustness against malicious peers that send wrong data. A mechanism to divide clients by whether they have delivered a full piece, delivered blocks of a completed piece, are of unknown quality, or have mistreated us in the past has been implemented partially.
- *Torrent/peer* should handle multiple torrents at once. Currently multiple *peer*'s and *wm/torrent*'s have to be started. This is not necessarily bad. However, transferring multiple torrents in a single *peer* allows for better traffic and connection optimisation, i.e. getting more bandwidth in return for given bandwidth. For example the *n*m* best peers over all *m* torrents can be unchoked, instead of the best *n* for each torrent.
- UDP trackers, as opposed to the default HTTP over TCP trackers, might be useful, though mostly to lower the load on trackers.
- Http seeding extensions, for retrieving pieces from a web server when no peers with those pieces are available. It is not clear how commonly this is used though.
- *''Magnet URIs''* and the BitTorrent extensions protocol message could be implemented. It allows exchange of torrent files among peers, given en *info hash*. This makes BitTorrent more decentralized.
- Pieces are currently always requested in random order. Rarest-first piece selection could be implemented, to ensure better piece availability. It requires more accounting though, and is susceptible to manipulation.
- All pieces from the torrent file, thus all files specified in the torrent file are downloaded by *torrent/peer*. Support for selection a subset of the files may be implemented.

Testing is also a challenge, for example to test whether an anti-abuse measure works requires an abusing peer. Even though the protocol is simple, there are still lots of corner cases that need testing.