Btfs - a BitTorrent client

Mathieu Lonjaret
lejatorn@gmail.com

ABSTRACT

Btfs is an attempt at an implementation of the *BitTorrent* protocol. The primary goal is to design and program a BitTorrent client for *Plan 9 from Bell Labs* (referred to as *Plan 9* in the following) providing the basic functionalities found in most BitTorrent clients. The general design and user interface are elaborated with the Plan 9 file server model in mind.

1. Motivation

To start, there is no usable implementation of a BitTorrent client available on Plan 9. As it is a pretty popular protocol nowadays, it seems like a good idea to provide that possibility for a (hopefully) growing userbase. The distribution of the Plan 9 system could also benefit from it: there already are a few sites out of Bell Labs - already mirroring the Plan 9 iso - which could altogether be seeders for a torrent of the iso, hence relieving some of the load on the Bell Labs server.

Additionally, the BitTorrent protocol is both pretty simple and interesting as far as network protocols go, hence getting a lot of interest from various projects. Its distributed nature, the tracker component set aside, could lead to interesting developments on the Plan 9 platform. For example, the idea of a kind of coupling between BitTorrent and some venti servers to distribute the blocks was suggested.

2. Choices

2.1. Port or native

A port of the mainline BitTorrent client would probably be possible since Python has already been ported. However, there is no apparent obstacle to a native implementation following the Plan 9 file server approach. Appart from the obvious satisfaction of having a native program which integrates well with Plan 9, it will also give the opportunity for an acme program as an additional user interface later (akin to acme Mail). Finally, designing this project in its entirety is an educational opportunity to find out how well Plan 9 specific concepts like the file server approach and the csp model would fit in for a peer to peer program.

2.2. Language

As most of the Plan 9 code is written in C, this is the obvious choice. It allows for a better integration with the rest of the system and makes it easier if one wants to reuse pieces of code from other programs. However *limbo* fans should note there is another ongoing effort for a BitTorrent client (on *inferno*) being written in limbo.

2.3. HTTP queries

A BitTorrent tracker listens to http queries, thus one could write and add to the client the code needed to send such queries, or one can rely on already existing components such as hget or webfs. The latter was chosen, because there is no need for a rewrite when webfs is perfectly suited for this task, and again because it fits better with the file server way. However, queries to trackers can contain NULs, and webfs

See mjl's contrib: http://plan9.bell-labs.com/sources/contrib/mjl/

currently does not allow that, so it needs to be slightly modified.

2.4. Threads

A BitTorrent client, like any peer to peer program, has to interact with a lot of other peers, and it is composed of several independant tasks, therefore it makes sense to distribute these tasks amongst several threads or procs. As threads are inherently safe to avoid race conditions, and since procs did not seem necessary, only threads are used for now.

The scheduling in use is what seemed to be the simplest: for each peer one thread is created and it gets all the pieces it can from this peer. The threads are synchronised with an *Alt* structure and each of them relinquishes control with *send(2)* at points in the execution where it would supposedly wait for network packets from the peer to come. An *Ioproc(2)* is used to avoid the situation where all the threads are blocked because one of them is waiting on a dial. This design is of course subject to changes depending on how well it will perform.

3. Implementation

The BitTorrent protocol makes use of a file (usually with .torrent extension) of metadata to describe the so called *torrent.* Btfs creates a Torrent struct in memory to represent this torrent and its metadata. Similarly, a Peer struct is used to hold the properties of each peer which btfs is communicating with.
The program is divided into four main parts as of now: the file server operations, parsing of the torrent file, operations on the Torrent structure, and communications with the tracker(s) and other peers.

The file server provides a synthetic tree created with *alloctree(2)* (mounted by default on /n/btfs) on which the user can act upon. The basic structure of the tree is envisionned like the following:

```
/
|-ctl
|-torrents
        |-c3cb0dfd1d2839861ea79367145b202db7c09c52
                |-tracker
                        |-announce
                |-pieces
                ...
        |-9a5a04f1ddff16de8f7bca8714e945083c4c53c7
        ...
```

where torrents are identified with their infohash (like c3cb0dfd1d2839861ea79367145b202db7c09c52). One can imagine a lot of others files which, when read, would return some useful information. Writing commands to the *ctl* file will be the basic way to control btfs, ie to add new torrents, stop some active torrents, throttle the upload/dowload rate, etc...
At the moment, only adding a torrent is supported, with the ’add /path/to/file.torrent’ command.

The torrent file metadata are organized as a bencoded structure - a simple encoding which consists of a dictionary (associative array) of key/value pairs, the values themselves being of one of the following types: byte string, integer, list, dictionary. For now, these data are just read sequentially and analysed according to a documented set of dictionary keys. The corresponding values are stored in the Torrent structure as the decoding goes. This part should and will most certainly be rewritten as a more generic parser.

Most of the operations on the Torrent structure relate to the torrent pieces. Those are represented with linked lists: a global one, and one for each peer. The elements of the global one correspond to the whole torrent’s pieces, and hold information such as the position of the piece in the torrent. The elements of a peer’s list represent the pieces which are yet to be downloaded from this peer.

The communications start with a call to the tracker, after the right request have been forged, using the information from the torrent file. The tracker reply is parsed in the same fashion the torrent file was. As said before, a thread is created for each peer reported by the tracker, with the following limit: the maximum number of peers is arbitrarily fixed (to 20) for now - some specifications on the protocol state that for best

See patched /sys/src/cmd/webfs/url.c: http://plan9.bell-labs.com/sources/contrib/lejatorn/url.c

efficiency this number should not be too high (< 55) anyway.
Each thread requests from its dedicated peer all the pieces, in random order, that the peer announced it had, until all of them have been acquired.

4. Current state

A lot more needs to be done before btfs displays most of the expected functionalities from a BitTorrent client. In particular, the current main issues are: absolutely no support for seeding, no reply from some specific trackers, and dealing with the concurrency for communicating with several peers at the same time.

No seeding not only means that it is currently impossible to distribute a resource, it also means that the download rates will be impaired since most clients choke peers which do not seed in return.
The trackers issue will be investigated soon, it probably comes down to the tracker expecting an optional parameter in the query, like the peer key or the tracker id.
As mentionned before, the general problem of multi peers communications without the threads blocking each other should have been solved by the use of ioprocs. However, the current behavior is not what was expected (all of the connections to the peers get closed except for the first one). This is the most immediate concern and is what needs to be fixed first and foremost, therefore any help on the matter would be greatly appreciated. Especially since once this is solved, btfs is expected to be usable to download in quite a few cases.

5. Todo

On a longer time scale, the following items are planned:

- General improvements of the code: better error handling, fix some corner cases, hardcoded values, and ugly algorithms.
- Better communications: requery the tracker regularly to get fresh info, reconnect to some peers, keep a pool of threads ready. Use the other trackers in the tracker list.
- Enhance the file server: more "features" through the ctl file, and more files to read for info.
- Add rarest pieces first, and end game algorithms.
- Add dht support.
- An acme program as an additional user interface.

6. Check it out

Btfs can be downloaded from http://plan9.bell-labs.com/sources/contrib/lejatorn/; the directory usually contains the latest updates while the tarball should be a bit more stable. One will of course need a valid torrent file to try btfs out. Either get it from any of the numerous torrent hosting sites or create it yourself, with the mainline torrent creator *btmakemetafile* for example.
A quick tutorial:

```
webfs
btfs [-d datadir] [-m mountpoint] [-v]
echo 'add /path/to/the/torrent/file.torrent' > /path/to/the/mountpoint/ctl
```

Datadir is the directory where the downloaded files will be written (defaults to the user's home), mountpoint is where the btfs tree will be mounted (defaults to /n/btfs), and -v is for some debug verbosity (very verbose, therefore much slower).

See http://wiki.theory.org/BitTorrentSpecification#Tracker_Response
The btfs binary is not copied out of its source directory, so one will have to copy it where suitable, or bind it, or invoke it with its path.