Team 22
Open Bittorrent Directory Replication Final Report

Adam Howard
ahowar31@vols.utk.edu

Maurice Marx
mmarx@vols.utk.edu

John Reynolds
jreyno40@vols.utk.edu

Jeremy Rogers
jroger44@vols.utk.edu

Matthew Seals
mseals1@vols.utk.edu

April 24, 2015

Customer: Dr. James S. Plank

Contents

1 Executive Summary

The purpose of the Open Bittorrent Directory Replication project is to develop a free alternative to the Bittorrent Sync utility. Bittorrent Sync offers users file replication across multiple computers in a way similar to the widely used Dropbox application, but is built on top of the Bittorrent Protocol, requires no centralized file server, and has no limits on the size of the data that can be stored in it. Our team's goal for the course was to produce the desired software suite and support systems. To help make this goal feasible, we decided to split the project into three main parts: the torrent utility, the tracker, and the configuration tool. The torrent utility can be viewed as the client program for out product. Users may run it as either a foreground or a background process. The tracker is a web server that will keep track of which of the daemons is online, and to communicate to that daemon the locations of its peers. The configuration tool is a script which the user can use to change settings for the utility, such as the login information of the tracker or the directories that need to be monitored. The configuration tool also has a graphical user interface, or GUI, that presents a user-friendly alternative to the command line for changing the Open Bittorrent Directory Replication software settings. As of the time tis report was written, the core functionality of all three parts is finished, and the software is to be released under the BSD 3-Clause License.

2 Requirements

1. Definitions (*Taken from http://jonas.nitro.dk/bittorrent/bittorrent-rfc.html*)

1.1. Peer - A peer is a node in a network participating in file sharing. It can simultaneously act both as a server and a client to other nodes on the network.

1.2. Neighboring peers - Peers to which a client has an active point to point TCP connection.

1.3. Client - A client is a user agent that acts as a peer on behalf of a user.

1.4. Torrent - A torrent is the term for the file (single-file torrent) or group of files (multi-file torrent) that the client is downloading.

1.5. Swarm - A network of peers that actively operate on a given torrent.

1.6. Seeder - A peer that has a complete copy of a torrent

1.7. Tracker - A tracker is a centralized server that holds information about one or more torrents and associated swarms. It functions as a gateway for peers into the swarm.

1.8. Metainfo file - A text file that holds information about the torrent, e.g. the URL of the tracker. It usually has the extension .torrent.

2. Utility

2.1. Usage

2.1.1. The utility will run as a background process (daemon) on the local system.

2.1.2. The utility will monitor a number of specified directories on the local file system.

2.1.3. The utility will automatically create and modify a Metainfo file (.torrent) for each monitored directory.

2.1.4. The utility will periodically check monitored directories for file modifications by their timestamps.

2.1.5. The utility will inform peers when a modification occurs in one of its monitored directories by distributing an updated version of its Metainfo file.

2.1.6. The utility will listen for updates from peers, and will modify its monitored directories to reflect that update.

2.2. Configuration

2.2.1. The utility will have a number of configuration parameters which can be adjusted based on the user's preference.

2.2.2. Configuration parameters will be specified within a text file stored on the local file system.

2.2.3. Networking settings will be defined within the configuration file.

2.2.4. Directories to be monitored by the utility will be defined within the configuration file.

2.2.5. Each directory defined in the configuration file may have a number of parameters defined (required parameters are indicated and optional parameters will be given some default value).

2.2.5.1. The directory's path on the file system.

2.2.5.2. An identifier for the directory.

2.2.5.3. How often the utility should check the directory for updates.

2.2.5.4. Upload/Download rate limits for transfers to and from the directory

2.3. File Manipulation

2.3.1. The utility will attempt to modify files in the monitored directories, as well as Metainfo files, automatically.

2.3.2. The utility will allow the user to add files within a monitored directory.

2.3.3. The utility will allow the user to remove files within a monitored directory.

2.3.4. The utility will allow the user to rename files within a monitored directory.

2.3.5. The utility will allow the user to modify files within a monitored directory.

2.4. Network Communication

2.4.1. NAT and Gateway

2.4.1.1. The utility may automatically choose a port and use Universal Plug and Play (UPnP) to ensure that it is accessible behind a router.

2.4.1.2. The user may manually define the networking settings if he/she chooses to do so.

2.4.2. Tracker

2.4.2.1. Users may communicate with any trackers they choose for coordinating their swarm of devices.

2.4.2.2. Security considerations are only guaranteed with any degree of certainty when communicating with tracker services we provide.

2.4.2.2.1. No passwords will be stored in plaintext by a service we control.

2.4.2.2.2. Communication with the tracker will be encrypted using TLS/SSL.

2.4.2.3. Trackerless operation using Distributed Hash Tables will be supported.

2.4.3. Peers

2.4.3.1. Users will be required to authenticate a new peer to any given client.

2.4.3.2. Peers will communicate to trade up-to-date versions

of the monitored directories' Metainfo files.

2.4.3.3. File transfer between peers will utilize the Bittorrent Protocol.

3. Supporting Software and Services

3.1. Configuration Tool

3.1.1. The Configuration Tool can be issued commands to start and stop the utility.

3.1.2. The Configuration Tool will provide a command line interface.

3.1.3. The Configuration Tool will modify the text file the utility users for configuration.

3.1.4. The Configuration Tool will allow the user to add to, remove from, or modify the list of monitored directories without modifying the contents of the monitored directories.

3.2. Application-Specific Private Tracker Service

3.2.1. We will provide users a tracker for instances of the utility to connect to.

3.2.2. The service will provide a web interface for managing the tracker.

3.2.3. The service will require user registration and authentication.

3.2.4. The service will allow users to add or remove monitored directories.

4. Standards, Licensing, and Supported Platforms

4.1. The software will be published using the BSD 3-Clause License.

4.2. The utility will comply with Bittorrent Protocol Version 1.0 (BTP/1.0).

4.3. All software components will support deployment on the GNU/Linux operating system.

4.3.1. The Debian distribution and its derivatives will be specifically targeted for support.

3 Change Log

4 Design Process

4.1 Tracker

To manage the daemons, a server was needed that would keep track of which of the daemons is online, and to communicate to that daemon the locations of its peers. This server is referred to as a tracker, and is a webserver that responds to requests from the daemons with an up-to-date list of all the peers it is currently communicating with. In addition to communicating with the daemons, the tracker will also needed to allow the user to manage

their account and the directories which are monitored and shared between the daemons.

The tracker was developed using Django, a high-level Python web framework. We chose Django because it provides many features that helped to alleviate the tracker design, such as user accounts and automatic form validation. The primary reasons that we chose Django over any alternatives is that it was free and simple to use, and it is published open-source under a BSD license, so it was appropriate for the intended distribution of this project. The tracker application exposes two main "views," that observed by the utility, and that which the user, or the configuration tool acting as the user, contacts to allow and disallow certain peers and to manage which directories are to be monitored.

Daemons communicate with the tracker over Bencoded HTTP requests and responses. A Bencoded request contains a dictionary of key/value pairs describing the daemon and the directory it is requesting information about. The response contains a list of active peers that the tracker is aware of, and that the daemon contacts to attempt to obtain parts of the requested files. It is important to note, that while the tracker maintains the peer information, it does not have the content of any of the files within the directory and does not necessarily know anything about the torrents it manages.

4.2 Configuration Tool

The Configuration Tool is such that we needed to come up with a configuration file structure that was not only easily human-readable, but also easy to parse from a programming standpoint. Originally, while brainstorming a way to do this, we decided to use an nginx-style structure, as follows.

```
# Top-level block, also this is a comment
daemon {
        # Where to find the daemon's peer_id
        peer_id_file ~/.btsf/id;
}

# Should only be one of these blocks
network {
        # Where to find the tracker
        tracker_domain www.example.con/tracker/;

        # Authentication creditials for the tracker
        tracker_username john.t.example@example.com;
        tracker_password supersecretpassword;

        # The port the daemon users for peer communications
        daemon_port auto;
```

```
}

directory {
        # Directory identifier
        directory_name pictures;

        # Directory's torrent file
        directory_torrent ~/.btsf/pictures.torrent;

        # Path to the directory on the local machine
        directory_path ~/personal/pics;

        scanning_rate 3000;
}
```

This proved to be difficult to read and edit, and also appeared to be difficult to parse. After researching different ways to go about this, we came across the ConfigParser python library, which creates, modifies, and writes to configuration files of a different format, shown below.

```
[daemon]
PeerName = peer
PeerID =
ScanRate = 60
```

```
[network]
TrackerUsername = jroger44
DaemonPort = 8000
TrackerDomain = home.elemnir.com

[movies]
ID =
DirectoryPath = ~/movies

[music]
ID =
DirectoryPath = ~/music

[pictures]
ID =
DirectoryPath = ~/Documents/Pictures
```

The IDs are blank because they are filled in with the return value from the tracker, and this is a fresh configuration.

We used Python 3 to implement this tool, since we felt that file I/O and parsing are a strength of a scripting language such as Python. Also, the team members working on this part had previous experience using it.

Originally, the configuration tool consisted of two parts: a first time setup script and a bootstrap script. The setup script was only for first time setup, and it had the user input setting such as the tracker domain, the tracker username, and the directories to be monitored. The bootstrap script was to register the user, the peer, and all of the shares (or directories) with the tracker. This approach was abandoned later in the project for a more customizable approach. With the current style of the script, it can be invoked with `python openbdr_config.py [command]`, where "command" some function, such as `addshare, setuser, registeruser`, etc. This was to give the user more control over the configuration tool, and to make writing the GUI easier.

The other part of the configuration tool is the graphical user interface, or GUI. For this, we chose to use the Qt framework to implement this part. We chose Qt for a number of reasons, but the main ones being ease of use and familiarity. Qt is easy to use thanks to its language, which is C++ combined with QML, Qt's proprietary language. Since the team has experience in C++, QML was not a problem to learn. The Qt application shows the options that can be edited with the configuration tool, and will edit them appropriately.

5 Lessons Learned

6 Contributions

7 Signatures