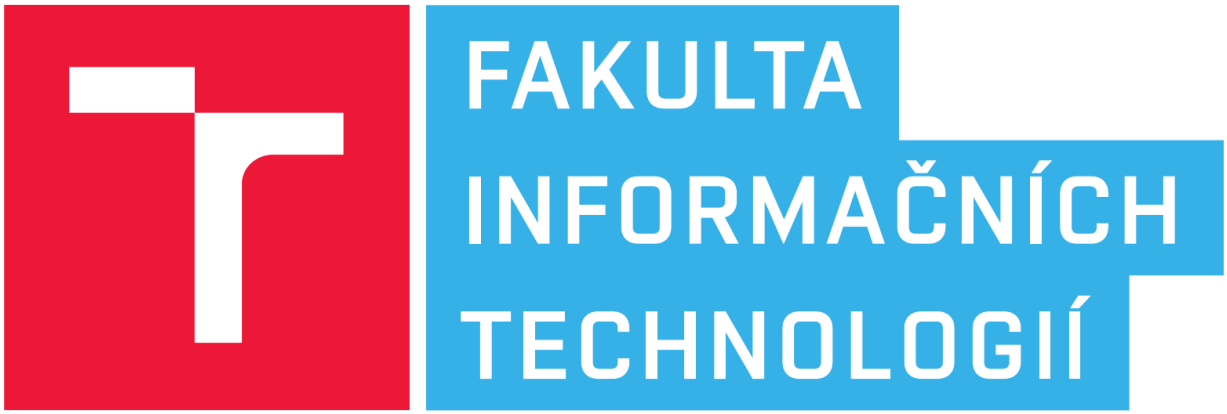
Vysoke Ucení Technicke v Brne



Network Applications and Network Administration

Analýza BitTorrent služby KickassTorrents

22. november 2015

Author: **Vojtech Mašek**, xmasek15@stud.fit.vutbr.cz

1 Description and Application design

There are three basic use cases how application can be used, one is one is using passed torrent file, that file is decoded using bencode and parsed by own torrent parser, then announces are iterated and trackers one by one being requested for list of current active peers connected to the tracker, downloading or seeding this particular torrent. If everything goes well, there should be a response from the trackers. Every response is again parsed an peers IP addresses an application ports are appended to the list of all peers data, at the end of peers gathering, list of peers data is written to the file named in format *[torrent_hash].peerlist*.

Second use case is XML file used as input. That file contains pre-downloaded, structured information about one or more torrents. XML file is opened an parsed, the human readable text file is created and using information located in imputed feed, first of torrents is downloaded and process continues almost the same way as the first use-case, except the file name of this torrent is now taken from the feed.

Last use-case is based on getting RSS feed from the URL of server. This feed is downloaded and then process continues exactly like the second one.

2 Implementation

In the implementation requirements were two choices for the programming language. One was C++ and the other was Python. Because of my previous experience of school project dealing with sockets and TCP communication in Python, I've decided to improve some of my existing solutions for HTTP requests, add support of the UDP request and fit the solution to the needs of this project.

2.1 Torrent parser

After using bencode on torrent file it still needs to be parsed to get all the meta data from it. This data contains information such as tracker announces or torrent info. Both of these information are again produced to prepare them for further needs and usage. Torrent announces are divided to he lists of HTTP ones and UDP ones. Torrent info is rebencoded and after-wise used as the input to the SHA1 hash function that will generate torrent hash from it. That has is again used when connecting to the tracker.

2.2 Tracker connector

Torrent server announces are used to connect to the trackers and get the responses. There can be two types of trackers and this script supports both of them, first one is HTTP tracker that is using standard TCP connection implemented by sockets. Second type is an UDP tracker, this variation is considered as project extension and is also implemented in this script. Basics of the connector used to communicate with the tracker are the same, but for TCP and UDP are ways of handling communication divided and called functions are different.

2.3 Logging

While writing the script, there was a need for it not only to be self documenting, understandable and well commented, but also logs and error messages needed to be done just right. Standard python style logging was good option but nothing so heavy was needed.

When considering test driven development for this project there were a few problems, the biggest was that, especially when testing communication and getting peer lists that are not static but can change from time to time. That and time shortage is why tests were only manual and no automatic unit tests are included. But basic functions that are used for TCP connections have been pre-tested when they were being prepared.

2.4 Downloading files

Files are downloaded using request from urllib module. This request send to server as standard HTTP request containing method, that is in this case GET and headers. As headers are send statements describing how the communication with server will look. This script is trying to look as web browser, without this was sometimes connection refused. Also script is sending some optional headers as for example encoding of wanted response.

Response is than checked. If it's compressed it will be proceed to decompress (gzip compression is supported). If everything seems right, response data is written to file.

2.5 Communication

Script is communicating with the tracker, in simple basic way it sends request and tracker will respond with specific message that needs to be parsed the own for both different way. Also sending the messages to receive that response is a bit different

2.5.1 TCP connection

Before connecting to the TCP tracker, socket needs to be created, after that a messages is prepared. The connection will be based on simple HTTP GET message containing headers and most of all, urlcoded variables that are needed for server to understand the message and then reply with correct and usable response. Urlcoded message will be added as path part of the URI sent to the tracker. It contains some required parameters like hash of the torrent or optional like peer ID. There is a lot more parameters sent and all of them are described in specification of bit torrent protocol. Response from a server needs to be bencoded, after that we can access the

peers in the ordered dictionary returned from bencode. This binary representation of peers need to be processed and prepared. Bin peers parser will take care of that.

2.5.2 UDP connection

Also here before connection is a need to create own socket with small difference in flags. When starting communication we must create UDP connection request containing unique connection and transaction IDs. When server respond, the response is parsed and connection ID is again extracted. Now we can create UDP announce connection request and finally get the response containing encoded peers data in binary representation. That can now be passed to bin peers parser.

2.5.3 TCP and UDP communication graphs

2.5.4 Timeout

When connecting to tracker announce timeout limit must be considered, some connections (especially UDP trackers) will take a long time to respond. Sometimes is too long, way more than 20 seconds and some does not respond at all. After several test and cases I've decided to use 10 seconds as default timeout of the sockets.

This timeout can be easily changed by setting another specific value to global variable `*timeout*` located in isaConnector.py file. For example `*isaConnector.timeout = 2*` for 2 second timeout or `*isaConnector.timeout = None*` for no timeout (Not recommended).

2.6 Binary peers parser

Peers that are returned in ether of TCP or UDP parsed and prepared response, are still not ready to be outputted. Whole peers data is just long field of bytes, where each of six bytes entities represents one peer. First four bytes are IP address and last two are port where peer is communicating.

Every thing is in big endian network convention. After converting every peer one by one to '*address:port*' format, data can be returned and printed to the file.