**TIMCrawler**

***TIMCrawler – Torrent Impatience Monitor Crawler***

***A research about peer abandonment in Bittorent swarms***

**Lior Chikotay, Omer Shwager, Asaf Shamir**

**Workshop Course in Tel Aviv University**

**B.S.c in Computer Science**

**Guided by Professor Hanoch Levi and Hillel Avni**

**Work also guided by Dr Eli Brosh**

# Research Methodology

**Objectives of the research**

Create a tool that will enable gathering statistical information regarding peer abandonment (i.e. departure of a peer before complete download of the file) in BitTorrent swarms.

We wished to study the relations between two global (per swarm) measures on peer abandonment: size of the swarm (or more specifically number of seeders) and the size of the file being downloaded.

We also wished to capture two main individual (per peer) measures: download rate and download progress. We wanted to measure the impact of these two parameters on peer abandonment, in different swarms.

**Main idea**

Implement a BitTorrent client, and modify it in order to get all the relevant information regarding peer abandonment, and the two main measures: download rate and completion rate. A peer is considered to have abandoned when he disconnects from us, and he’s not reachable for a ‘reasonable’ amount of time (i.e for over a certain time period - we can’t reconnect with him, and the tracker doesn’t report on him). The two main measures (peer’s download and completion rate) will be concluded from the initial bitfield (sent by the peer upon handshake), and the ongoing ‘have’ messages received from the peer.

**Our Client**

Our Client implements the BitTorrent protocol, taking into consideration our main objective.

* Never announce itself as a seeder, as some clients don’t send relevant information to seeders (e.g ‘have’ messages). In this matter we implemented a configurable behavior of our Client (regarding when and how to stop itself from becoming a seed) in order to find the optimal recipe. In all the runs in this research we configured the Client to stop accumulating pieces of the file once it had 50% of the file (i.e. keep asking for pieces and playing by the rules, just not saving those pieces, and therefore never become a seeder with all the file).
* Preserve connection with non-seeders peers to whom we were connected at some point ("customer retention"). Meaning that our client continuously tries to connect with disconnected peers in order to indicate abandonment. We found that clients hardly refuse an incoming connection, even if we don’t contribute them much (they might 'choke' us, but then we are still connected and can follow their stats). Our objective was to preserve connection with them, even if we are ‘choked’, as long as they are in the swarm. Whenever a peer (which was not a seeder) has disconnected from our client, we would continue to try and reconnect with it every 3 minutes indefinitely (until the end of the run). Each run in our research had an average duration of 24 hours.
* Connect to a large amount of peers (more than a regular client) in order to make the results distinct.
* ‘Play by the rules’ (i.e. send pieces and follow the protocol) in order to maintain long connection periods.

In order to be able to improve the client’s behavior (with respect to our objective), we used proper configuration (e.g. how many connections, how to avoid becoming a seeder etc.).

**Identifying peer abandonment**

We did post analysis – all the analysis of abandonments etc was made after the runs, by analyzing the data we collected during the runs.

A peer is considered to have abandoned the swarm if all the following hold:

* He is not (or hasn't become) a seeder (in this we also included peers with over 95% completion of the file, assuming these peers might disconnect without informing us of getting the last pieces.
* He disconnected from us, and did not respond to our re-connection tries for a reasonable amount of time. We perform the re-connection tries to all disconnected peers every time we get Announce response from the tracker (every 3 minutes - configurable).
* We defined a reasonable amount of time at which the peer didn't respond to be 30 minutes.

**Event logging**

Our methodology was to log any relevant information for indicating abandonments, as well as all the relevant measures regarding each peer (i.e. peer’s download and completion rate). Indicating a case of abandonment with high certainty is pretty difficult, and therefore we log any information that might help us to do so (all the data collected is specified results sheets). In order to be able to collect lots of data and make the analysis easier, we use a DB to store all relevant events. The basic data blocks to store the events:

* Test - running of a single TIMCrawler. Holds information about the swarm and client’s operation mode.
* Session - a single connection with a peer, starting from the handshake and ending with the peer’s disconnection. Hold information about the peer, session times, peer’s download and completion rate at the end of the session. Note that we can have several sessions with a peer on a single test. In that case, the last session will be taken into consideration when analyzing abandonments.
* Tracker session - information received from the tracker about the peer (the last time we got the peer’s details from the tracker).

# Results of our research

After completing the development of our client, we had conducted a total of 21 successful runs, with an average duration of 24 hours. In these runs we collected data of over 21,000 non-seeder peers (The full results are detailed in the attached excel sheet).

The research has outcome with some very interesting results.

**Abandonment rate by number of seeders in the swarm**

Results show that number of seeders affects significantly the abandonment rate.

Swarms with up to 100 seeders had abandonment rate of 12.4%, while swarms with 100-1000 seeders had 6.9%, and swarms with over 1000 seeders had 1.4%.

This can be explained by the theory that the number of seeders (and in general the size of the swarm) has great affect on the download speed of peers in the swarm, and therefore on their potential abandonment.

**Abandonment rate by file size**

We can see that there is no obvious correlation between the size of the file and the abandonment rate.

According to the data we collected, abandonment rate was very similar in all 3 groups we defined for file sizes (in files under 800 MB the abandonment rate was 5.7%, in files of size 800-2000 MB abandonment rate was 5.3%).

Apparently, the size of the file is not a significant factor in abandonment equation.

**Abandonment rate by swarm size & file size combined**

Here each group is a combination of swarm size and file size.

As we found no correlation between file size and abandonment rate, there is not much point in this analysis as well. Still, we can see here also that there is correlation between swarm size (number of seeders) and the abandonment rate.

**Number of abandoners by completion rate**

We can see that there are many abandonments of peers with very small (or zero) completion rate of the file. However, from completion rate of 5% and above, the number of abandoners seems to have a very mild linear decrease.

A possible explanation for this is that after users start to download a file using Bittorrent they look at the download status in their client (download rate, ETA, peers stats etc.), and decide if to continue with this download or to abort. The fact that there is only a small decrease in abandonments by completion rate after the first 5% of the file, suggest that at this stage the completion rate doesn't have a strong affect on their potential abandonment (there is still some effect). For instance, if a peer must close his bittorrent client because he is leaving home with his PC, then the completion rate at which he'll quit will be random.

**Number of abandoners by download rate**

\*\* in these graphs we only used sessions that were longer than one minute, as very short sessions might have a lot of "noise" (for example, if a peer disconnected from us just before he sent a HAVE message – then this might have a major influence on the deduced download rate for that peer when the session is very short).

We can see the download rate has a distinct affect on the number of abandoners.

Distribution seems to be exponential (for instance, we have 47 abandonments of peers with DL rate up to 10 kb/s, 7 abandonments of peers with DL rate of 100-110 kb/s etc.). We can also see that there is a long tail in this distribution.

It appears that the download rate has a critical significance in predicting peer abandonment.

**Limitations to the research**

* Inaccurate peer download rate - it's calculated by the only way possible which is the number of piece HAVE messages received per time unit times piece size divided by time unit. Since most clients send those HAVE messages in batches there could be inaccuracies, especially for very short sessions.
* A peer might leave the swarm but return to it after a longer time then our run (usually 24 hours), in this case, he will be considered by us to have abandoned the swarm.

# Technological challenges and implementation

**Technological Challenges**

* **How to collect the data**: We needed a way to store as much data as we could about the behavior of the peers in our swarms. Our plan was to gather as much "raw" data as we could in a manner that would later allow us to easily perform different analysis on the data according to what we think is the best technique, and we wanted to be as flexible as we could about it. For that purpose we decided to store the data in a **relational** **database**, so that a big part of our analysis can be performed through the use of SQL queries. We chose **MySQL** DB. We created a DB schema (see database schema section) to represent the table structure we needed. In order to interact with the DB from our Java client we used **JDBC**. In our design, which is **Object Oriented**, we used layers of abstraction in order to keep the code as clean and as versatile as possible. We confined interacting with the DB to the class DBStatsWriter. The stats logging layer, which implements the **observer design pattern**, listens to important events in the client and uses the DB layer to log them. In order to be efficient and save expensive DB calls, it holds an **in-memory data structure** which maps each peer with its current session's stats. When the session ends, or when the test is terminated, all of the in-memory data is then written to the DB.
* **Keeping track of our peers**: We had two challenges in that area: **1**. Connect to as many peers efficiently and quickly; **2**. Once we follow a peer, keep a close track of it in order to know exactly if and when it has abandoned the swarm. In order to handle the first challenge, we had to change the way the original client connected to peers. The original client would get a collection of peers from the tracker and try to connect to them in a synchronous, serial manner. This process took a lot of time and it also lead to low peer connection rates. We changed this process by using a **threadpool** that ran the jobs of connecting to many peers in a **parallel, asynchronous** manner. For the second challenge, we maintained a collection of the peers who have disconnected from us, and we try to reconnect to them (using the threadpool) with every announce interval.
* **Trace logging**: In order to keep track of the client's performance, both online and offline, we used the **SLF4J** logging API for Java. This creates a simple to read and easy to track (by external tools) log of everything happening in the client. We made sure to log any important event to the log.
* **CLI**: In order to control the client (for instance telling it to end the test and conclude the data) and to get online information about its performance we wrapped the client with a **Command Line Interface** module. We used an external package called **NaturalCLI** for that purpose.
* **Configuration File**: All client parameters, both technical and business logic ones are loaded from a configuration file. These include the connection threadpool size, the frequency of announcing to the tracker, maximum concurrent network connections, parameters that influence the client's behavior (see Client Operation Modes section) etc. Nothing is hard-coded, and there's no need to build the code every time we want to change one of these parameters.

**Main components – Detailed**

## Client

### Operation modes

Added an option to run the client in different modes to test where we get the best results (best statistics). The modes are:

* **Normal** - No change in the behaviour of the original client.
* **NeverNotifySeeder** - When reaching a certain completion rate (configurable), stop sending HAVE messages when acquiring a new piece, and stop updating the bitfield sent to other peers upon a successful connection. Continue to download normally until full completion. The purpose is never to be seen as seeders to other peers, in case some clients stop sending HAVE messages to seeders.
* **HalfSeedDropNewPieces** - When reaching a certain completion rate (configurable), ignore incoming pieces but in all other ways continue to act normally - request new pieces, upload etc. The purpose is to forever stay as active as possible to maintain substantial communication with other peers for as long as possible.

### Reconnect to known peers

Constantly reconnect to clients that were connected to us in order to maintain their status.

Announce requests from tracker  
Send announce requests to tracker much more often than regular Clients (configurable). We use this in order to get as much information as possible. Former researches have shown that this behavior is acceptable by the tracker.

Connect to peersUse a threadpool for connecting to new peers received from tracker. This is done to accelerate the peer connection process.

## StatsLogger

* This is a module designed to keep track of all information of other peers which is relevant to our goals.
* This information is logged using StatsWriter module.
* For each peer the Client connects to - create and maintain a “Session” block of data (e.g. time of connection, number of times connected to this peer, download rate, completion rate etc).
* When a peer disconnects - log this session’s data using StatsWriter.
* For each Announce message received from the tracker - log all peers that were announced. The motivation for this: if we get a peer from the tracker some time after we’ve considered it to have abandoned the swarm (because it has disconnected from our client and is not accepting connections at all) - that will mean that this peer didn’t in fact abandon the swarm. This scenario is possible when a peer changed its peerId or ip/port, or if it refuses connections from our client after disconnecting from it (we are not sure yet whether this is a possible scenario because it should first connect anyway on its listening port and then disconnect).

## DBStatsWriter

This is the DB access layer, which writes to the DB statistics received from the StatsLogger. Has ‘intimate’ knowledge about the DB scheme.

## Configuration

Configure common parameters from properties file

## CLI

Basic command line interface that supports: ‘info’ - print an info line about current status of client’s connection; ‘stop’ - stop current test, and properly close all components.

## The Database Schema

Tests table

|  |  |  |
| --- | --- | --- |
|  |  |  |
| + | id | ID of the current test |
|  | crawler\_id | ID assigned to our client |
|  | mode | Operation Mode |
|  | mode\_settings | String representing the specific settings for the Operation Mode. Specifically, the threshold completion rate. |
|  | start\_time | Test start time |
|  | end\_time | Test end time |
|  | info\_hash | unique ID of the swarm - 20 bytes of hash |
|  | total\_size | number of bytes in all torrent’s files |
|  | piece\_size | number of bytes in each piece |
|  | num\_pieces | Total number of pieces in torrent |
|  | num\_seeders | Number of seeders at the time of the first announce |
|  | num\_leechers | Number of leechers at the time of the first announce |

### Sessions table

|  |  |  |
| --- | --- | --- |
|  |  |  |
| \*,+ | fk\_test\_id | Foreign key to tests table |
| + | peer\_ip |  |
| + | peer\_port |  |
| + | peer\_id | Hash value assigned to the peer |
| + | session\_num | The number of the session between the peer and crawler |
|  | start\_time | Time of handshake with the peer |
|  | last\_seen | Time of last message from peer |
|  | last\_seen\_by\_tracker | Relevant only for “session\_num=0”, which is the corresponding tracker data record |
|  | last\_dl\_rate1 | Calculated peer download rate – granularity I |
|  | last\_dl\_rate2 | Calculated peer download rate – granularity II |
|  | last\_dl\_rate3 | Calculated peer download rate – granularity III |
|  | total\_download\_rate | Calculated peer download rate – based on entire session |
|  | completion\_rate | % of the file downloaded by the peer |
|  | peer\_client | String representing the client the peer used |
|  | initial\_bitfield | Bitfield received on handshake |
|  | last\_bitfield | Last constructed bitfield |
|  | bitfield\_recv | Boolean – did we receive a bitfield from the peer |
|  | last\_num\_seeders | Number of seeders at the disconnection of the peer |
|  | last\_num\_leechers | Number of leechers at the disconnection of the peer |

+ - primary key column \* - foreign key column