

BitTorrent Tracker: deliverable 1

Group 01

Irene Díez

Jesus Sesma

1 Architectural design

Figure 1 shows the architectural design of the BitTorrent Tracker. All the tracker swarm members receive UDP multicast requests from the peers; however, just the master answers. The communication between the peers is done via UDP, and the tracker's instances contemplate UDP/JMS communication among them.

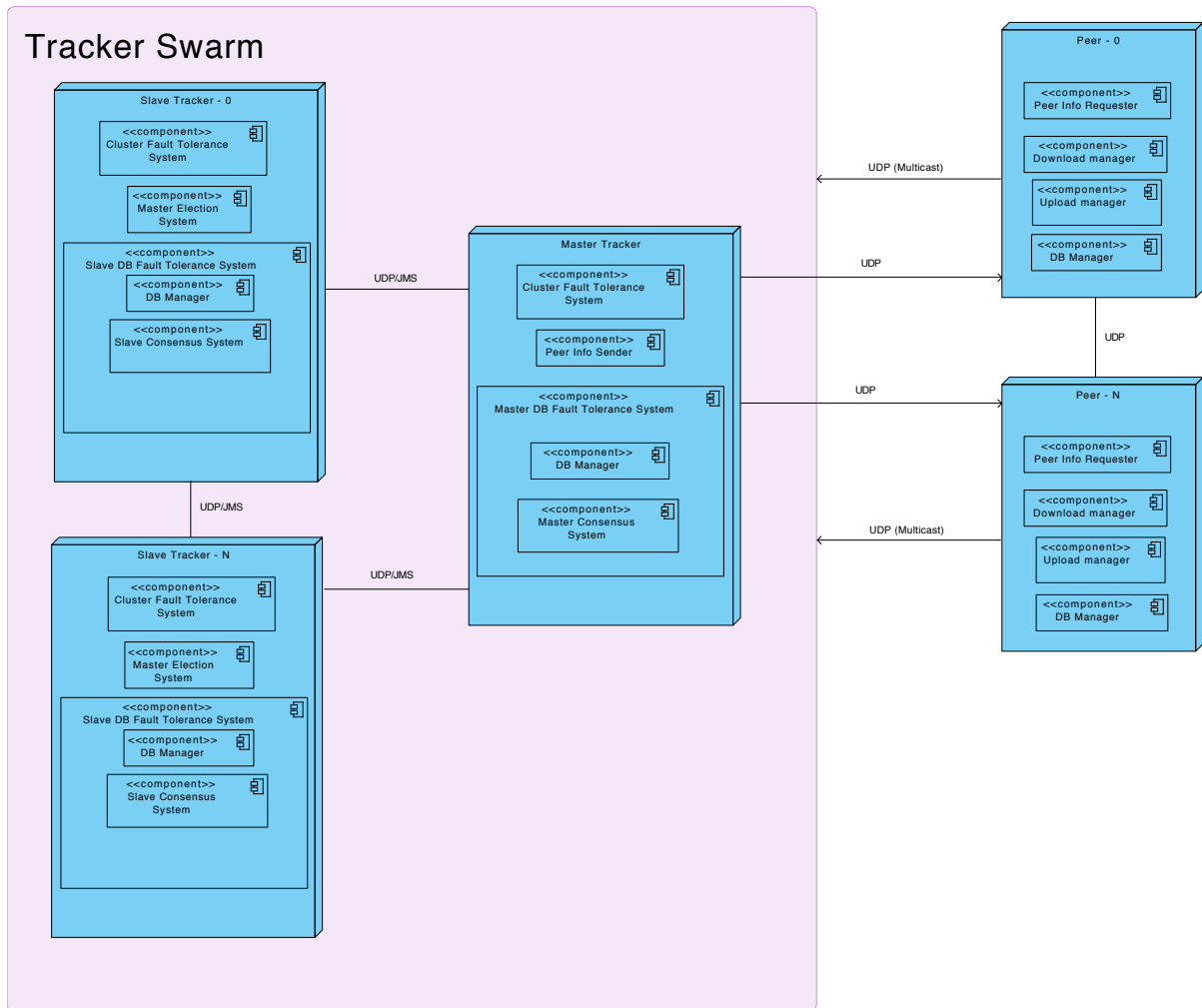


Figure 1: Architectural design of the Tracker.

2 Functionality

The tracker's functionality is summarised in table 1. The following list describes more thoroughly the functions previously shown in table 1 that each component is responsible for:

Identifier	Entity	Description
Peer Info Sender	Master tracker (MT)	Sends information about the peers and the content they have available.
Master DB Fault Tolerance System (M-DFTS)	MT	Composed of the <i>DB Manager</i> and <i>Master Consensus System</i> , ensures that the DB is replicated among all the swarm members.
Master Consensus System	MT	This component manages the DB replication.
Master Election System	Tracker slaves (TS)	Chooses a new master among the slaves when the previous one has fallen.
Slave DB Fault Tolerance System (S-DFTS)	TS	Composed of a <i>DB Manager</i> and a <i>Slave Consensus System</i> , listens to the M-DFTS' orders to ensure the DB replication in the slave.
Slave Consensus System	TS	Ensures the DB replication of the slave.
Peer Info Requester	Peers (P)	Requests information about the peers and the available contents.
Download Manager	P	This component handles the downloads in the clients.
Upload Manager	P	This component handles the uploads in the clients.
Cluster Fault Tolerance System	MT and TS	In charge of sending keep-alive messages among all the members of the swarm.
DB Manager	MT, TS and P	Interface with the DB system.

Table 1: Summary of the functionality implemented in each entity.

- Master tracker
 - *Cluster Fault Tolerance System*: this component is implemented in all the tracker's instances, and it is in charge of knowing the state of all the instances. This component sends and receives Keepalive (KA) messages from all the instances of the cluster. When the master dies sends a notification to the *Master Election System* to start the new master election process.
 - *Peer Info Sender*: answers to a client's information request sending information about the available peers with a specific content.
 - *Master DB Fault Tolerance System*
 - * *DB Manager*: handles the Master-Slave database schema described at Section 3, see figure 2.
 - * *Master Consensus System*: it has two main functions, (i) when a DB transaction needs to be propagated among all the instances, this component is in charge of coordinating such event by waiting for all the instances to be ready, and transmitting the order; (ii) when a new slave is created it sends the necessary information to keep its DB up to date.
- Tracker slaves
 - *Cluster Fault Tolerance System*: refer to the Master Tracker's description of this item.
 - *Master Election System*: it is notified by the *Cluster Fault Tolerance System* about a failure in the Master. This component selects a new master among the available slaves. When activated, and while the Master election process lasts, the tracker will not listen to the clients requests.
 - *Slave DB Fault Tolerance System*
 - * *DB Manager*: refer to the Master Tracker's description of this item.
 - * *Slave Consensus System*: this component has two functions; on the one hand (i) when the *Master Consensus System* requests an operation to be propagated, checks the slave's status and prepares it to do such operation; when the slave is ready, notifies the master, and finally, when the master orders to commit, complies. On the other hand, (ii) when the slave is a brand-new instance, it requests the latest DB information to the master and waits for its instructions to commit.
- Peers
 - *Peer Info Requester*: requests information to the tracker about the available peers with some content.
 - *Download Manager*: manages the download process.
 - *Upload Manager*: manages the upload process.
 - *DB Manager*: handles the Peer database schema described at Section 3, see figure 3.

3 Data schema

Our system contemplates two different database schemas, on the one hand the tracker's master and slaves will implement the schema shown in figure 2. This schema has two tables: (i) `PEER-INFO` where information regarding the peers' host and ports is stored; and (ii) `CONTENTS`, where the tracker will store which peers have available some specific content. Both tables' fields are self-descriptive.

```
PEER-INFO (id:INTEGER, host:VARCHAR(255), port:INTEGER)
          CONTENTS (sha1:STRING(40), peer_id:INTEGER)
```

Figure 2: DB schema for the tracker's master and slaves.

On the other hand, the peers will need to remember the progress they have made during a download; thereby, they will store which chunks they have downloaded so far for a specific file. It must be underlined that our system will always transfer chunks of the same size, with the exception of the last chunk, or when the content's size is inferior to our default chunk size.

This characteristic is implemented using the schema shown in figure 3, with the table `CHUNK`; its fields are self-descriptive.

CHUNK (sha1:STRING(40), offset:INTEGER)

Figure 3: DB schema for the tracker's peers.

3.1 DB technology

We have decided to use SQLite [1] as our storage technology, mainly because we have previous experience with it; but more importantly, because this is a didactic project without the high availability and performance needs that a professional project's database requires.

Regarding the use of a classic SQL database over the NoSQL paradigm, since this project implies the synchronisation of the DB over the trackers, we think that the data should be as normalised as possible and complying to the third normal form (3NF); thereby, we discard NoSQL databases.

4 Interaction model design

The following section describes the interaction model design, we have taken into account the interaction model described at [2, 3], as well as the BitTorrent specification [4, 5, 6] to make our own.

- Tracker-Tracker.

1. Keepalive (KA) messages: are sent every second to all the instances in the tracker swarm. A member is considered down if two followed KA messages are not received.

A KA message has the following structure:

[type, ID]

Where:

- type (32 bits): 0, for KA message.
- ID (160 bits): ID of the instance.

All swarm members must keep and update a table (for now on referred as the *IP-ID* table) with the values: ID-IP-port, that collects information of the ID, IP and port of all the alive instances.

2. Master election (ME) messages: these messages are sent when the master is down (two followed KA messages aren't received). These messages are sent every 3 seconds during the Master Election process. The messages have the following format:

[type, payload]

Where:

- type (32 bits): 1, for ME message.
- payload (160 bits): the minimum tracker ID.

These messages must be sent to all tracker swarm members when the a slave detects that the master is down. The slave chooses the minimum ID among the slaves that appear at its own IP-ID table (taking into account its own ID), and sends its decision to all the swarm members. It must wait to receive ME messages from all the members of its IP-ID table.

3. Hello (HI) messages: these messages are sent when a instance wants to join the tracker swarm. The new instance must begin sending KA messages even though it does not have still an ID. They have the following format:

[type, connection_id]

Where:

- type (32 bits): 2, for HI message.
- connection_id (64 bits): randomly selected by the new instance.

When a HI message is received the master will answer with the following packets:

[type, connection_id, assigned_id, contents_sha, [info_hash, host, port]...,]

Where:

- type (32 bits): 2, for HI message.
- connection_id (64 bits): the same connection_id sent by the new instance.

- `assigned_id` (160 bits): the ID assigned for this instance by the master.
- `contents_sha` (160 bits): SHA-1 of the (`info_hash`, `host`, `port`) triplets.
- `info_hash` (160 bits): SHA-1 of the torrent.
- `host` (32 bits): IP of the peer that has the torrent.
- `port` (16 bits): port of the peer that has the torrent..

The last (`info_hash`, `host`, `port`) triplet will be repeated for each row in the `CONTENTS` table of the master.

When the new tracker slave successfully receives the master's data it will send the following packet:

[`type`, `connection_id`, `assigned_id`, `contents_sha`]

Where:

- `type` (32 bits): 2, for HI message.
- `connection_id` (64 bits): the same `connection_id` sent by the new instance.
- `assigned_id` (160 bits): the ID assigned for this instance by the master.
- `contents_sha` (160 bits): SHA-1 of the (`info_hash`, `host`, `port`) triplets.

4. Database synchronisation (DS) messages: for each peer scraping request listened by the tracker, the slaves must send a *DS-ready* message if they have listened the request, which indicates that they are ready to commit. This packet has the following fields:

[`type`, `connection_id`, `action`, `transaction_id`, [`info_hash`]...]

Where:

- `type` (32 bits): 3 for *DS-ready* message.
- `connection_id` (64 bits), `action` (32 bits), `transaction_id` (32 bits) and `info_hash` (160 bits): are the fields described at *Peer announce*, interaction 1.

When the master has received this packet from all the slaves at its IP-ID table, it will send a *DS-commit* message, that will indicate to the slaves that they must commit the changes. This packet has the following structure:

[`type`, `connection_id`, `action`, `transaction_id`]

Where:

- `type` (32 bits): 4 for *DS-commit* message.
- `connection_id` (64 bits), `action` (32 bits), `transaction_id` (32 bits): are the fields of the previous packet without the variable `info_hashes`.

When the slave successfully commits, it must send a *DS-done* packet:

[`type`, `connection_id`, `action`, `transaction_id`]

Where:

- `type` (32 bits): 5 for *DS-done* message.
- `connection_id` (64 bits), `action` (32 bits), `transaction_id` (32 bits): are the fields of the previous packet without the variable `info_hashes`.

- Tracker-Peer

- Connection request: a peer requests a connection to the tracker to execute a specific action.

1. Peer requests the connection with the following packet (128 bits):

[`connection_id`, `action`, `transaction_id`]

Where:

- * `connection_id` (64 bits): initialised to 0xFEELDEADBAADBEEF.
- * `action` (32 bits): 0, for connection request.
- * `transaction_id` (32 bits): random number generated by the peer.

2. Master tracker answers with one of these packets (128 bits):

- (a) Normal response (128 bits):

[`action`, `transaction_id`, `connection_id`]

Where:

- * `action` (32 bits): 0, for connection request.

- * **transaction_id** (32 bits) the **transaction_id** previously sent by the peer.
 - * **connection_id** (64 bits): the **connection_id** generated by the tracker, which will be used by the peer in future requests. The peer can use the same **connection_id** for one minute, the tracker will accept the **connection_id** for a specific peer for two minutes.
- (b) Error response (≥ 64 bits):
 [action, transaction_id, error_string]
 Where:
- * **action** (32 bits): 3, for error.
 - * **transaction_id** (32 bits) the **transaction_id** previously sent by the peer.
 - * **error_string**: string describing the error, may be left empty.
- This error response is used by the master at every error situation, thereby we will be referring to this packet in future error situations.
- Peer announce: a peer requests info about a torrent.
1. Peer sends the following packet (784 bits):
 [connection_id, action, transaction_id, info_hash, peer_id, downloaded, left, uploaded, event, ip, key, num_want, port]
 Where:
 - * **connection_id** (64 bits): id given by the tracker after successfully establishing a connection.
 - * **action** (32 bits): 1, for announce.
 - * **transaction_id** (32 bits): random number generated by the peer.
 - * **info_hash** (160 bits): SHA-1 of the announcing torrent.
 - * **peer_id** (160 bits): peer's id.
 - * **downloaded** (64 bits): number of bytes downloaded in this session.
 - * **left** (64 bits): number of bytes left to download.
 - * **uploaded** (64 bits): number of bytes uploaded in this session.
 - * **event** (32 bits): status; 0 none, 1 completed, 2 started, 3 stopped.
 - * **ip** (32 bits): peer's IP address.
 - * **key** (32 bits): unique key generated by the peer.
 - * **num_want** (32 bits): maximum number of peers wanted in the reply. A -1 indicates that the peer wants at most the tracker's default value.
 - * **port** (16 bits): the port the peer is listening to.
 2. Master tracker replies with one of these packets:
 - (a) Normal response (160 bits (min), to 4096 bits (max) ($160 \text{ bits} + (48 \text{ bits} * 82 \text{ tuples})$)):

[action, transaction_id, interval, leechers, seeders, (ip, port)...]
 Where:
 - * **action** (32 bits): 1, for announce.
 - * **transaction_id** (32 bits): the **transaction_id** previously sent by the peer.
 - * **interval** (32 bits): minimum number of seconds that the peer must wait before reannouncing itself.
 - * **leechers** (32 bits): number of peers in the swarm still downloading.
 - * **seeders** (32 bits): number of peers in the swarm which are seeding.
 - * **ip** (32 bits): ip of a peer.
 - * **port** (16 bits): listening port of a peer.
 The (ip, port) tuple is optional, it may happen that there aren't any seeders for the torrent.
 Note that the tracker answers by default using the Compact Peer List extension, which uses 6 bytes to represent a peer, as described at BEP 23 [6].
 - (b) If there are errors (≥ 64 bits): The master sends the packet described at *Connection request*, iteration 2b.
- Peer scraping: a peer sends information about a torrent file.

1. Peers sends info (288 bits (min), 3968 bits (max) ($288 \text{ bits} + (160 \text{ bits} * 23 \text{ hashes})$)): `[connection_id, action, transaction_id, [info_hash]...]`
Where:
 - * `connection_id` (64 bits): id given by the tracker after successfully establishing a connection.
 - * `action` (32 bits): 2, for scrape.
 - * `transaction_id` (32 bits): random number generated by the peer.
 - * `info_hash` (160 bits): SHA-1 of the resource (at least one).
2. Master tracker replies:
 - (a) Normal response (160 bits):
`[action, transaction_id, [seeders, complete, leechers]...]`
Where:
 - * `action` (32 bits): 2, for scrape.
 - * `transaction_id` (32 bits): the `transaction_id` previously sent by the peer.
 - * `seeders` (32 bits): number of seeders.
 - * `complete` (32 bits): number of times the torrent has been downloaded.
 - * `leechers` (32 bits): number of leechers.

The triplet (`seeders`, `complete`, `leechers`) must appear at least once, and repeats for each `info_hash` asked in the scraping request.
 - (b) If there are errors (128 bits): The master sends the packet described at *Connection request*, iteration 2b.

5 Failure model design

- Tracker-Tracker

1. Keepalive (KA) messages: there is no failure model for this type of messages.
2. Master Election (ME) messages: depend on the values of the IP-ID table that each slave has. When an election is in progress, each slave must wait until it receives the decisions of all the members of its IP-ID table. If everything goes right, (Situation-A) the slave with the lowest ID is chosen as the new master; however, during the election several things may go wrong.
 - If a slave incorrectly chooses an ID which is not the lowest (according to an accurate IP-ID table), it must listen to what the majority of the votes says (Situation-B). This may happen when two KA messages of the lowest-ID-slave do not reach that slave, and the slave chooses wrongly.
 - If there is not a majority, the election process will continue until Situation-A or Situation-B is reached. Situation-A or Situation-B will be eventually reachable when (i) ME messages are correctly delivered or, (ii) the IP-ID table is updated.
 - If Situation-A is not reached and by Situation-B the majority of the slaves chooses wrongly, a new election will start, since in the IP-ID table the master won't be the one with the lowest ID.

This process will be repeated until consensus is reached.
3. Hello (HI) messages: these messages will be ignored during a Master Election process.

The new instance must wait 3 seconds if no KA messages are listened to elect itself as master, when it elects itself as master its ID will be a random version 4 UUID (which are virtually impossible to be the same in two or more instances). If two or more new instances elect themselves as masters, when the KA messages eventually reach their destinations, a Master Election process will start, since all the instances but one will have wrongly chosen themselves as masters.

When the tracker master answers to a HI message, it will wait 10 seconds until the new slave answers with the confirmation, if no response is received the master will resend the request. This scheme will be repeated until the new slave confirms the transaction, or the slave goes down.

4. Database synchronisation (DS) messages: when the master does not receive a *DS-ready* message within 10 seconds from all the members that it has on its own IP-ID table, it will send again the scraping request originally sent by the peer until the slave sends the *DS-ready* packet or the slave goes down. When this is completed, it will follow the same scheme when the *DS-done* messages are not received.

DB synchronisation is a key issue in the tracker, and thereby it is forced.

The scraping messages are stored in a FIFO structure in all the swarm members, so until transaction N has been done, transaction N + 1 will not start. Thereby, if a master dies the new master chosen after the ME process and the new slaves will be consistent.

- Tracker-Peer: For all the messages of this type, if no response is received after 15 seconds, the peer must resend the request. The peer should follow this scheme until one minute has passed from the first resend. If the tracker has not still answered, the peer must consider it down, and thereby notify the user accordingly.

6 Graphical interface

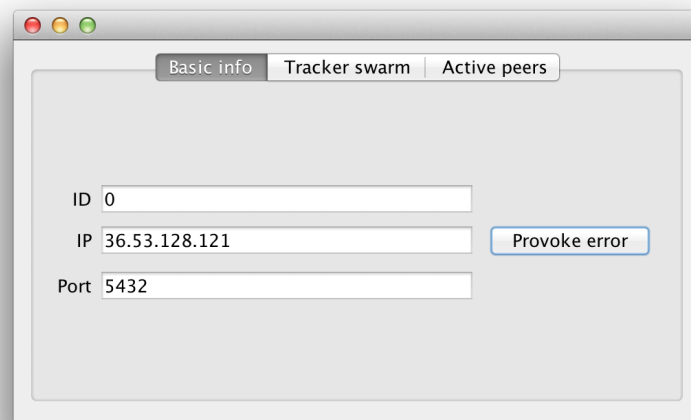
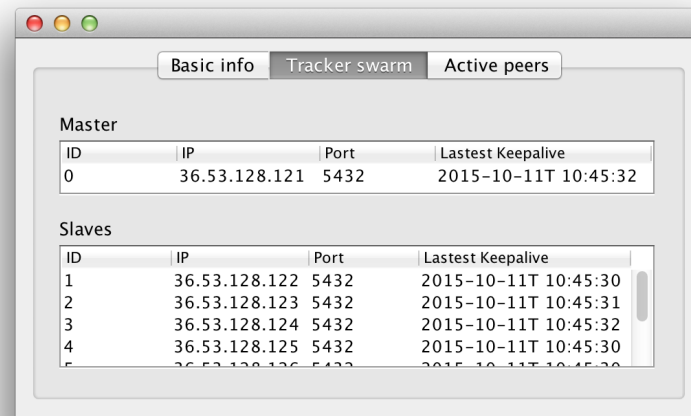


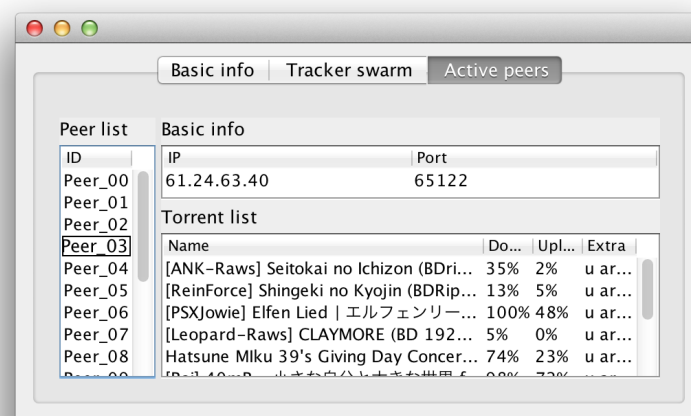
Figure 4: Basic information of the current tracker.



Master			
ID	IP	Port	Lastest Keepalive
0	36.53.128.121	5432	2015-10-11T 10:45:32

Slaves			
ID	IP	Port	Lastest Keepalive
1	36.53.128.122	5432	2015-10-11T 10:45:30
2	36.53.128.123	5432	2015-10-11T 10:45:31
3	36.53.128.124	5432	2015-10-11T 10:45:32
4	36.53.128.125	5432	2015-10-11T 10:45:30
5	36.53.128.126	5432	2015-10-11T 10:45:30

Figure 5: List of the slaves trackers and the master.



Peer list		
ID	IP	Port
Peer_00	61.24.63.40	65122

Torrent list			
Name	Do...	Upl...	Extra
[ANK-Raws] Seitokai no Ichizon (BDri...	35%	2%	u ar...
[ReinForce] Shingeki no Kyojin (BDRip...	13%	5%	u ar...
[PSX]Jowie] Elfen Lied エルフェンリー...	100%	48%	u ar...
[Leopard-Raws] CLAYMORE (BD 192...	5%	0%	u ar...
Hatsune Miku 39's Giving Day Concer...	74%	23%	u ar...

Figure 6: List of active peers' basic information and downloading/uploading torrent list.

References

- [1] SQLite. <https://www.sqlite.org/>.
- [2] Bittorrent udp-tracker protocol extension. http://www.rasterbar.com/products/libtorrent/udp_tracker_protocol.html.
- [3] Bittorrent Protocol Specification v1.0. <https://wiki.theory.org/index.php/BitTorrentSpecification>.
- [4] The BitTorrent Protocol Specification. http://www.bittorrent.org/beps/bep_0003.html.
- [5] UDP Tracker Protocol for BitTorrent. http://www.bittorrent.org/beps/bep_0015.html.
- [6] Tracker Returns Compact Peer Lists. http://www.bittorrent.org/beps/bep_0023.html.