2016/17

Tracker BitTorrent Project – Deliverable 2

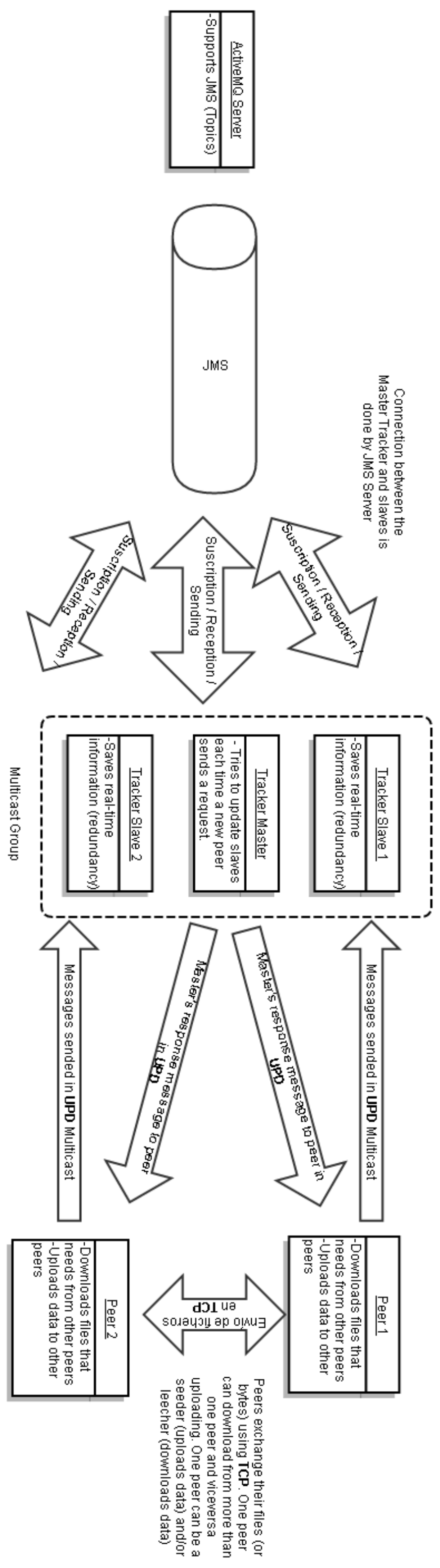
Aitor De Blas Granja / Kevin Cifuentes Salas

Advanced Distributed Systems / Group 004

15-10-2016

Index

Deployment architectonic model



ActiveMQ Server
-Supports JMS (Topics)
JMS
Connection between the Master Tracker and slaves is done by JMS Server
Suscription / Reception / Sending
Suscription / Reception / Sending
Suscription / Reception / Sending
Tracker Slave 1
-Saves real-time information (redundancy)
Tracker Master
- Tries to update slaves each time a new peer sends a request
Tracker Slave 2
-Saves real-time information (redundancy)
Multicast Group
Messages sended in UPD Multicast
Master's response message to peer in UPD
Master's response message to peer in UPD
Messages sended in UPD Multicast
Peer 1
-Downloads files that needs from other peers
-Uploads data to other peers
Envío de ficheros en TCP
Peer 2
-Downloads files that needs from other peers
-Uploads data to other peers
Peers exchange their files (or bytes) using TCP. One peer can download from more than one peer and viceversa uploading. One peer can be a seeder (uploads data) and/or leecher (downloads data)

Description of the functionality of every entity in the domain:

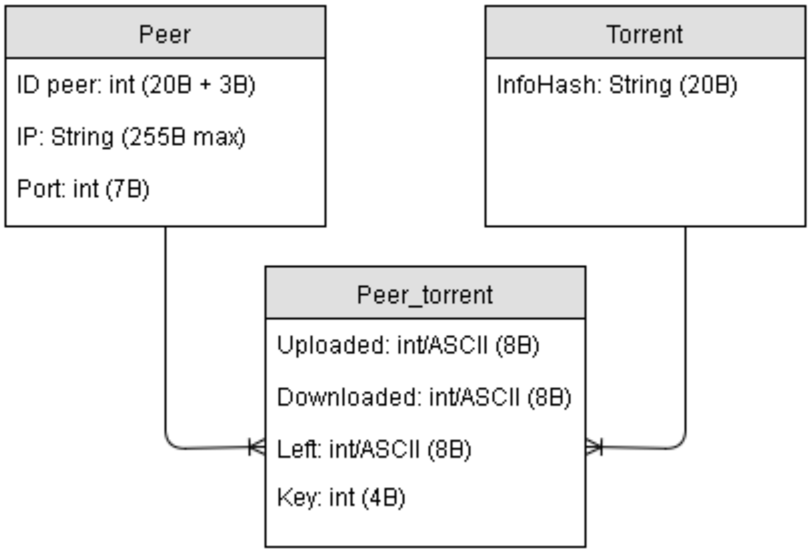
CODE	ENTITY	FUNCTIONALITY
COD_TM_1	Tracker-Master	Receive the request of a peer (along with the files to download and to upload)
COD_TM_2	Tracker-Master	Handle the request of a peer: - Send a ready-to-update message to each and every tracker slave and wait for a response. - Upon the slaves' answer, as long as the average response is OK, the master will update the database and send an order to make the slaves update their own database. If the vast majority of the answers is ERROR, then do nothing. In any of both cases, the master will send the list of peers back to the peer.
COD_TM_3	Tracker-Master	On a new peer request, sends an update message to its slaves to make them update their own database
COD_TM_4	Tracker-Master	- Sends keepalive message to every tracker through JMS - Reception of every keepalive message coming from other trackers.
COD_TM_5	Tracker-Master	When the instance of the tracker is closed, the tracker should delete its database (.sqlite file) before close itself completely.
COD_TM_6	Tracker-Master	- Receives a request from the tracker-slave to send a copy of the database back to the tracker-slave - It checks among the already existing tracker-slaves the request from the new ID (tracker-slave), if that ID is new (not repeated), then, the database is sent back to the tracker-slave confirming at the same time, that the new ID is eventually assigned to that new tracker-slave. If the mentioned ID is repeated, then, it sends a message back to the tracker-slave stating that the ID was already used.
COD_TS_1	Tracker-Slave	- Sends keepalive message to every tracker through JMS - Reception of every keepalive message coming from other trackers. If the one not sending the keepalive message is the master, a new master

		will have to be selected (bully algorithm)
COD_TS_2	Tracker-Slave	- Reception of master's ready-to-update message and delivery of the answer based on whether it is ready or not (OK or ERROR respectively.) - If master's answer is affirmative, the slave will perform the required update in the database. If the answer is negative, the update is cancelled.
COD_TS_3	Tracker-Slave	When the instance of the tracker is closed, the tracker has to delete its database (.sqlite file) before close itself completely.
COD_TS_4	Tracker-Slave	When it comes to decide which ID is assigned to a new tracker-slave when the latter is connected to the JMS service, the following will happen: the tracker-slave will listen all keepalive messages during the next 5 seconds. Within that period of time, the tracker-slave will detect which IDs are free, and it will choose one among the free ones.
COD_TS_5	Tracker-Slave	After the tracker-slave has auto-assigned an ID, it sends a request to the Master in order to obtain a copy of the database.
COD_TS_6	Tracker-Slave	Receives the database bytes sent by the tracker-master.
COD_TS_7	Tracker-Slave	Receives the message from the tracker-master stating that sending the database was not possible and therefore, a new ID has to be selected.
COD_TS_8	Tracker-Slave	With respect to the Bully algorithm, the approach taken in this project towards the decision of coming up with a new tracker-master lies basically in choosing the highest ID among the slaves. Since we have a list of currently active trackers, we know which one is the highest. The latter will wait for a period of time before announcing (broadcasting a message) it is the new Master.
COD_P_1	Peer	Whenever it connects for the first time, the required space is reserved and it sends a message to the tracker (meaning, multicast group) containing the files to be downloaded and uploaded.
COD_P_2	Peer	Reception of tracker's answer and extraction of the information regarding the peers.
COD_P_3	Peer	Based on tracker's answer, it establishes the required connections with the rest of the peers (concurrency)

		which owns the desired files as well as the approval of the downloading request from the rest of the peers. The download will be done in chunks. For every downloaded chunk, every peer will be informed.
COD_P_3	Peer	It sends information periodically to the tracker, thus obtaining the updated list of peers from the tracker.
COD_P_4	Peer	The peer will let the tracker know when the download is finished.

Definition of the data scheme of the trackers

The technology opted towards the information persistency is SQLite. Due to the fact, it does not rely on a client-server architecture, it is a database that can be attached to the project without any issue thanks to its simplicity and popularity.
Every database, table, the data itself, everything is included and integrated within a single file making the management something very fluent, easy and fast.
For the management of the database, we do not leave *Sqliteman* and *DB Browser for SQLite* out of the equation.



Peer	*ID peer: Peer ID* *IP: Peer address* *Port: The port in which the client is listening*
Torrent	*Info_hash: Torrent ID*

The table Peer_torrent represents the relationship between Peer and Torrent: one peer can have n torrents and one torrents can be related with n peers.

Peer_Torrent	*Uploaded: number of bytes uploaded* *Downloaded: number of bytes downloaded* *Left: If the file isn't downloaded, how many bytes are left* *Key: Unique ID for the peer in the torrent* *(Probably appearing: ID_peer and Info_Hash as Foreign Keys and Primary Keys of this table)*

Definition of interaction models between different entities

- **Tracker-Tracker:**
  - ID request from tracker: after waiting for keepalives and choosing a free ID, a tracker slave sends a message to the master for approbation and for requesting the database. Information will be exchange using XML, inside the body of JMS messages.
    ```
    <head>
          <type>IDSelection</type>
    </head>
    <body>
          <id>######</id>
    </body>
    ```
  - Masters positive response to ID request: Information will be exchange using database in bytes, only the type will be specified at the properties part in the JMS message, type “PositiveIDReq”.
  - Masters negative response to ID request: Information will be exchange using XML, inside the body of JMS messages.
    ```
    <head>
          <type>NegativeIDReq</type>
    </head>
    <body>
    </body>
    ```
  - Keepalive message sending between Trackers. Information will be exchange using XML, inside the body of JMS messages.
    ```
    <head>
          <type>keepalive</type>
    </head>
    <body>
          <source>
                <id>#####</id>
                <ip>#.#.#.#</ip>
                <port>####</port>
                <typeTracker> Master / Slave</typeTracker>
          </source>
    </body>
    ```
  - Update request message sending: Information will be exchange using XML, inside the body of JMS messages.
    ```
    <head>
          <type>UpdateRequest</type>
          <updateid>#####</updateid>
    ```

```
</head>
<body>
</body>
```

- Response message from tracker slaves to the tracker master: Information will be exchange using XML, inside the body of JMS messages.

  ```
  <head>
        <type>SlaveResponse</type>
        <updateid>#####</updateid>
  </head>
  <body>
        <slaveID>###############</slaveID>
        <status>OK / ERROR</status>
  </body>
  ```
- Update message sending (update/abort): Information will be exchange using XML, inside the body of JMS messages. Depending on the response at the `<resolution>` element, will continue or stop reading the XML.

  ```
  <head>
        <type>Update</type>
  </head>
  <body>
        <resolution>Update / Abort</resolution>
        <info torrentHash=”########################################”>
              <peerid>######</peerid>
              <ip>#.#.#.#</ip>
              <port>#####</port>
        </info>
  </body>
  ```
- Master selection message: Information will be exchange using XML, inside the body of JMS messages.

  ```
  <head>
        <type>MasterProclamation</type>
  </head>
  <body>
        <id>#####</id>
  </body>
  ```

- **Tracker-Peer**
  - Connection requests from peer to tracker. At first, this data will be sent with the peer’s request:
    - **Info_hash**: urlencoded 20 byte SHA1 hash of the *value* of the info key from metainfo file (.torrent). Will be a bencoded dictionary.
    - **Peer_id**: urlencoded 20 byte string that represents the peer, which is generated at client startup.
    - **Port:** port number where the client is listening on.
    - **Uploaded**: total amount in base ten ASCII.

        - **Downloaded**: total amount downloaded in base ten ASCII.
        - **Left**: number of bytes still has to download in base ten ASCII.
        - **Compact**: can be setted to 1 to request a compact response which replaces the peer list by a string containing peers information with 6 bytes per peer.
        - **No_peer_id**: indicates that the tracker can omit the peer_id in peer list.
        - **Event: *started***
        - **IP (optional):** true ip address of the client machine.
        - **Numwant (optional):** number of peers that the client wants to receive from the tracker. Default number is 50.
        - **Key (optional):** additional identification.
        - **Trackerid (optional):** identifier of the tracker, related with previous announce.
    - Tracker's response to peer's request. At first, this data will be sent with the peer's request:
        - **Failure reason:** if present, then no other keys may be present string.
        - **Warning message (new, optional):** similar to failure reason, but the response still gets processed.
        - **Interval:** time that the client should wait between sending regular messages.
        - **Min interval (optional):** minimum time that the client should wait to reannounce.
        - **Tracker id:** string that the client should send back in next announcements.
        - **Complete:** number of peers with the entire file (seeders). Integer.
        - **Incomplete:** number of non-seeder peers (leechers). Integer.
        - **Peers:** dictionary
            - Peer id: string.
            - Ip: peer address. string.
            - Port: peer's port. integer.
        - **Peers:** binary model (another way to send the dictionary)
    - Peer's state updating messages sent to the Tracker by peer. At first, this data will be sent with the peer's request:
        - **Info_hash**: urlencoded 20 byte SHA1 hash of the *value* of the info key from metainfo file (.torrent). Will be a bencoded dictionary.
        - **Peer_id**: urlencoded 20 byte string that represents the peer, which is generated at client startup.
        - **Port:** port number where the client is listening on.
        - **Uploaded**: total amount in base ten ASCII.
        - **Downloaded**: total amount downloaded in base ten ASCII.
        - **Left**: number of bytes still has to download in base ten ASCII.
        - **Compact**: can be set to 1 to request a compact response which replaces the peer list by a peers string with 6 bytes per peer.
        - **No_peer_id**: indicates that the tracker can omit the peer_id in peer list.

        - **IP (optional):** true ip address of the client machine.
        - **Numwant (optional):** number of peers that the client wants to receive from the tracker. Default number is 50.
        - **Key (optional):** additional identification.
        - **Trackerid (optional):** identifier of the tracker, related with previous announce.
    - Messages sent to complete the download:
        - **Info_hash**: urlencoded 20 byte SHA1 hash of the *value* of the info key from metainfo file (.torrent). Will be a bencoded dictionary.
        - **Peer_id**: urlencoded 20 byte string that represents the peer, which is generated at client startup.
        - **Port:** port number where the client is listening on.
        - **Uploaded**: total amount in base ten ASCII.
        - **Downloaded**: total amount downloaded in base ten ASCII.
        - **Left**: number of bytes still has to download in base ten ASCII.
        - **Compact**: can be setted to 1 to request a compact response which replaces the peer list by a peers string with 6 bytes per peer.
        - **No_peer_id**: indicates that the tracker can omit the peer_id in peer list.
        - **Event: *completed***
        - **IP (optional):** true ip address of the client machine.
        - **Numwant (optional):** number of peers that the client wants to receive from the tracker. Default number is 50.
        - **Key (optional):** additional identification.
        - **Trackerid (optional):** identifier of the tracker, related with previous announce.
- **Peer-Peer**
    - Messages sent to start a new connection with a peer (initial handshake). This is their form: `<pstrlen><pstr><reserved><info_hash><peer_id>`
        - Pstrlen: length of pstr. In this case will be “19”
        - Pstr: string identifier of the protocol. In this case will be “BitTorrent protocol”.
        - Reserved: 8 reserved bytes, which can change the protocol default behaviour.
        - Info_hash: 20 byte SHA1 hash of the metainfo file. The same one that we used in the Tracker request.
        - Peer_id: 20 byte unique string that represents the client.
    - Messages sent during the exchange of data between peers. This is their form: `<length prefix><message ID><payload>`
        - There are different types of message, like we describe as follows:
            - Keep-alive: `<len=0000>`
                - Needed to maintain connection between peers. There is no messageID and no payload.
            - Choke: `<len=0001><id=0>`
                - Fixed length and there is no payload.

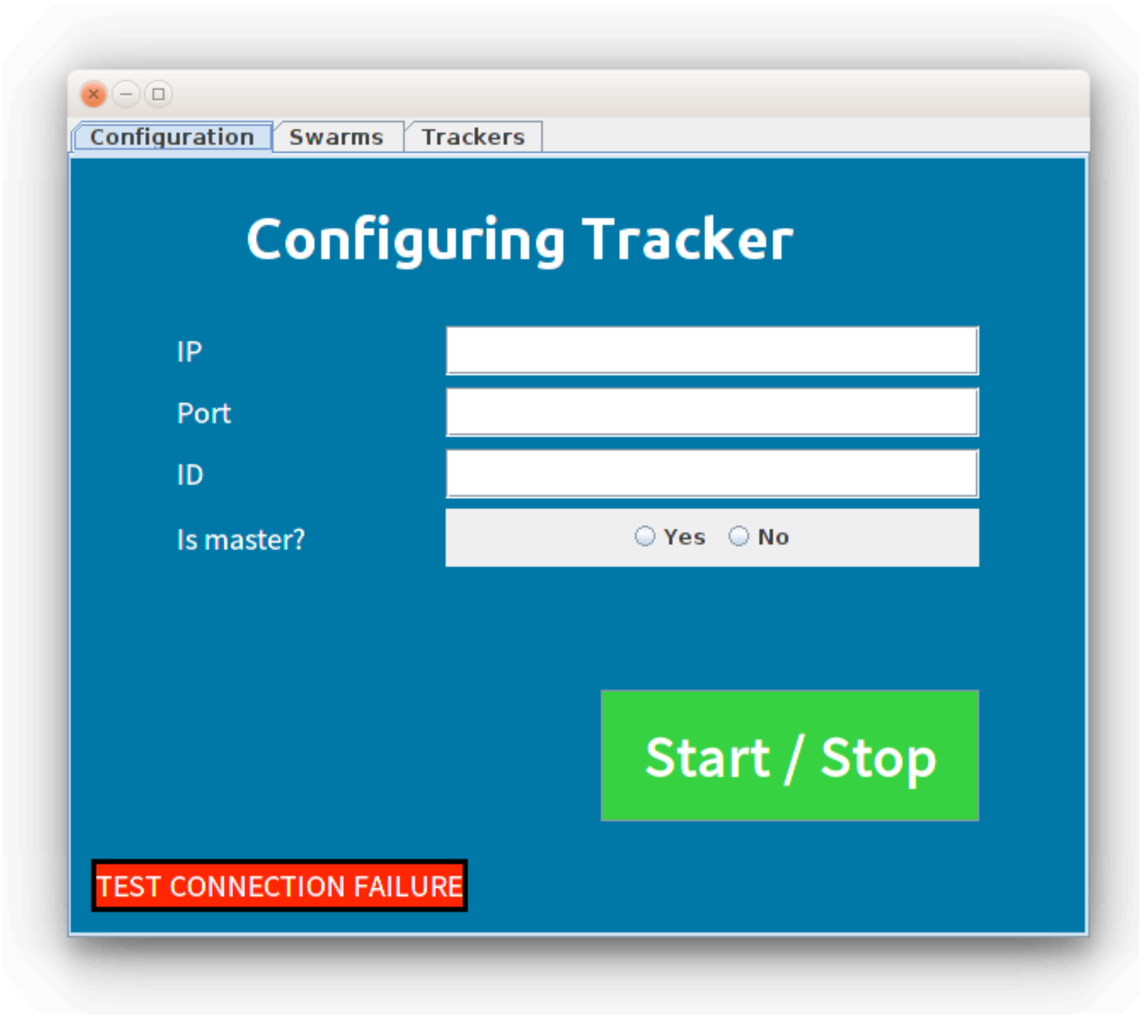
- Unchoke: `<len=0001><id=1>`
    - Fixed length and there is no payload.
- Interested: `<len=0001><id=2>`
    - Fixed length and there is no payload.
- Not interested: `<len=0001><id=3>`
    - Fixed length and there is no payload.
- Have: `<len=0005><id=4><piece index>`
    - Fixed length. Not usually used, due to the fast that peers don’t advertise having a piece to a peer that already has that piece.
- Bitfield: `<len=0001+X><id=5><bitfield>`
    - May only be sent after the handshaking sequence is completed and before any other messages. It is optional. Length is variable and it represents the pieces that have been successfully downloaded.
- Request: `<len=0013><id=6><index><begin><length>`
    - Used to request a block, it is fixed length.
        - Index: integer specifying piece index (starting at 0)
        - Begin: integer specifying byte offset within the piece
        - Length: integer specifying the requested length.
- Piece: `<len=0009+X><id=7><index><begin><block>`
    - Valuable length (x), payload contains:
        - integer specifying piece index (starting at 0)
        - Begin: integer specifying byte offset within the piece.
        - Block: data, subset of the specified piece.
- Cancel: `<len=0013><id=8><index><begin><length>`
    - Fixed length. Used to cancel block requests.
- Port: `<len=0003><id=9><listen-port>`
    - Used with DHT Tracker. Specifies the port in the peer in which the DHT node is listening.

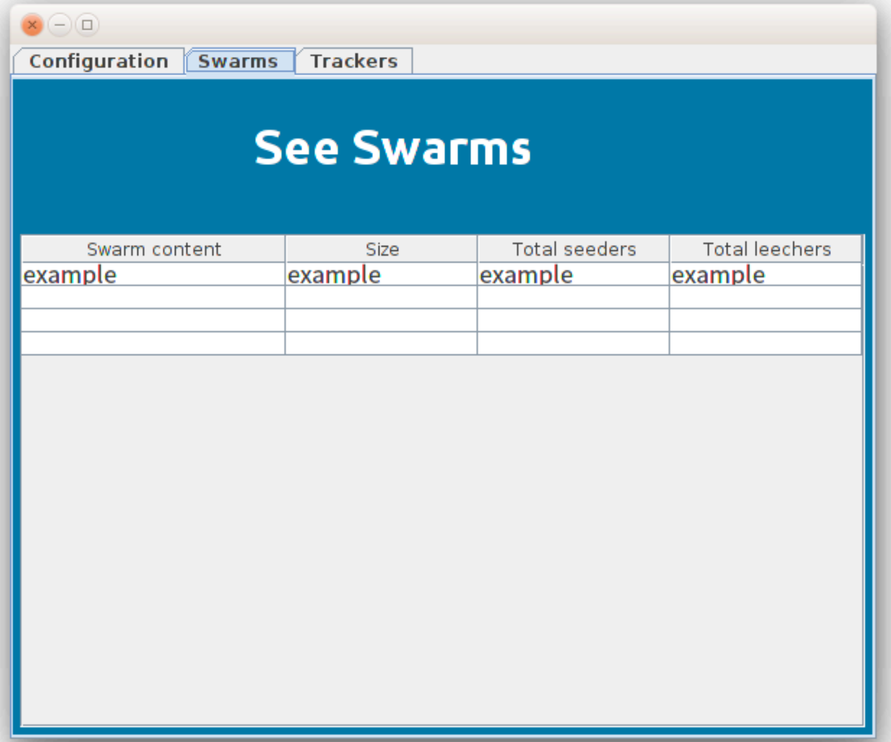
Definition of fault models for each of the entities

- **Tracker**
    - Two slaves connect to the system at the same time, listening to keepalives for choosing an ID. They choose the same ID, sending ID requests to the master.
        - Cause: Two slaves with same ID asking for DB
        - Solution: the first one receives positive response with the DB file, but the second one receives a negative response and it has to wait again to choose another ID.
    - Network error with keepalive:
        - Cause: Internet connection error, not sending keepalive
        - Solution: If the tracker that lost connection is the Master one, we select as Master one of the slaves using a bully algorithm. If the tracker is a slave, we run a new instance to replace it.
    - ActiveMQ:
        - Cause: activemq server goes down due to an issue sending messages (happened to me before).
        - Solution: execute again another instance of activemq server.
    - DB connection error:
        - Cause: connection refused, connection lost, bad request (sentence). Could be related with the non-availability of the tracker slaves when the master sends a DB update request.
        - Solution: try to reconnect after some time and save sql sentence for trying again.
    - Network error with tracker master's update message (not the request, the one specifying the resolution):
        - Cause: tracker master's failure, network error with tracker master's
        - Solution: try to resend the same previous message from the slaves (the one identified by ABORT or UPDATE, ERROR or OK)
    - Network error with trackers update request
        - Cause: tracker master's failure, network error with tracker master's
        - Solution: when a peer request is received by the slaves, a timeout function is launched expecting an update request from the master and it's stopped when they receive it.
- **Peer**
    - Not receiving response from the tracker:
        - Cause: response lost
        - Solution: after waiting a period of time (so there is a timeout), tries to send again the same request.
    - Not receiving keep-alive message from other peer:
        - Cause: peer disconnected, peer not more seeding that file.
        - Solution: close connection with that peer.

Graphical User Interfaces

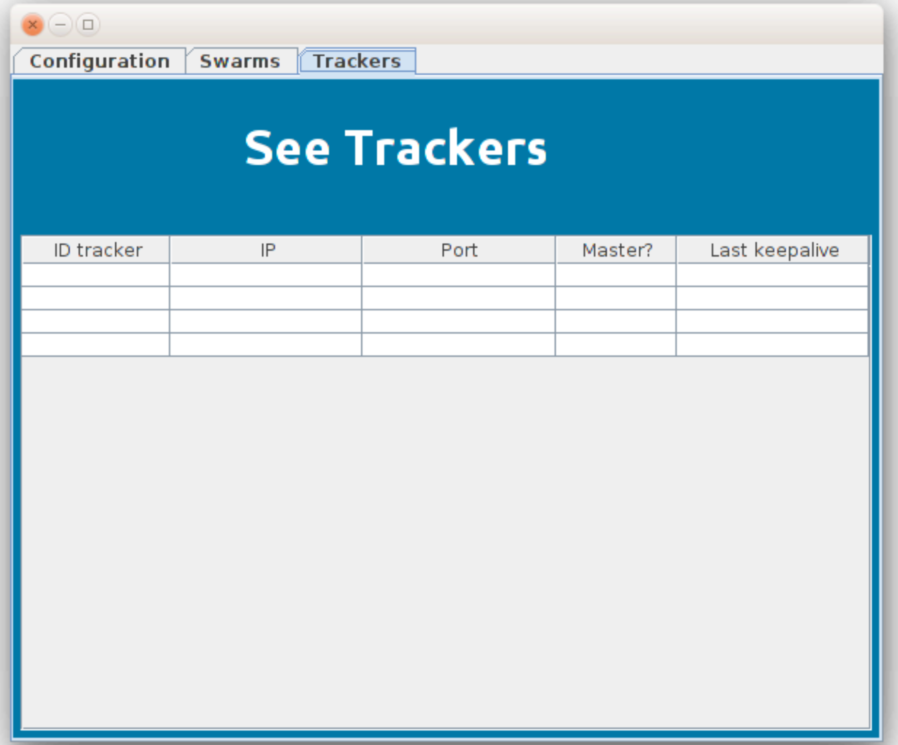
Several inconsistency aspects have been detected when running the user interfaces within different operating systems (iOS, Ubuntu). This gave us a reason to include this section so that the evaluator can objectively assess the real result of the user interfaces.





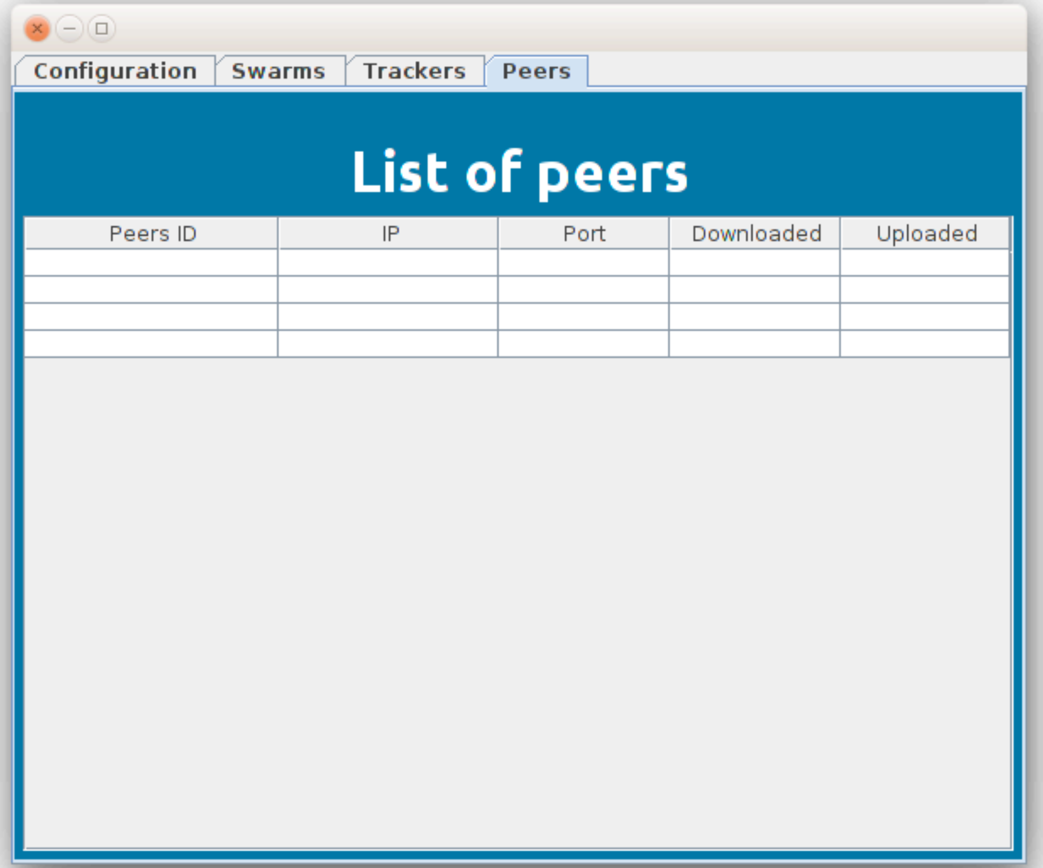
See Swarms

Swarm content	Size	Total seeders	Total leechers
example	example	example	example



See Trackers

ID tracker	IP	Port	Master?	Last keepalive



List of peers

Peers ID	IP	Port	Downloaded	Uploaded