P2P FILE SHARING

‘The Octopus’ Group  
  
Team Members:

AHMAD, FARHAN

ALIPOURSIMAKANI, KAMRAN

ANDERSSON EKSTRÖM, MAX

BERNTSSON, FREDRIK

CHADALAPAKA, GAYATRI

GHASEMI REZAEI, AMIN

IQBAL, NAYYAR

KUKKAPALLI, NAGA VYSHNAVI

NYHLÉN, JESPER

ROUTHU, VENKATA SAI KALYAN

SHAD MANFEAT, SEYEDEH MERSEDEH

ZAREI, KAMBIZ

Type of Document: Acceptance test plan

Version 1.0

Publication Date: May 8th ,2016

**1. PREFACE:**

***Section 2*: Glossary and abbreviations**

**Section 3: Acceptance Test Plan**

**Section 4: References’**

**Release v1.0 on 2016-05-08**

- Initial release

**2: Glossary and abbreviations**

Dark Peer - Is the peer which exceeds the valid time interval assigned by the bootstrap server and thus removed from list of known peers, it will be marked and known as Dark Peer in the network.

Dark Content - Is the files located on any dark peers.

Swarm - The main file which each peer gets from the server at its first connection which includes list of shared files, list of peers and information about which node shares the file and the swarm metadata.

Swarm Metadata - Includes filenames and file message digest

File Metadata - Is the set of headers combined with the filename.

Bootstrap-Server - Server which will inform the other nodes about the presence of any node connected to network and will fetch the swarm metadata from them.

Service: simply is the outcome of chain of functions to reach it designed and defined purpose.

Server: the device which is going to provide the services by using its resources.

Interface: is the presentation of results which is done by the servers based on service, and it is the visible part of a system architecture.

**3: Acceptance Test Plan**

**ATP 1**

|  |  |
| --- | --- |
| Test |  |
| Purpose |  |
| Requirements |  |
| Environment |  |
| Operation |  |
| Expected Result |  |
| Actual Result |  |
| Comment |  |

**ATP 2**

|  |  |
| --- | --- |
| Test |  |
| Purpose |  |
| Requirements |  |
| Environment |  |
| Operation |  |
| Expected Result |  |
| Actual Result |  |
| Comment |  |

**ATP 3**

|  |  |
| --- | --- |
| Test |  |
| Purpose |  |
| Requirements |  |
| Environment |  |
| Operation |  |
| Expected Result |  |
| Actual Result |  |
| Comment |  |

**ATP 4**

|  |  |
| --- | --- |
| Test |  |
| Purpose |  |
| Requirements |  |
| Environment |  |
| Operation |  |
| Expected Result |  |
| Actual Result |  |
| Comment |  |

**ATP 5**

|  |  |
| --- | --- |
| Test |  |
| Purpose |  |
| Requirements |  |
| Environment |  |
| Operation |  |
| Expected Result |  |
| Actual Result |  |
| Comment |  |

**ATP 6**

|  |  |
| --- | --- |
| Test |  |
| Purpose |  |
| Requirements |  |
| Environment |  |
| Operation |  |
| Expected Result |  |
| Actual Result |  |
| Comment |  |

**ATP 7**

|  |  |
| --- | --- |
| Test |  |
| Purpose |  |
| Requirements |  |
| Environment |  |
| Operation |  |
| Expected Result |  |
| Actual Result |  |
| Comment |  |

**4. REFERENCES:**

[1]The company P2P4fun would like you to design a simple yet efficient P2P file sharing application.

The application should mix features from fully-distributed solutions such as Gnutella<https://en.wikipedia.org/wiki/Gnutella> and BitTorrent <https://en.wikipedia.org/wiki/BitTorrent> in order to combine their strengths.

Each P2P peer software will be installed with a preconfigured IP address of a default bootstrap server. Upon start, the peers connect to the bootstrap server. From there they receive a small set (2-3) or randomly selected IP addresses of other peers as well as the IP addresses of additional bootstrap servers. Furthermore, they receive a list of published swarms. A swarm is defined by a shared file, the swarm metadata and a set active peers (IP addresses) sharing that file. In addition, they receive a black list containing IP addresses that have been banned from using the service. This is used as a measure of preventing spam and abuse. Peers must refuse to communicate with banned IP addresses. All IP addresses, with the exception of the default bootstrap server and those on the black list, have a validity time (let's say 3 minutes). Each peer is responsible to refresh the validity time for its IP address (for example, the refresh can be performed every 2 minutes). For each refresh, the expiration date is kicked forward by the corresponding validity time value. IP addresses that exceed the expiration date are removed from the bootstrap server. When all IP addresses from a swarm are discarded, the swarm is removed from the bootstrap server. Since timekeeping is important, all peers and servers should synchronize with a NTP server and use the same timezone (UTC).

The peers periodically (for example, every 5 minutes) retrieve the information from the bootstrap server. If the default bootstrap server is down, they try to connect another bootstrap server. If no bootstrap server is available, they use the current data they have. If a peer did not manage to bootstrap (does not have any data at all) it will present an error message to the user and quit.

After obtaining the data from the bootstrap server, the peer attempts to establish connections (HTTPS is recommended) to the peers received from the bootstrap server. If none of the connections ca be established, the peers waits until the next time they talk to the bootstrap server. At that point they can receive another 2-3 IP addresses for randomly selected peers. When two peers connect, they each select 2-3 known peer IP addresses and exchange them with each other. This exchange help discover dark peers (these are peers that have refrained from refreshing their IP address on the bootstrap servers). Each peer must maintain a maximum of 3 connections to other peers.

The bootstrap servers periodically (for example, every minute) sync their data with each other. Identical data from two servers is merged into a single entry, keeping the newest expiration date. Different data is appended to the list.

A peer can make a set of files available for sharing. For each file a swarm is created. The contents of the file is divided into a sequence of 1024-byte large chunks (blocks). The last block, obviously, can be smaller than 1024 bytes (but not zero). For each block, a message digest (SHA-1, SHA-2 or SHA-3) is computed. The message digest, the block size and sequence index (telling where in the sequence the block belongs) are put into a header, which is prepended to the block. The set of headers together with the file name is the file metadata. A message digest is computed over the metadata as well. The file metadata and its digest (but not the file contents) is the swarm metadata.

If the user marks this as a public swarm, the peer connects to the default bootstrap server (or other bootstrap server if the default one is down) and uploads the swarm metadata. Note that the file contents are not uploaded to the bootstrap server. The bootstrap server will associate the peer's IP address with the swarm metadata.

If a peer finds an interesting swarm on the bootstrap server it can engage into the swarm to download the file. To do that, the peer connects to one of the IP addresses associated with the swarm (there is only one IP address in the beginning) and obtains a list with file chunks available at that IP address. Then it requests a chunk from the file (it's up to you to decide which block is selected). If the peer has the requested block, it begins uploading it to the requesting peer. At the same time it appends the IP address of the requesting peer to the set of IP addresses associated with the metadata. If the requesting peer is not configured to run dark, it sends the swarm metadata to the bootstrap server. Upon reception, the bootstrap server associates the sender's IP address with the swarm. When the peer has completed downloading a file is must recompute the message digest over the data to verify its integrity. If the computed digest does not match the corresponding digest from the file metadata, the application must show an error message to the user and ask if to retry.

At some point another peer may join the swarm. This peer will have access to two peers holding the entire file or parts of it: the swarm creator and the peer from the previous paragraph. The new peer can therefore download one block from each peer. Any additional peer that joins the swarm becomes a potential uploader so that one can download multiple chunks simultaneously.

Some content will be non-public (dark) and thus not available from the bootstrap server. A user can search by filename for such content. A search message is sent over the connections established to peers and these forward it to their peers (you'll need a hop limit for the messages in order to prevent them from circulating in the network forever). A peer that receives the search message checks if the requested filename is used in any swarms that the peer participates in. If found, the peer connects directly to searching peer and sends it the swarm metadata. Multiple replies can arrive at the searching peer. You are free to use just the first one or find some strategy to coalesce them.

Obviously, it is expected that the P2P application must have a friendly graphical user interface (GUI) providing all necessities to create, list, and delete swarms, and to search, upload and download files. In addition, there should a way to select if a peer goes dark or becomes visible again. Progress of uploads and downloads must be shown together with estimated time to complete. The GUI must also show the IP addresses associated with a swarm and the IP addresses of the connected peers. The application must also keep track of upload and download bandwidth and show this as live time-series plots.

P2P4fun insists that all communication must be encrypted. They suggest that certificates should be used for each user as well as for the central server hosting the database. Certificates can be self-signed in the software release to the customer. Encryption features provided by well-known libraries such as OpenSSL/LibreSSL, libcurl or standard Java/Python libraries (for example, HttpsURLConnection, httplib.py, ssl.py) are allowed to be used. It should be possible to restart the system with encryption disabled for debugging purposes.

Finally, P2P4fun requests that all interaction user-to-user and user-to-server be based on a RESTful API with JSON data encoding. Such an API should be easily testable by using HTTP(S) and the command-line utility curl, from the package libcurl.

You must be able to demonstrate that your product supports at least 3 peers. P2P4fun is looking forward to receiving your project proposal!

[2] Tests Document v1.0

[3] Requirements Document v1.1