ID2212 | FISH: FIle SHaring

Tobias Johansson
Marcus Skagerberg

https://www.kth.se/social/course/ID2212/page/project-3-fish-file-sharing-a-distrib/

Table of content

1. Short Task Specification
    Structure One: Centralized Server
    Structure Two: Decentralised P2P System
2. Build Environment
3. Libraries
4. Implementation
    Structure One (centralized)
    Structure Two (P2P)
5. Running the applications
    Structure One
    Structure Two
6. Source code
    Structure One
    Structure Two

1. Short Task Specification

Structure One: Centralized Server

A server keeps track of clients and what files they are sharing. Clients that want to share and join the FISH network sends a register request to the server along with what files the client want to share. Clients in the FISH network can query the server for a search to see if any peer in the network has a specific file. The server replies with a list of clients that are sharing that file. The searching client can then choose to download a file from the list of clients sharing the file.

Structure Two: Decentralised P2P System

As in structure one but without a centralized server. All clients are peers and members in a P2P network. A client can join the P2P network by asking one of the members for information about all the members. The member replies with a list of known members that the joining client can use to contact to let them add the new client to their list of known clients. Each client possesses a directory listing the files the peer is sharing. To search for a file a peer sends a search request to all pers in the network. The searching client can then choose to download a file from a client sharing the file.

2. Build Environment

The build environment for both developers was a Maven project in IntelliJ under OS X. GitHub was used for version control.

3. Libraries

The project used Java SE Development Kit (JDK) 8 for its implementation. No other programming technologies or external libraries were used.

4. Implementation

For both implementations *request–reply* over TCP/IP with sockets was the only used method for the communication. Further was the classes `java.io.ObjectOutputStream` and `java.io.ObjectInputStream` the only ones used for message passing over sockets. In both implementations a classed called `Message` was used to implement a request-reply protocol. The class `Message` contains of two parts; a header containing a message descriptor and a body for data transfer.

For both Structure One and Two you can search for a part of a filename is gets the result of all matching files. The search request for “ka” could looks as the following:

```
Request: ka
Available at:
1. 127.0.0.1:62837/ka
2. 127.0.0.1:62837/kat
3. 127.0.0.1:62837/katt
4. 127.0.0.1:54244/ka
5. 127.0.0.1:54244/kat
6. 127.0.0.1:54244/katt
Download from (0 = none): [user choice here]
```

Structure One (centralized)

A server keeps track of clients and what files they are sharing with a `HashMap<Socket, ArrayList<String>>`. Clients that want to share and join the FISH network sends a register request to the server along with what files the client want to share. Clients in the

FISH network can query the server for a search to see if any peer in the network has a specific file. The server replies with a list of clients that are sharing that file. A client have to send an update request to the server if new files have been added. Otherwise the FISH network will not know that the client's shared files list have been changed.

The connection to the server and if a client's files are available on the FISH network depends on a continual socket connection between server and each client. If the connection is dropped the client is automatically unregistered from the network.

Protocol overview

Descriptor:	*Details:*
REGISTER	A client registers by sending a Message object with descriptor REGISTER, and content `ArrayList<Strings>` of its shared files. If request was successful, the server will respond with a message object with descriptor RESPONSE_OK.
REGISTER_OK	Reply to REGISTER, content `null`
SEARCH	A client can query the server for a search to see if any peer in the network has the file by sending a Message object with descriptor SEARCH and data String filename.
SEARCH_RESULT	Reply to SEARCH, content `ArrayList<FileAddress>` where `FileAddress` contains client IP, port and file name.
UNREGISTER	A client can send to the server to be unregistered (file list removed) and disconnected.
UNREGISTER_OK	Reply to UNREGISTER content `null`
FETCH_FILE	A client can send a request to another request to fetch a file. The content of this message is the file name. The response is a byte array containing the file data.
UPDATE	A client can sends an update of its shared files with the same content as REGISTER.
UPDATE_OK	Reply to UPDATE

Structure Two (P2P)

In this implementation each client runs a Group Membership Service (GMS) that handles the maintenance of a set of addresses to each other group members and handles request from other clients. The GMS also works as an abstraction layer for the communication where a broadcast to all other group members only needs a `Message`.

On join a client request a set of all other group members from *any* group member, all clients are equal in that case. After this the joining client broadcasts to the group and notify that it exist and the receivers of the broadcast adds the client the their respective set. To handle clients that leaves the network; requests during broadcast that fails triggers that the client is removed from the broadcasting clients member set.

The protocol for this implementation has two response descriptors that is OK and ERROR the rest is different requests.

Protocol

Descriptor:	*Description:*
OK	Reply to all requests
ERROR	Reply to invalid request
MEMBERS	A client can request a set of an other group members registered clients. Content is `null`.
ADD	A client can request to be added to an other group members set of of group members. Content is `null`.
SEARCH	A client can request to get a list of the files that match a search query. Content is the search query.
FETCH	A client can request to get a file from another client. Content is the filename, response content an array of bytes containing of the data file.

5. Running the applications

Structure One

To run the application the arguments below should be given to client and server. If no arguments is given default values is given.

Client:

```
java Client [download_path] [shared_file_path] [server_address] [server_port]
```

Server:

```
java Server [server_port]
```

The user interface for the client is as following:

```
1. Search
2. Update
3. Exit
> [console input here]
```

In the choice of Search the user is prompted with what to search for if files are found the user is prompted which one to download, or if not. In choice of Update an update file list is sent to the server.

Structure Two

To run the application the arguments below should be given.

Peer - New P2P network:

```
java Client [download_path] [shared_file_path] [local_port]
```

Peer - Joining Existing P2P network:

```
java Client [download_path] [shared_file_path] [local_port] [member_ip] [member_port]
```

The user interface is the same as in Structure One with the exemption that Update is replace by Member which list the clients current member set.

6. Source code

The source code is publically available at GitHub under the MIT licence in two separate repositories. Links is given below.

Structure One

https://github.com/tobiajo/fish-centralized

Structure Two

https://github.com/tobiajo/fish-p2p