FISH

Emil Tullstedt

January 11, 2017

Contents

1 Task Specification

FISH is a *distributed* file sharing system for local networks using UDP multicast to search for files shared by a FISH-peer in the network. [1] A file is

[1]There is also an implementation where the file searching is done using a centralized server and persistent TCP connections between that server and the peers in the network which is not the subject of this report

shared either individually or by being part of the shared directory for the FISH client and is assumed to be uniquely described by filename within the network.

When a file is downloaded from another peer in the network, it will be stored in a specified download destination directory (by default `/tmp`). If the specified download destination directory is equivalent (either by being the same as or a link to) to the shared directory a file downloaded through FISH will cause an automatic renewal of files shared from the specific host.

2 Development Environment

2.1 The implementor's hardware *&* software

Computer Lenovo ThinkPad X250 with Intel Core i3-5010U @ 2.10GHz and 8 GB DDR3 RAM

Operating System Fedora 23 running Intel 64 Linux 4.8.13

Java JRE & JDK 64-bit OpenJDK 1.8.0 (Java SE 8)

Build system Gradle 2.5

2.2 Compiling *&* Running

1. Go to the root directory in the *git repository*
2. Make sure the **multicast** repository is the currently checked out repository (by running `git checkout multicast` if necessary)
3. Run `$ gradle build` in order to build the application.
4. Go to *build/classes/main*
5. Run `$ java is.mjuk.fish.Client` <*`file or directory to share`*>

The application is now running and should be able to parse the commands **find**, **destination** and **download** to find a file, get (without any arguments) or set the download destination (default `/tmp`) and download a file to the current download destination.

3 Description of Technical Implemenation

3.1 Overview of Classes

The entire FISH-implementation is packaged in the *is.mjuk.fish* Java package with separation of concerns on a class-level. The source files are available in *<gitroot>/src/main/java/is/mjuk/fish.*

Client	Command-line interface and launcher for the FISH protocol implemenation
DatagramHandler	Handles multicasting between peers
Downloader	Thread which download requested files in the background
Helpers	Static functions and constants that can be used for convenience
PeerListener	Open a ServerSocket that sends file per request
UnicastListener	Listens to the peer sharing for information about searches

There is also a single interface `ConnectorInterface` which is currently used only by `DatagramHandler`. The `ConnectorInterface` defines a bare minimum for a class running on one or more threads which communicates over one or more sockets.

3.2 UDP Multicast for File Discovery

A peer in the network may ask for files over the UDP Multicast channel *239.10.10.10*. If a peer (including the original peer) finds the specified file amongst their shared files, they will respond to the original query using unicast.

The peer who queried for the file may proceed to ignore the information, display it to the user or open a TCP-connection to a selected peer in order to download the file to it's own destination directory. The implemenation assumes the user never wants to ask for files for the purpose of ignoring the replies, although that might be abused by a malicious client which is purposefully trying to incite a DDoS of a network if FISH were widely deployed within a network or in a more legitimate use-case used by a client which has successfully downloaded a file and isn't interested by more file listings.

3.3 Network Packages

All communication is serialized over the network as UTF-8 byte packages except for sending the files in themselves (which are transmitted as-is in 4096 byte chunks)

3.3.1 Requesting a file listing

A request for a listing of available sites where a file may be found is sent over the UDP Multicast channel in the format `FIND <unicast port> <file name (may contain spaces)>`.

3.3.2 Informing about the existance of a file

A response to a file listing request takes the form `<TCP Peer port> <file name (may contain space)>` and is sent to the UDP unicast port given in a file listing request.

3.3.3 Requesting a file download

The requester opens a connection to the requestees TCP peer port followed by `DOWNLOAD <file name (may contain spaces)>`. The requestee responds by sending a 5 bytes status message as described below

`E_DNF` Did not find the file (and won't send)

`K_SEN` Found the file. Proceeding to send it to requester

If an error occurred (status message beginning with `E` in case of future expansion of the FISH protocol) the connection is terminated and the requester is assumed to proceed to the next possessor of the file (if possible) [2] and not contact the requestee anymore for the same file request. *Note:* The requester should not assume repeated failure if the user initializes a new request and should thus ask the requestee again if so happens.

If the file is successfully found and transmission begins (as indicated by `K_SEN`) the file is sent over the same established TCP socket in incremental

[2] I.e. the file is available from more peers. The requester should request the file from the next available peer until successful and this is how my implementation is handling this

chunks of 4096 bytes (statically assigned value) until done. This operation is assumed successful if the connection is gracefully closed.

3.4 User Interface

The user communicates with FISH using the FISH Client which is a simple command-line interface which accepts **find**, **destination** and **download** as described in section 2.2.

destination	Display current download destination
destination <path>	Set a new download destination
download <file>	Send a file listing request for the specified file and download it automatically
exit	Close all connections gracefully and die
find <file>	Send a file listing request for the specified file

Whenever the client receives the response to a file listing request it displays the filename, hostname and TCP port to the user (whom may then proceed to manually download the file using netcat or a similar tool, by simply stripping the first five bytes of the resulting file).

4 Running the Application

See section 2.2 "Compiling *&* Running" and section 3.4 "User Interface" for instructions on how to compile and run the application.

5 Documentation

The implementation's documentation is availabe in Javadoc-form on

`https://sakjur.mjuk.is/id2212`