REPORT 5: PEER-TO-PEER FILE SHARING PROGRAM

Department of Mathematical Sciences
Computer Science Division
University of Stellenbosch
7600 Stellenbosch
October 2022

1.INTRODUCTION

This report aims to highlight the functionalities and implementation of the peer-to-peer file sharing program using java, that allows users to access media files from other connected users and uses peer-to-peer software program to search through those media files and be able to download them in their computers. This peer-to-peer file sharing program is enhanced with some security algorithm to ensure that connected clients can download files from another client at their approval. Our programs aim to achieve these requirements for peer-to-peer file sharing.

2.GROUP MEMBERS

[Takunda, Charles, Mudima, 23969156]
[Ben, Sibusiso, Mahlangu, 23629428]

3.OVERVIEW

In our implementation of the peer-to-peer file sharing program, clients can send and receive files from each other, in addition to that, caters for file searching on the current available directories that they submitted and can download the files they need. The application caters for more than two clients at a time, and for optimality, client(s) uses ports and get assigned peerid to ensure that the files that a peer wants to download is sent to the relevant peer. The client has a pop-up GUI that, notifies the users that the server is ready for clients to connect and guide clients on the next steps to do as indicated in (**7**) below. The application is comprised of five java files namely:

1. Client.java
2. Server.java
3. ServerDownload.java
4. Engine.java
5. Fileinfo.java

4.UNIMPLEMENTED FEATURES

1. Security on shared files
2. Dual stream connection
3. Pause/resume functionality

5.IMPLEMENTED FEATURES

1. Sending and receiving of files
2. Upload and downloads files from any connected devices
3. A minimal Client GUI (partially)
4. Searching of files using substrings or full word
5. Single stream connection
6. Progress indicator

6.DESCRIPTION OF FILES

- **Client.java:** Implements the sending and receiving of files
- **Engine.**java: used to ask the user if wants to join as client or server
- **Server.java:** Accepts file search request from clients, if available, notifies the relevant client that the file in search is available to be downloaded.
- **ServerDownload.java**: reads in the files that is submitted using TCP and sends it to the relevant client.
- **Fileinfo.**java: Contains the structure of the files

7.COMPILATION OF FILES

1. Open your terminal and run javac *.java
2. Run Engine
3. Select 1 to run the Server and 2 to run the client
4. Server must be run first, so select 1
5. Run Engine again
6. Select 2 to allow a client to establish a connection with the Server
7. Enter your username as indicated by the pop-up prompt
8. Submit your directory of files that you want to be accessed by other joined clients
9. Enter your port (any as long is available) and peerid number, if you joined first your period is 1, then increments as more clients join in
10. A pop-up prompt will allow you to search your file
11. If available, the server will indicate the port and peerid that file belongs to
12. Type the name of the file you want to download
13. Type the port and peerid number where the file you want to download is at
14. The file is then downloaded and save in the same directory you submitted

EXPERIMENT 1: THE RELATIONSHIP BETWEEN THE NUMBER OF FILE SENT AND THE RATE AT WHICH THEY ARE RECEIVED/DOWNLOADED

DESCRIPTION

In this experiment we test how the increase in the number of file sent affect the rate at which the client(s) receives/download them

HYPOTHESIS

As the number of files sent increases, the rate at which they are received or downloaded increases also given other factors are kept constant

VARIABLES

1. Dependent variable: The rate of receiving(mb/s)
2. Independent variable: The number of files sent

METHOD

1) Compile and run the program as indicated in **(7).**
2) now as a client upload a file that needs to be sent over and record the rate at which it took another client in the connection to receive the file when done downloading it. (Can use a stopwatch to measure the time but most efficient way we used had a program that measures the rate of transfer.
3) Repeat step 2 but now increase the number of files sent by a client and record the rate

RESULTS

The number of files sent over	The rate of receiving and downloading (mb/s, rounded to the nearest integer)
1	15
5	13
13	10
20	8
40	5

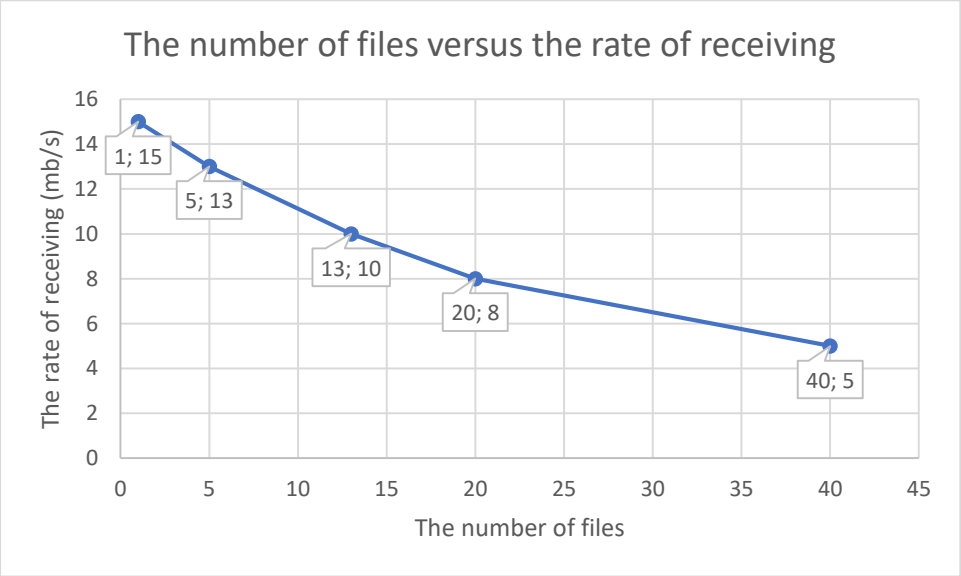

Figure 1: The graph shows the relation between the number of files sent and the rate of receiving them

CONCLUSION

There exists a directly proportional relationship between the rate of receiving and downloading of files and the number of files sent. As the number of files sent over increases, the rate at which a client can receive and download them increases too.

9.ISSUES ENCOUNTERED

- Dual stream connection works over localhost, but fails when using HAMACHI
- The program was able to encrypt files but having some hard time to decrypt the sent files, so we had to take the whole security enforcement on files out. The algorithm we initial chose was Advanced Encryption Standard (AES) algorithm from JCE

10.DATA STRUCTURES USED

- Arraylists for storing the files and usernames
- Object inputstream and outputstream
- Data inputstream and outputstream

11.OVERALL CONCLUSION

We did not find this project that challenging, just that if we allocate our time very well, we would have been able to complete the other unimplemented features and even extra functionality but overall, the knowledge it entitled was worth consuming and we have gained a lot of information on how peer to peer file sharing in a system functions and lastly, our program did achieve some of the requirements stated.