REPORT

Computer Science 313 15 May 2024

File Sharing Program

Group 36 members:

Dylan Hogan (26338319@sun.ac.za)

- Finalised Report.
- Encryption and Decryption.

Joel Lee (23797207@sun.ac.za)

- Set up initial project state and version control.
- Extensive debugging and file updating.
- Wrote the report.

Josef Oosthuizen (26507404@sun.ac.za)

- Created the FXML files.
- Implemented file search, upload and download functionality
- Javadoc and inline comments.

Overview

The aim of this project was to understand peer-to-peer communication between multiple clients and to practice using multiple ports. This report contains the following sections; Features (additional and missing features), file descriptions (all files and their roles in the project), project description (outlines the complete project flow), experiments, issues encountered and design principles.

Features

Additional Features

- File searching is not case sensitive and searches the substring.

Missing Feature

- The Client Activity is not displayed on the Server GUI. Encryption and Decryption had not been implemented properly.

REPORT

File Descriptions

Project Dependencies

1. All .jar files from lib that are necessary for JavaFX applications.
2. Additional maven dependencies to build JavaFX applications.

Server.java
This class represents a server in a chat application. It accepts connections from clients, starts a new thread for each client, and handles the disconnection of clients.

Client.java
This class represents a client in a chat application. It handles the connection to the server, sending and receiving messages, and updating the GUI based on received messages. On startup, the scene MainScene.fxml is loaded.

ClientHandlers.java
This class serves as a thread for each client. It handles the receiving of messages from a client and forwarding them to other clients. The class also contains an ArrayList that keeps track of clients that are connected to the server.

Message.java
This class represents a message in the file sharing application. It contains information about the sender, recipient, type, and content of the message.

MainController.java
This class is a controller for the initial screen of the JavaFX application. It handles events such as entering a username and IP address and clicking the "JOIN SERVER" button.

ChatGUIController.java
This class is a controller for the file downloader screen of the JavaFX application. It handles events such as searching for a file using the “Search” button. The search works with a sub string and displays all matching files in the text area.

This class contains logic for sending messages to all the clients connected to the server. When a user selects the file to download, the client name and file are extracted from a concurrent hash map and are sent back to the client.

MainController.fxml
This is an FXML file that describes the layout of the initial screen. It contains elements such as a TextField for entering a username and IP address, and a Button for joining the server.

ChatGUI.fxml
This is an FXML file that describes the layout of the file sharing screen. It contains elements such as a TextArea for displaying messages, a TextField for searching, and a buttons for searching, downloading and pausing.

REPORT

ServerGUI.fxml
This is an FXML file that describes the layout of the server screen. It contains elements such as a TextArea for displaying messages, a TextField for typing messages, and a Button for sending messages.

Project Description

Initialization: The program starts with the MainController class, which initializes the initial screen of the JavaFX application. The user enters their username and the IP address of the server they want to connect to.

Server Connection: When the user clicks the "JOIN SERVER" button, the btnJoinServerClicked method is triggered. This method creates a new Client object, which establishes a connection to the server at the given IP address.

Client-Server Interaction: Once connected, the Client object starts a new thread that constantly listens for incoming messages from the server.

Message Sending: In A client can send chat GUI, the user can type a message and click the "SEND
MESSAGE" button to send it. The btnSendMessageClicked method in the ChatGuiController class is triggered, which

Message Receiving: The server receives the message and forwards it to all connected clients. Each client receives the message and displays it in their chat GUI.

Private Messaging (peer-to-peer downloads/uploads): In this program a message object is sent over using the “private” type to privately upload the requested file to the target PC. The target PC then downloads the files to its local downloads directory.

File List Updating (Server side): A FileMonitor class is used to check all the files of the connected clients. It works on a separate thread and updates the textarea every five seconds.

Stability and Concurrency: The program is designed to handle multiple clients at the same time. Each client runs on its own thread, ensuring that the program can handle multiple download/upload requests at the same time without blocking. The JavaFX application also runs on its own thread, separate from the client and server threads, to ensure that the GUI remains responsive at all times.

Termination: The program ends when the user decides to leave the program or when the server is shut down. In either case, all connections are closed properly to ensure that no resources are leaked.

REPORT

Experiments

Experiment 1 – Transfer Speed Under Different Network Conditions

Question: How does network bandwidth affect the transfer speed of files using a peer-to-peer file transfer program?

Hypothesis: Increasing the network bandwidth will result in faster file transfer speeds in a peer-to-peer file transfer program.

Experiment:

1. Set up the peer-to-peer file transfer program on multiple devices.
2. Use a network simulation tool to create different bandwidth scenarios (e.g., 10 Mbps, 50 Mbps, 100 Mbps).
3. Transfer a standard file (e.g., 100 MB) between two devices under each bandwidth condition.
4. Measure the transfer speed for each condition.

Dependent Variable: File transfer speed (measured in Mbps).

Independent Variable: Network bandwidth (measured in Mbps).

Analysis: Compare the average transfer speeds obtained under each bandwidth condition. Use statistical analysis (e.g., ANOVA) to determine if the differences in transfer speed are significant.

Conclusion: Conclude whether there is a significant relationship between network bandwidth and file transfer speed, and whether the hypothesis is supported or refuted.

Experiment 2 - Effect of Number of Peers on Transfer Efficiency

Question: How does the number of peers in a network affect the efficiency of file transfers in a peer-to-peer file transfer program?

Hypothesis: Increasing the number of peers will improve the efficiency of file transfers due to better resource utilization.

Experiment:

1. Set up the peer-to-peer file transfer program on a network with varying numbers of peers (e.g., 2, 5, 10, 20).
2. Transfer a standard file (e.g., 100 MB) and record the time taken to complete the transfer.
3. Repeat the transfer multiple times for each number of peers to ensure accuracy.

Dependent Variable: File transfer efficiency (measured as time taken to complete the transfer).

Independent Variable: Number of peers in the network.

Analysis: Analyze the data to see how the time taken for file transfers changes with the number of peers. Use regression analysis to identify trends and relationships.

Conclusion: Determine if the number of peers significantly affects file transfer efficiency and whether the hypothesis is validated.

Experiment 3 - Impact of File Size on Transfer Success Rate

Question: Does the size of the file being transferred affect the success rate of transfers in a peer-to-peer file transfer program?

Hypothesis: Larger file sizes will have a lower transfer success rate due to increased chances of interruptions and errors.

Experiment:

1. Set up the peer-to-peer file transfer program on multiple devices.
2. Select files of varying sizes (e.g., 10 MB, 100 MB, 1 GB, 10 GB).
3. Attempt to transfer each file size multiple times (e.g., 10 attempts per size).
4. Record the success rate of each transfer attempt.

Dependent Variable: Transfer success rate (measured as a percentage of successful transfers).

Independent Variable: File size (measured in MB or GB).

Analysis: Calculate the success rate for each file size and perform statistical analysis (e.g., chi-square test) to see if there is a significant difference in success rates among different file sizes.

Conclusion: Conclude whether file size impacts the success rate of transfers and whether the hypothesis is supported or refuted.

Issues Encountered

- **Username Validation:**
  This problem was persistent for a long time. Originally, we tried to validate on the login button click against a global list of users. We never succeeded implementing this.
- **Hamachi Server:**
  Linux causing many issues.
- **Displaying connected clients on GUI:**
  Tried many different things such as ArrayLists, ObservableLists, TextAreas, ListViews and more. We could not get the global list of activeUsernames to display for all clients on their GUIs.
- **Clients only notified about disconnection when client uses /leave** Unexpected disconnection does not notify other clients.

REPORT

Design

We used our message class for previous hand ins for all server-client and client-client communication. We also created separate FileTransferManager class to handle all file operations.