5/17/2023

Project 4

Secure File Transfer, Group 17

Computer Science 313

Clinton Elves, Jayden Abrahams, Cindi-Ann Jeffery, Langton Chikukwa, Konanani Mathoho

Members

Name	Student Number	Email	Roles
Clinton Elves	24007676	24007676@sun.ac.za	Report, Security
Jayden Abrahams	23559322	23559322@sun.ac.za	File Sharing
Cindi-Ann Jeffery	24695823	24695823@sun.ac.za	Tester
Langton Chikukwa	23607769	23607769@sun.ac.za	GUIs
Konanani Mathoho	24774898	24774898@sun.ac.za	Project leader

Introduction

This project is a Secure file sharing platform that allows users to share files over Transmission Control Protocol (TCP) through the use of Client User Interfaces (CUIs). Communication with messaging and voice note functionality are also available, also implemented over TCP and UDP respectively. Messages are encrypted with the use of keys generated with the RSA encryption method and a hash function. File Transfers are secured via a Diffie-Hellman handshake.

Features

- The option to view the file after it has been downloaded.
- There is Chat functionality (Messages and Voice Note)
- When Hamachi is used, users need not enter an IP address as their IP address will be self-generated.
- The ability to view files uploaded to the shared folder.
- Active user list.
- Drag and drop functionality for uploading with multiple files.

Description of Files

Client.java

The main method creates a thread which calls startUpWindow(). This creates the GUI that prompts the user for a port number, IP address, server port number and username. After receiving the inputs it closes the window and instantiates a client object and calls the listenForMessages() method. This creates a thread which, while the socket is connected, listens for incoming packets. By checking the packet type it decerns the type of data it is carrying and what to do with it. For encryption there are the encryptString() and decryptString() methods for encrypting the messages using RSA keys, and the hasher method for encrypting and decrypting the Keys that are sent over the Network. Before file sharing takes place, a Diffie-Hellman handshake takes place.

ClientHandler.java

In the Constructor, input and output streams are opened, it receives a packet and if it is of type #CONNECTING, it checks for valid username, valid connection. It is also responsible for reordering and updating the active users list. It's main functionality throughout the duration of the running network is to forward packets between client via their respective ClientHandelers. Finally it makes use of a removeUser() method and offTheSystem() method for removing users and closing the input and output streams.

Packet.java

This file makes use of heavy use of polymorphism, having multiple constructors for the packet objects to be instantiated in many way ways depending on the needs of their respective content.

ProgressBar.java
This Class opens a JFrame window to display the JProgressBar of the downloading/uploading of a file. Making use of an ActionListener and ActionEvent for pause and resume functionality.

Search.java
The constructor takes in and sets a global search string and a list of file objects. Search() makes use of HashMaps to sort and analyse the file names. This method also makes use of a function, levenshteinDis() to calculate the distance (how far the search string is from the filename) between strings.

Serializer.java
This Class contains only two functions. One for changing an object into a byte array, toBytes(), and one for changing a byte array into an Object. This class was made simple for ease of utility.

Server.java
The public keys for the Diffie Hellman handshakes are initialised here and are passed on to the ClientHandlers, who intern pass them on to the Clients. The server socket is also created here and is passed to the ClientHandlers. It creates a simple JFrame Graphical User Interface (GUI) to display text describing traffic on the network.

Receiver.java
A receiver is instantiated in client to capture an incoming file over TCP. It instantiates a progress bar that can be used to pause/resume the download.

Sender.java
A sender is instantiated in client and is used to send the file over int byte arrays of a specified chunk size using a DataOutPutStream.

Project Description
The three main classes are the Client, ClientHandler and the Server. The other classes are written to provide assistance for the running of these prementioned three.

Server
Creates a server socket for all the Clients to connect to via the ClientHandlers. A small graphical user interface (GUI) is created for a visual basic history of traffic over the network. Creates public encryption keys Which are also distributed to Clients via ClientHandlers. (See Server.java)

ClientHandler
Listens for Packets to forward to other Clients, and keeps track of the active user list. It also removes clients and closes input/output streams when the network connection is terminated.

Client
Handles input values to initialise the client and opens a more involved GUI for users to interact with. It then calls a listenForMessages method, that Process incoming packets.

Experiments

Experiment1:

Hypothesis:

Client is thread safe, and when multiple clients join a server. The other clients do not disturb the new joined Client sand it does not interrupt them.

Methodologies:

Four clients are connected to the same server, now observer them as they send each other messages and a client send a file to the another one.

Findings:

No visual delay could be detected as clients could simultaneously send messages and download/upload files.

Conclusion:

Our findings agree with the hypothesis. This is a result of the implemented multithreading, as sending, receiving messages and downloading and uploading files, are all happing on their respective thread.

Experiment 2:

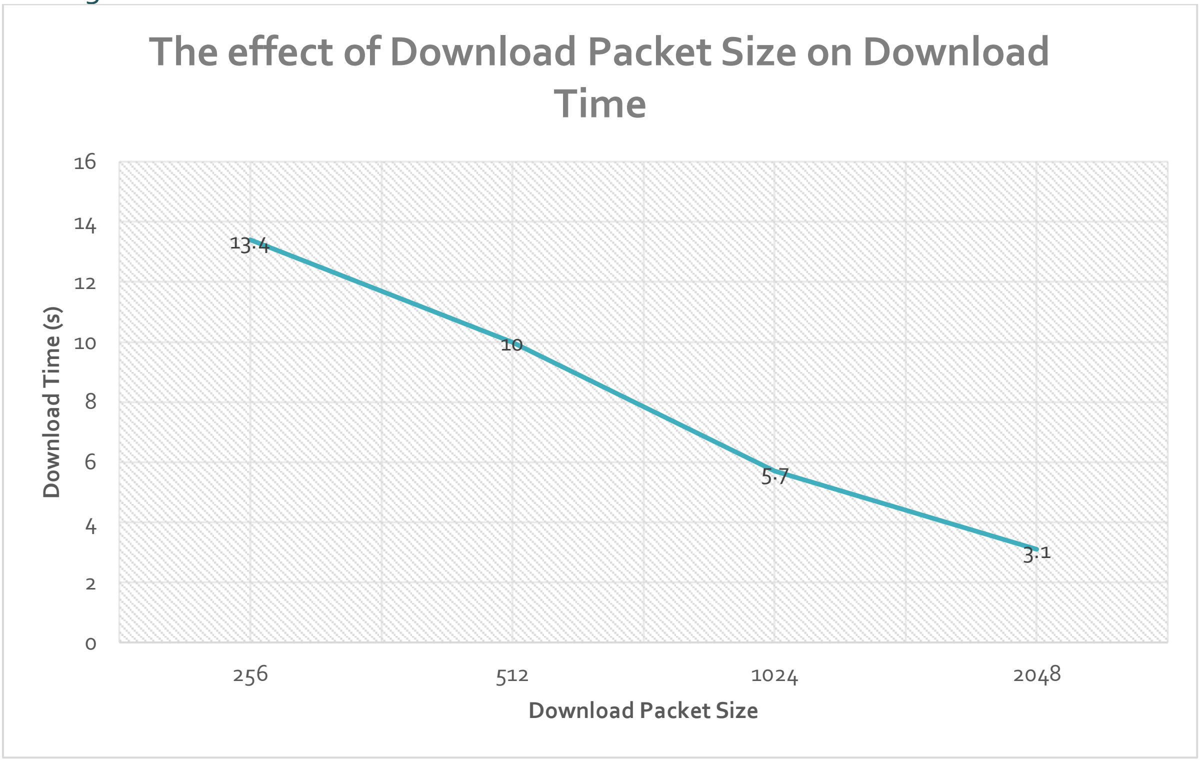
Hypothesis:

Increasing the size of the byte arrays in the packets being sent to deliver portions of the file to the receiver, will cause the download time to decrease.

Methodologies:

Compare download speeds of varied chunk sizes on the same file. Tabulate and analyse further.

Findings:



Analysis:

As can be seen from the graph, Download time does decrease as the Download Packet size is increased. Just by increasing Packet size by a factor of 4, the Download Time decreased by a factor of 4.3.

Conclusion:

The Hypotheses is correct. The analyses proves an inverse correlation between Download Packet Size and Download Speed.

Issues encountered

- When choosing how to encrypt messages, it was decided that a coded RSA class be used. It worked but only for small strings, and it took long. (Solution: use built in Cipher Class)
- Implementing the Diffie-Hellman key exchanges was at first challenging, but after finding the fault where we didn't think to include the changes made in Packets to accommodate keys in the ClientHandler (only the Client). This was quick to remedy.
- For file sharing UDP was used at first, but after problems were encountered with pausing and resuming the downloads. This was alleviated by switching to TCP.

Significant Data Structures

Numerous arrays and arrayLists are used throughout the code for this project. ArrayLists were used for storing clientHandlers and more complex objects as they can dynamically shrink or grow and provide efficient access to elements via index. In addition to this, in the Search class, the search() method makes use of HashMaps to sort and order filenames for comparing to search string.

Design

To make the implementation of more complicated code in Client.java and ClientHandler, the following design decisions were made:

Packet.java

A heavy amount of polymorphism was used via making multiple constructors for all the different cases where packets are made to send different types of data that make varied use of the available fields provided by the class.

Serializer.java

Throughout the project, multiple Input/Output (data/file) streams were used. Having a class to convert objects to bytes and bytes to objects reduced the amount of lines of code and clutter to help with legibility.

Sender.java and Receiver.java

These classes could have been implemented inside the Client class, however, the Client.java file is by far the largest code file and have these two classes separate made them easy to modify

Compilation

To compile the program make use of the make file.

- make – compile and run
- make clean – delete client files and class files.