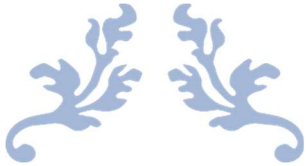


RBUDP FILE TRANSFER PROGRAM

Project 2

AUGUST 19, 2022

Markus Sass (22548890@sun.ac.za)
Chris Langeveldt (23632135@sun.ac.za)

Table of Contents

Overview

The aim of this project was to develop a file transfer program using two different protocols, using TCP and RBUDP. Furthermore, the goal was to understand the trade-offs between the relationship of performance and complexity between the two protocols.

In this document we discuss all required features implemented, unimplemented features, additional features implemented, description of files, program description, experiments, issues encountered, significant data structures, compilation, execution and libraries and packages.

Required Features Implemented

Functionality:

1. Implementing a GUI for both client and server
2. Being able to select the file to send
3. Sending and receiving files through TCP
4. Sending and receiving files through RBUDP
5. Make use of unique sequencing numbers
6. Implementing a progress bar representing file transfer state

Unimplemented Features

- Not correctly handling dropped packets and out of order datagram packets

Additional Features Implemented

- Exquisite GUI
- Preview functionality of viewing images and txt files before downloading
- Having the option to download a file or not
- TCP sending multiple files

Description of files

File_Server.java

This is the receiving side of the file transfer. This file is responsible for the GUI, preview and download of the transferred file.

File_Client.java
This file is responsible for the sending of files through a selection of the file and the choice of protocol to use. This also includes a neat GUI.

ServerListener.java
This class is used to instantiate the unique sequence of numbers sent through the RBUDP protocol.

MyFile.java
This file contains the class for the file object that is being sent from client to server and contains all the relevant information of the file using getters and setters.

Timeout.java
This class is to initiate checks if a socket takes too long to connect.

Program Description

Receiver-side
File_Server.java is responsible for the receiving communication aspect of the file transfer. On start-up, user is prompted regarding the port number as well as the protocol used for receiving, either RBUDP or TCP. A progress bar shows the status of receiving the file. Once received, the file name is displayed. The user can select the file (there can be multiple), and a preview (if of type image or txt) is displayed. The user then has the option of downloading or cancelling. If the user requests a different use of protocol, this can be done by restarting the application and following the prompts.

Sender-side
File_Client.java is responsible for the selection of file, and protocol and the communication aspect of sending of files. Firstly, the user selects the file of choice, then the user has the option of the protocol to use to send the file. The user is then prompted with the IP and port number inputs and then file is transferred. Two buttons are presented on the GUI, each for the different protocols. Some validation and verification are checked to ensure the file's existence and protocol. Only single RBUDP file can be sent, however, multiple TCP files can be transferred.

Experiments

These experiments relate to the features mentioned in the Required Features Implemented section.

Data transfer throughput between TCP and RBUDP:

Hypothesis: All data is transferred completely using both protocols

Variables used: Selected file and its hash code (Independent) and received hash code download from server (dependent)

Method: Transferring a file and comparing the original hash code size and the transferred hash code size for both protocols. Command used: Get-FileHash <filename> -Algorithm MD5

Results:

Original file hash: 96DFA3DA0B621A48B39E78EE17B5E6B4

TCP file received hash: 96DFA3DA0B621A48B39E78EE17B5E6B4

RBUDP file received hash: 96DFA3DA0B621A48B39E78EE17B5E6B4

Conclusion:

Both protocols matched hash outputs of the original file. Hence, we can conclude that the data transfer throughput was successful and in line with our hypothesis.

Transfer rate of RBUDP:

Hypothesis: RBUDP will transfer as a faster rate than TCP

Variables used: Files of different sizes (Independent) and time taken to receive fully (dependent)

Method: Measuring the time taken to send a 150.6kB and then a file of size 283.6kB over both rbudp and tcp.

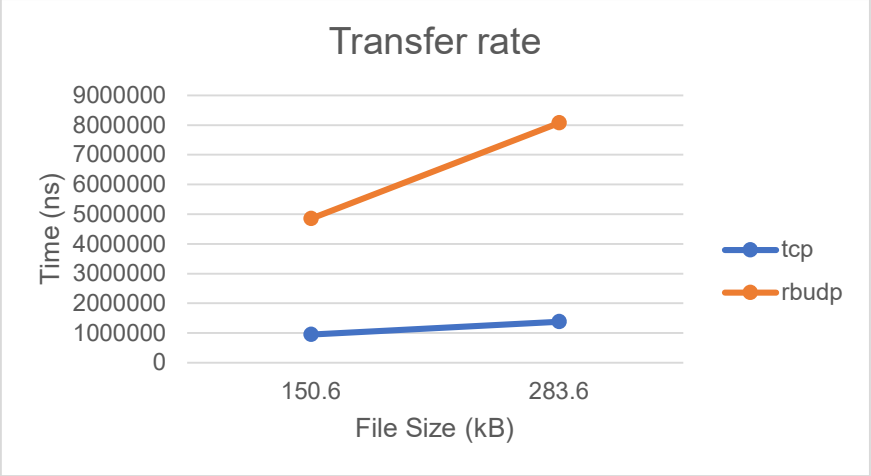
Results:

150.6kB over tcp – 956443 ns

150.6kB over rbudp – 4855825 ns

283.6kB over tcp – 1385100 ns

283.6kB over tcp – 8081356 ns



Conclusion:

For our program tcp was faster. This is not in line with our hypothesis. It might however only be the case for smaller files and also on localhost. It might also be due to inefficiences in our program.

Packet size used in RBUDP:

Hypothesis: A higher packet size will result in faster transfers

Variables used: The same file with different packet sizes (Independent) and time taken to receive (dependent)

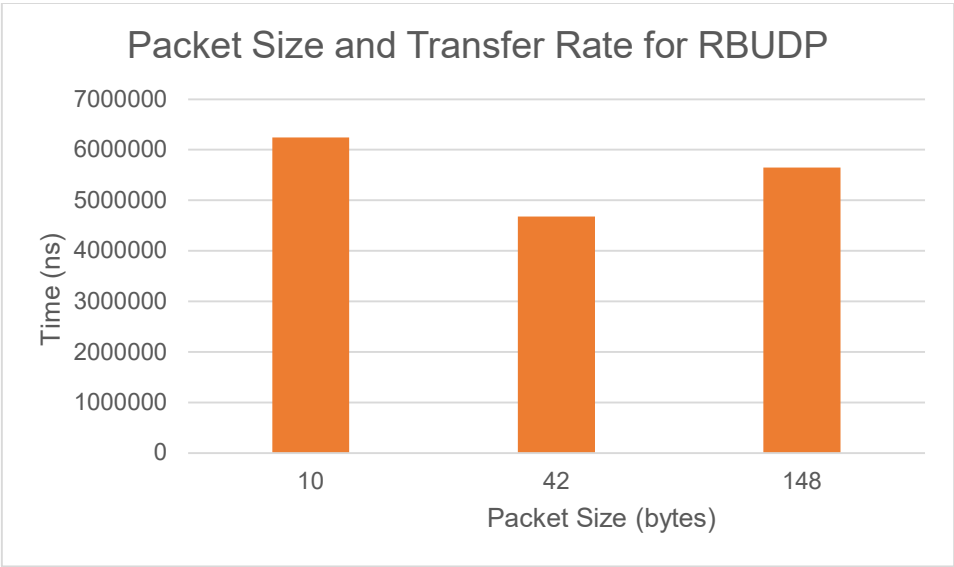
Method: Measuring the time taken to send a file of size 150.6kB with packet sizes 512, 1024 and 2048.

Results:

512 - 5871114 ns

1024 - 4855825 ns

2048 - 6327703 ns



Conclusion:

The hypothesis does not seem completely true. If packet size is too small or too big, speed is decreased.

Blast size used in RBUDP:

Hypothesis: A higher blast size will result in faster transfers

Variables used: different blast sizes (Independent) and time taken to receive (dependent)

Method: Measuring the time taken to send a file of size 150.6kB with packet size 1024 and blast sizes 10, 42 and 148.

Results:

10 - 6243598 ns

42 - 4678321 ns

148 - 5647329 ns

Conclusion:

The hypothesis does not seem completely true. If blast size is too small or too big, speed is decreased.

Varying packet loss rates:

Hypothesis: Varying packet loss rates will be unsuccessful

Variables used: packet loss rates (independent)

Method: Used a random drop in packet comparing with TCP and RBUDP

Results: Failed to successfully produce varying packet loss rates

Conclusion: Although the method was used, the results could not be measured as a result of no retransmission, hence, the hypothesis was correct.

Conclusion from experiments

As mentioned in every point above, we concluded that close to all required functionality works as we thought intended. However, the only surprising part was that for smaller files through localhost, tcp seemed to perform better than rbudp.

Issues Encountered

During the demo, we encountered that RBUDP could not transfer files across the network using Hamachi, however, on localhost is worked 100%, hence we could hypothesize it to Eduroam's network restrictions.

Significant Data Structures

The only notable data structure we have used is an Array list of the files transferred. We also created our own myFIle data structure that contains additional information about the selected file, such as the extension, id, name, and data.

Compilation

Server

javac File_Server.java

Client

javac File_Client.java

Execution

Server

java File_Server

Client
java File_Client

Libraries and Packages

- java.io
- java.net
- java.util
- java.awt
- javax.swing