****

Group 17

File sharer offering TCP or UDP for sharing. In addition to a messaging platform.

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

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

Project 2

Computer Science 313

# Reliable Blast User Datagram Protocol (RBUDP)

|  |  |  |  |
| --- | --- | --- | --- |
| **names** | **Student number** | **email** | **roles** |
| **Clinton Elves** | **24007676** | **24007676@sun.ac.za** | **Project tester** |
| **Jayden Abrahams** | **23559322** | **23559322@sun.ac.za** | **Project leader** |
| **Cindi-Ann Jeffery** | **24695823** | **24695823@sun.ac.za** | **GUI** |
| **Langton Chikukwa** | **23607769** | **23607769@sun.ac.za** | **TCP** |
| **Konanani Mathoho** | **24774898** | **24774898@sun.ac.za** | **UDP** |

### Introduction/Overview

A connectionless reliable protocol was implemented for data transferring, specifically the Reliable Blast Datagram Protocol (RBDUP). It was then compared on a performance bases against Transmission Control Protocol (TCP) which is a connection-oriented protocol.

### Features

In addition to file sharing, a messaging platform is also available to users. A Server Graphical User Interface (GUI) was created to display the connected clients. A directory is created for each client to house files sent to the respective clients. After the file has been received, the user needs only to click on the file in their folder to open it. Similarly like on the Receiver GUI, there is a progress bar on the Sender to indicate the percentage of the file uploaded. The Progress bars are only used for RBUDP.

### Description of files

**Client.java:** Runs on multiple threads. It starts a new thread that will execute a runnable object which will create a GUI window which provides a way a user can interact with the program. It will either run on the Transmission Control Protocol or the Reliable Blast User Datagram Protocol depending on which protocol the user chooses to use.

**Server.java:** It actively listens for incoming connection requests from the client. Whenever a client connects to the server it makes a new thread for which the new client can be run.

**ClientHandler.java:** It implements Runnable, making it run on a separate thread. It handles connections of the clients and the communication of server with the clients.

**Packet.java:** Creates a packet used specifically for signaling. This Packet includes the sequence number and the length of the data that will be in the corresponding packet made using RDTPacket.java which will contain content from the file that is being sent.

**RDTAck.java:** Implements the Serializer class. It creates an acknowledgment packet that will be sent to the sender to show that the packet was received successfully. The packet has the sequence number of the packet being acknowledged. This packet is used in the RBUDP.

**RDTPacket.java:** Implements Serializer. It creates a packet that can be sent over a RBUDP. The packet contains the sequence number, the content of the file being sent and a boolean which shows whether the packet was the last one to be received or not.

**Serializer.java:** It converts an object in memory into a stream of bytes that can be stored on a disk or transmitted over a network. It does this by using two static methods which are **toBytes(Object o)** which converts an object into a byte array and **toObject(****byte[] bytes)** which converts a byte array into an object.

**Receiver.java:** Receives packets from the sender via TCP or the RBUDP. It sends an acknowledgement back to the sender if the file sent by the sender was successfully received.

**Sender.java**: Sends packets to the receiver using TCP or RBUDP. If the packet sent was not successfully received, it will resend it again it knows that the packet was not successfully received because of the use of sequence numbers and acknowledgements attached to the packets used in each Protocol. In TCP, java handles this. For RBUDP, the extra functionality was added in code to ensure safe transmission.

### Project description

This is a java project, used for transferring files through RBUDP and TCP.

**Client.java:** The main method of the class is to make a new thread and execute the run method which creates the startup window by calling the **startUpWindow().**

* **listenForMessages():**  Listens for incoming connections via the TCP socket connection. It reads the packet coming in and it checks the type of the packet by checking the packet’s type field and performs specific actions based on the type. If an exception occurs, it prints the stack trace and closes all necessary input and output streams and sockets.
* **myProgressBar():** Shows the progress of the file being received by the recipient.
* **sendFileTCP() and receiveFileTCP():** Send and receive the file using the TCP protocol.
* **sendFileRBDUP():** Sends a file using the RBDUP protocol.
* **filetoByteArray():** Takes a file as an input parameter and returns a byte array with the contents of the file.

**C**lie**ntHandler.java:** Has a constructor which initializes all the instance variables.

* **forwardPacket():** Forwards the packet to the specified recipient if it is connected to the server.
* **offTheSystem():** Closes all necessary input and output streams and sockets.
* **run():** Receives an incoming packet and sends it in a specific way depending on the packet type using the **forwardPacket()** method.

**Receiver.java:** Has a constructor that initializes the instance variables. It also receives a datagram packet from the sender and deserializes the packet. It adds the packet to an arraylist if it was successfully received and it also increments the number of the packet it is expecting to receive. It creates an acknowledgment packet and sends it to the sender

* **writeFile():** Creates a folder that stores all the files that a certain client has received

**Sender.java:** Has a constructor that initializes the instances variables.

* **Sending():** Creates a packet to be sent and sends it with some probability of loss. It also serializes the packet made using a RDTPacket object. It adds the sent packet into an arraylist of RDTPacket objects. In addition to increasing the sequence number of packets sent.
* **ack():** Creates an acknowledgment packet which must be sent to the sender to indicate that the packet was received successfully when we are using the RBDUP protocol. If the packet is not received, then without the acknowledgment packet, the data packet will be retransmitted.

**Server.java:** The server connection is started, handling all traffic for the existing chat application and the added file sharing feature by giving feedback on how clients are communicating.

**Serializer.java:** Converts the file into a byte stream and is used to handle the conversions to and from byte streams.

**RDTPacket.java:** Creates a packet containing content from the file that is being sent.

**RDTArc.java:** Creates an Acknowledgment, used in case of a need for retransmission.

### Experiments

#### Experiment 1: Varying the packet size

**Hypothesis:** It is expected that as the chunk size is increased, the file will be transferred faster.

**Method:** A JPEG file of 13.2 kB (13,186 bytes) was sent to the receiver using a larger chunk size for every new time it was sent. These speeds were then used to prove or disprove the hypothesis.

**Results:**

|  |  |
| --- | --- |
| **Chunk size for RBUDP** | **Download Time ms (milliseconds)** |
| 2 | 47 |
| 4 | 41 |
| 6 | 40 |
| 8 | 36 |
| 10 | 34 |

**Analysis:** As the chunk size is decreased by 2, the increase in speed is approximately 2.6 ms on average.

**Conclusion:** As stated in the hypothesis, as chunk size increases, so does the speed of transmission. This may stop being true if the amount of data in a chunk starts exceeding the file size. It is recommended that larger chunk sizes be used for transmitting larger files.

#### Experiment 2: Comparing the speed of TCP to UDP

**Hypothesis:** It is expected that RBUDP will be faster than TCP due it having no error correction, and have less strict criteria for retransmission.

**Method:** A PDF file of 29.1 MB (29,076,731 bytes) was sent to a receiver whilst the receiver timed the file download process. The speeds were then compared to prove or disprove the hypothesis.

|  |  |  |
| --- | --- | --- |
| **TCP speed in ms** | **RBUDP speed in ms** | **RBUDP chunk size** |
| 67 | 11819 | 2 |
| 11431 | 4 |
| 11021 | 6 |
| 10663 | 8 |
| 10464 | 10 |

**Analysis:** TCP was faster than the programs implementation of RBUDP by approximately 10000 – 11000 ms. Upon reviewing the code, it was found that unlike TCP, for RBUDP the receiver was printing the percentage downloaded after every new chunk was received, slowing the program down.

**Conclusion:** TCP was more efficient than RBUDP in terms of transfer speed. Disproving the hypotheses. If this experiment were to be repeated, it would be suggested that all print statements be removed for a more accurate reading. However, printing the percentage transmitted is a nice feature for the user. Speed versus aesthetics is the trade of to be considered for production.

### Issued encountered

* One of the issues encountered was that the Receiver would be listening and waiting for a sender to connect to it, but the sender just would not connect for some reason, but we eventually got it working by moving the sender methods into the client class.
* It wasn’t possible to get the project working with Hamachi, thus the project only works on a single machine.

### Significant data structures

**Array Lists:** Array lists are dynamic data structures that allow us to store a collection of elements that can be accessed using an index-based system. Unlike arrays, array lists do not have a fixed size, which made the storing of clients more flexible.

### Design

The GUI of the file-sharing program allows users to upload and download files easily. Java Swing was used to implement the GUI for both the client and the server. The program is multithreaded and allows clients to send files to each other and receive files from each other. The clients are also able to communicate with each other. The files can be sent via two methods the client can choose from, RBUDP or TCP. The server displays the connected users as well as the files being sent.

### Compilation

It is advisable to ensure that the Server is operational and running before starting any client programs.

To compile and run the Server:

* “Javac Server.java”
* “Java Server”

To compile and run the Client:

* “Javac Client.java”
* “Java Client”