A picture containing symbol

Description automatically generated**KINGDOM OF SAUDI ARABIA**

**Ministry of Higher Education**

**Al-Imam Mohammad University**

**College of Computer & Information Sciences**

**Computer Networks Project Report:**

**Simple Reliable File Transfer Protocol with UDP and Acknowledgments**

|  |
| --- |
| Khaled Helayhel  Class: 172 |
| Abdulrahman Afit Askar Alanazi  Class: 171 |
| Faisal Othman Alkherayef Class: 171 |

**Team members:**

**Instructor:**

|  |
| --- |
| Dr. Mohammad Alotaibi |

**Academic year: 2023**

**Contents:**

[**Outline:** 2](#_Toc149577053)

[1. Introduction 2](#_Toc149577054)

[1.1 Aim of the project: 3](#_Toc149577055)

[2. Language Appropriateness 3](#_Toc149577056)

[**Functionalities** 5](#_Toc149577057)

[3. Project Architecture 5](#_Toc149577058)

[3.1 Packet Sequencing 5](#_Toc149577059)

[3.2. Recognitions (ACKs) 5](#_Toc149577060)

[3.3. Retransmissions 5](#_Toc149577061)

[3.4. Splitting Files and Managing Metadata 6](#_Toc149577062)

[3.5. Types of Packets and Headers 6](#_Toc149577063)

[3.6 Statistics 7](#_Toc149577064)

[3.8 Command Line 7](#_Toc149577065)

[3.8 Untrustworthy Channel 7](#_Toc149577066)

[**Description of Protocol Configuration** 7](#_Toc149577067)

[1.1 Sequence Numbers 7](#_Toc149577068)

[1.2 Acknowledgments (ACKs) 8](#_Toc149577069)

[**Message Format Description** 9](#_Toc149577070)

[1.1 Packet Structure: 9](#_Toc149577071)

[**User Manual** 9](#_Toc149577072)

[**Make file:** 10](#_Toc149577073)

[**Snapshot Illustrating Application Execution** 12](#_Toc149577074)

[**Technical challenges:** 14](#_Toc149577075)

[**References:** 16](#_Toc149577076)

# **Outline:**

# 1. Introduction

The goal of this project is to establish a dependable file transfer application across an erratic channel. The purpose of this project is to simulate an unreliable channel and handle a variety of tasks over an unstable network channel, including packet sequencing, acknowledgments (ACKs), retransmissions, file splitting, metadata handling, packet types and headers, statistics output, and command-line operations.

## 1.1 Aim of the project:

The aim of this project is to develop a Simple Reliable File Transfer Protocol with UDP and Acknowledgments by implementing features such as packet sequencing, acknowledgments, retransmissions, to ensure efficient data transfer.

# 2. Language Appropriateness

Because of its adaptability and capacity to make use of a number of crucial language features, libraries, and data structures that are extremely pertinent to the particular needs of our file transfer program, Java was chosen as the language for our project.

Object-Oriented Programming (OOP): A well-organized and modular codebase was designed and implemented mostly thanks to Java's OOP features.

Java Networking API: Network application development is made easier by Java's extensive networking library. The classes Datagram Socket, Datagram Packet, and InetAddress made it possible for us to effectively handle network communication.

Collections Framework: Strong data structures like HashMap and Set were made possible by Java's Collections Framework to manage the packets and acknowledgments. Safe concurrent access to shared data structures was therefore guaranteed.

Exception Handling: We were able to properly manage mistakes, exceptions, and timeouts during data transmission thanks to Java's exception handling framework. This enhanced our application's dependability.

Multithreading: Lang's multithreaded Java program. We were able to establish parallel execution thanks to thread support, which allowed packets to be sent and received simultaneously. For the transfer of data in real time, this is necessary.

File Handling: Effective file reading and sending was accomplished by using Java's file handling libraries. This made it easier for the recipient to rebuild files that had been divided into packets The Java Interactive. Java.IO and the file. FileInputStream: These classes were meant to be used while handling files. FileInputStream assisted you in reading data from the files, whereas File let you represent the files you wished to deliver.

Timers: Java's ScheduledExecutorService was used to implement timers in order to control packet timeouts. In situations where acknowledgments were not received promptly, this functionality enabled us to retransmit packets.

Exception Handling: We were able to properly manage mistakes, exceptions, and timeouts during data transmission thanks to Java's exception handling framework. This enhanced our application's dependability.

# **Functionalities**

## 3. Project Architecture

There are two primary parts to the project: the Sender and the Receiver. The sender is in charge of handling acknowledgments, retransmissions, file splitting into packets, and packet sequencing. Receivers receive packets, process them, identify lost packets, and acknowledge them.

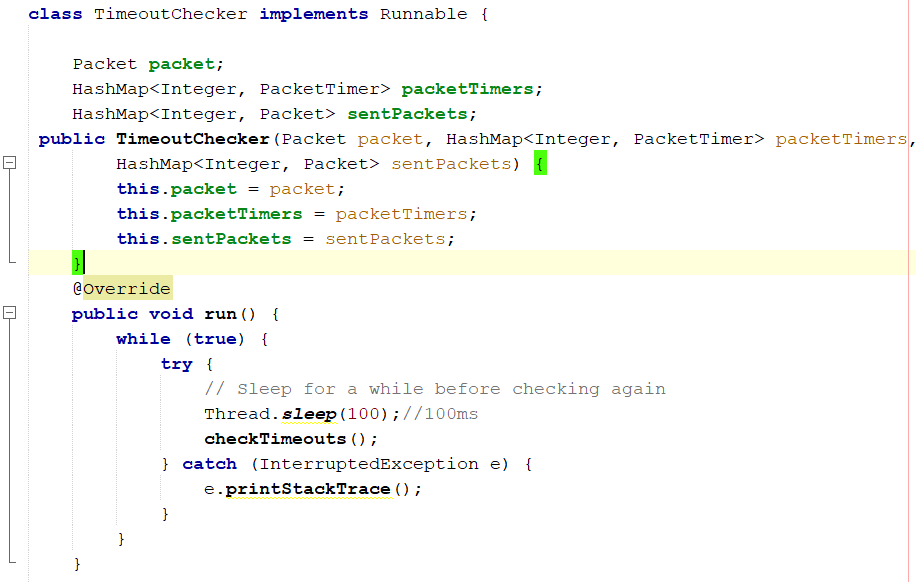
## 3.1 Packet Sequencing

The order in which packets are transmitted from the sender to the recipient is guaranteed by packet sequencing. A distinct sequence number is given to each packet, enabling the recipient to put them back together in the right order. Data Sender (Creation of Packets): Several kinds of packets can be generated, including.

3.2. Recognitions (ACKs) The purpose of acknowledgments (ACKs) is to verify that packets were successfully received. The sequence number of the most recent packet that was successfully received is sent by the recipient in ACK packets.

## 3.3. Retransmissions

**Create Timer:**



## 3.4. Splitting Files and Managing Metadata

To enable transmission, the application divides files into smaller packets. Information handled by metadata includes the file name and total size of the file.

## 3.5. Types of Packets and Headers

Data, ACKs, and metadata are among the packet types that can be distinguished from one another using packet types and headers. Important details including sequence numbers, packet types, data sizes, are contained in headers.

## 3.6 Statistics

Produced Accurate statistics are provided by the application on a number of different topics, such as delay and throughput. This data is useful in evaluating the file transfer's performance.

3.8 Command Line  
The use of command-line arguments facilitates the specification of essential details, including the IP address of the receiver, the port to be used, and the file intended for transmission, ensuring the application is both flexible and user-friendly.

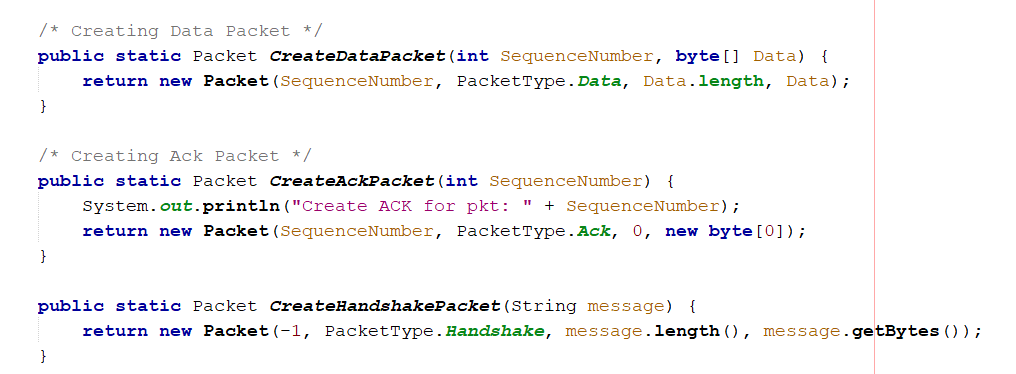
## 3.8 Untrustworthy Channel

The program replicates real-world network conditions by simulating an unstable network channel with packet loss. It is possible for packets to get misplaced, arrive out of order, or be duplicated. The receiver scans for various packet types and responds to them:

# **Description of Protocol Configuration**

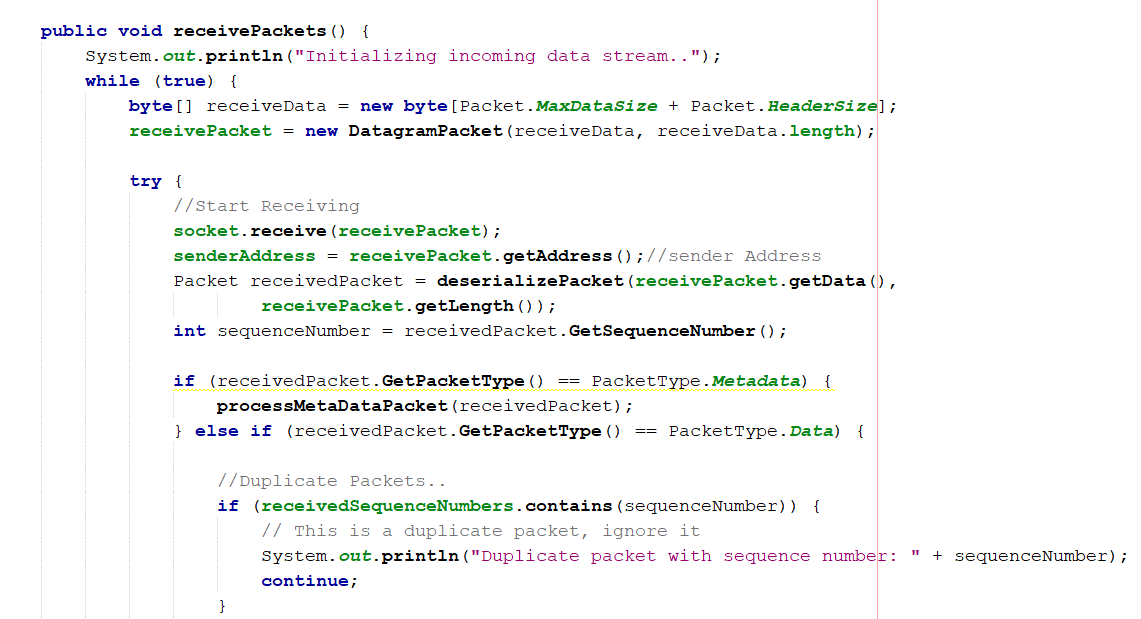
## 1.1 Sequence Numbers

Sequence numbers are used in the sender to maintain track of the order of each packet. As packets are created, they are given sequence numbers, which are used to identify acknowledgment. Sequence numbers and other packet information are encapsulated in the Packet class, which is used in the Sender class. A sequence number is given to a packet when it is created.



## 1.2 Acknowledgments (ACKs)

Acknowledgments (ACKs) are used by the receiver to verify packet receipt. When a packet is successfully received, the recipient notifies the sender via an ACK that the packet has been received. The sequence number is taken out of a packet and used to create the ACK packet by the Receiver class.



# **Message Format Description**

1.1 Packet Structure: The packet has the following structure.:

- A header (16 bytes) with the data size, packet type, and sequence number.

The actual file data is contained in data (variable bytes).

500-16=484bytes

Sequence Number 4 bytes  
Packet Type 4 bytes  
Data Size 4 bytes  
Header Size 4 bytes

# **User Manual**

1. **Pre-Setup Requirements**

Confirm that the Java Development Kit (JDK) is installed on your system.

1. **Instructions for Running the Program**

To initiate the file transfer utility, please proceed as follows:

* Compilation of Java Files
* Start by compiling the Java files for both the sending and receiving entities using a Java compiler. Execute the subsequent commands:
* For the server component: javac ReceiverAsServer.java
* For the client component: javac SenderAsClient.java

1. **Launching the Programs**

* Next, operate the sender and receiver programs in separate terminal instances. It is crucial to initiate the server before the client. Use these commands to run the programs:

To start the server: java ReceiveAsServer arg1 arg2

To launch the client: java SendAsClient arg1 arg2 arg3 arg4

Ensure to replace arg1, arg2, arg3, and arg4 with the actual Network Configuration arguments required by program. Remember, the server must be running before starting the client.

**3. Network Configuration**

1. **Server IP:** 127.0.0.1 arg1
2. **Server Port:** 6500 arg2
3. **File Path**: filePath.txt arg3
4. **Packet Loss Probability** = 0.1, 0.3, or 0.6 arg4

# **Make file:**

# Makefile for a File Transfer using UDP project with two main classes

# Define the Java compiler

JAVAC = javac

# Define the Java compiler flags

JFLAGS = -g

# Define the source directory

SRC\_DIR = src

# Define the build directory

BUILD\_DIR = bin

# Define the main classes

SENDER\_MAIN = Run\_Client

RECEIVER\_MAIN = Run\_Server

# Define source files

SOURCES=$(wildcard $(SRC\_DIR)/\*.java)

# Define class files

CLASSES = $(SOURCES: $(SRC\_DIR)/%.java=$(BUILD\_DIR)/%.class)

# Default target

all: $(CLASSES)

$(BUILD\_DIR)/%.class: $(SRC\_DIR)/%.java

@mkdir -p $(@D)

$(JAVAC) $(JFLAGS) -d $(BUILD\_DIR) $<

receiver: all

java -cp $(BUILD\_DIR) $(RECEIVER\_MAIN) arg1 arg2

sender: all

java -cp $(BUILD\_DIR) $(SENDER\_MAIN) arg1 arg2 arg3 arg4

clean:

rm -rf $(BUILD\_DIR)

PHONY: all sender receiver clean

# **Snapshot Illustrating Application Execution**

With probability 0.1:



A black screen with white text

Description automatically generatedWith probability 0.3:

With probability 0.6:

**Technical challenges:**1. Packet Sequencing

Getting packets to arrive in the right order and without duplicates is a challenging task. Solution: Give each packet a sequence number. It should keep track of the expected sequence number and discard any duplicate or out-of-order packets.

2. Acknowledgments (ACKs)

Putting in place a system that allows the recipient to send acknowledgement packets for each data packet they receive is a challenge. Solution: Create a special acknowledgment packet format and make sure the recipient replies to the sender for each packet they receive.

3. Retransmissions

The difficulty is in timing each packet and retransmitting if an acknowledgment isn't received in a timely manner. Solution: For every packet, use a timer (java.util.Timer in Java, for example). Send the packet again if the timer goes off before you hear back.

4. File Splitting  
Timing each packet and retransmitting in the event that an acknowledgment is not received promptly provide a challenge. We came up with the idea to use the java.util timer. Consider setting a timer for each packet. If the timer expires before you receive a response, send the packet again.

5. File Metadata

The challenge is to keep each packet under the maximum size restriction when we divide large files into smaller packets for transmission. Solution: Put in place a system that reads the file in sections and generates a packet for each section. Ensure that the chunk size is limited to 500 bytes.

6. Packet Types

Obstacle here while data transfer can start, the sender of the data and the recipient of the data must exchange file metadata, such as filename , file size. We can see that creating a specific packet format meant for MetaData sharing as well as making sure it happens before file transmission could be a means to solve the problem.

7. Packet Header

The difficulty is in telling data, acknowledgment, and handshaking packets apart.

Solution: To represent distinct packet kinds, include a packet type field in the packet header and use different values.

8. Ports

Creating a packet header with all the necessary metadata is the challenge.

Solution: Include fields for all necessary metadata and provide a class or structure for the packet header.

9. Command-Line Arguments

Using port 6000 for the sender and port 6500 for the recipient presents a challenge.

Solution: In the sender and recipient programs, hardcode or configure the port numbers.

10. Statistics

Problem: Allowing the sender to use command-line options to transfer a file and to specify the IP address and port number of the recipient.

Solution: To receive and properly parse command-line inputs, use Java's args[] array.

**References:**  
1. **Java Network Programming**

Oracle’s Java Tutorials – Custom Networking: Official tutorials from Oracle on network programming in Java.

2. **Computer Networking**

Computer Networking: Principles, Protocols and Practice: An open-source eBook that covers the fundamentals of computer networking.

3**. Reliable Data Transfer Protocols**

Transport Layer - Reliable Data Transfer (Kurose and Ross): Sections from the "Computer Networking: A Top-Down Approach" textbook that discuss reliable data transfer.

4. **UDP Programming**

Beej's Guide to Network Programming: A well-known guide that includes a section on UDP programming.

5. **Online Forums and Q&A Sites**

Stack Overflow: For asking specific programming-related questions.

Reddit - r/learnprogramming: A community for learning and asking programming-related questions.

6**. GitHub and Open Source Projects**

Exploring open-source projects that have implemented similar protocols can provide valuable insights. Searching on GitHub with keywords like “reliable UDP”, “file transfer protocol”, or “Java networking” may yield useful results.

7. **Video Tutorials**

Video tutorials on YouTube, Coursera, and edX that cover Java network programming and computer networking basics.