Kennesaw State University

Project Report

Bryson Phillip, JaDante Hendrick

CS4504

Patrick Bobbie

4/25/22

Abstract

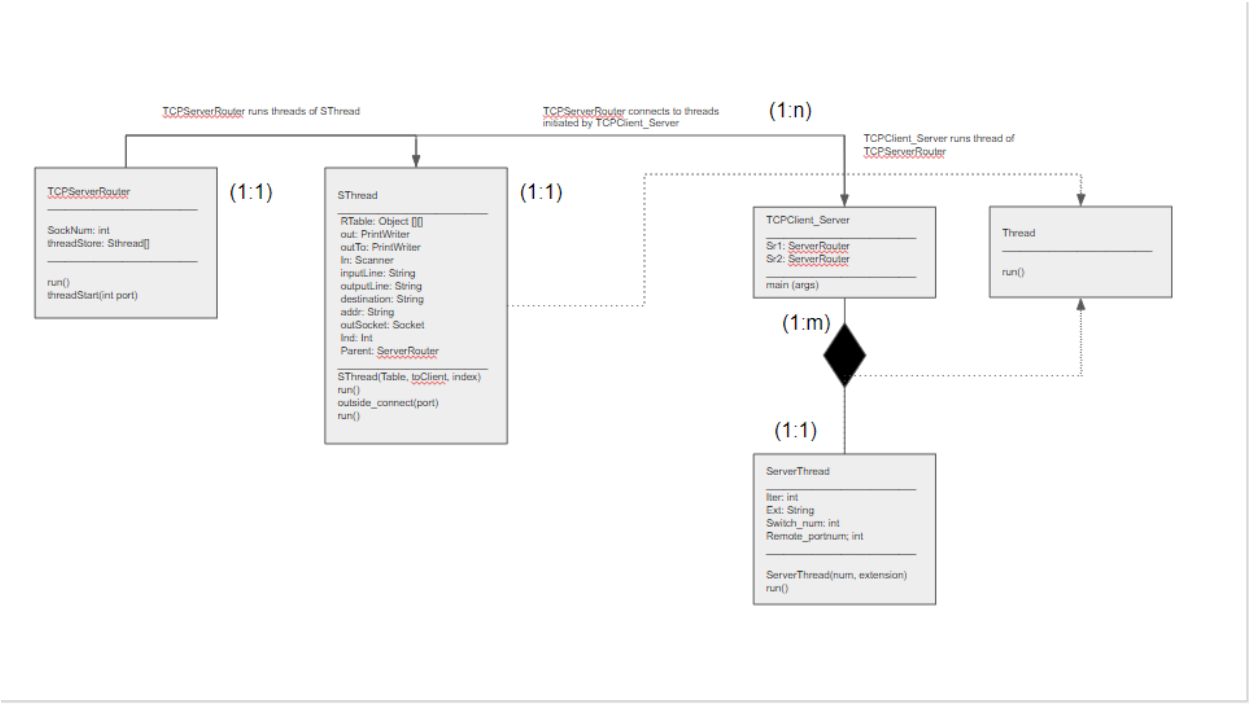
The purpose of this report is to showcase the performance of distributed systems using transmission control protocol (TCP). The type of distributed system that will be examined is the peer to peer system (P2P). Details regarding the implementation of the peer to peer system are described below. The tests on the systems involved finding out the transfer speeds of different file types when multiple pairs are being used . The report is concluded with a user guide that explains how to configure and use the files to run the system.

Introduction

This report will explore our group's implementation of the peer to peer network based off the modified code from part one. In order to execute this we used 4 classes: TCPClient_Server, ServerThread, SThread and ServerRouter. These classes are explained in more detail below. We also ran a test that tested the transfer speeds of multiple file types. That is gone into more detail below as well. Lastly, the report is concluded with a user guide that explains how to configure and use our implementation.

Design

UML Diagram



This UML diagram illustrates the relationships between all of the classes that make up the network. The classes that can be run individually are TCPServerRouter and TCPClient_Server. Both classes initiate or are composed of objects which extend Thread. Sthread, composing TCPServerRouter, contains an object table, wrapped input and output streams, socket objects, variables for storing messages, and addresses for communication. It has a method which implements run() and can be threaded. TCPClient_Server works as the entry point for the code. The client_server may create 10 or more thread pairs of ServerThread which each use Sthread combinations to forward data between nodes.

TCP Client Server

TCPClient_Server serves a similar function as TCP Server Router in that it is responsible for starting client/server threads. However, ServerThread can function as both a client and server. Two ServerRouters are instantiated on different port numbers and correspond to the N and M network clusters mentioned in the specs. By default, 20 peers are run in separate threads with incrementing port numbers. The 10 peers with the higher port numbers connect to the Sr2 ServerRouter while the rest connect to Sr1 (This has been altered during the testing phase). A user input loop is used to determine which subset of the running peers will be transmitting data as well as the corresponding receiver and the file. This information is stored in array ‘temp’ and is automatically stored in each of the necessary peers before they are activated.

Snippet of code that runs the server start loop:

```
import ...

public class TCPClient_Server {
        public static ServerRouter Sr1 = null;
        public static ServerRouter Sr2 = null;
        public static String [][] temp;
        public static void main (String[]args) {
            try {
                temp = new String[10][3];
                Scanner scan = new Scanner(System.in);
                Sr1 = new ServerRouter( num: 5549);
                Sr2 = new ServerRouter( num: 5550);
                Sr1.start();
                Sr2.start();
                Thread.currentThread().sleep( millis: 1000);
                int a = 0;
                while (true) {
                    System.out.println("Enter port number of a transmitting peer: ");
                    temp[a][0] = scan.nextLine();
                    System.out.println("Enter port number of a receiving peer: ");
                    temp[a][1] = scan.nextLine();
                    System.out.println("Enter the file to be transmitted: ");
                    temp[a][2] = scan.nextLine();
                    System.out.println("Add another pair?");
                    String temp4 = scan.nextLine();;
                    a++;
                    if (temp4.equals("no")) {
```

```
                }

            }

            ServerThread[] tempstor = new ServerThread[20];
            for (int i = 1; i < 21; i++) {
                ServerThread SERVER = new ServerThread(i, extension: "txt");
                SERVER.start();
                tempstor[i - 1] = SERVER;
            }


            Thread.currentThread().sleep( millis: 1000);

            for (int i = 0; i < a; i++) {
                tempstor[Integer.parseInt(temp[i][0]) - 5551].remote_portnum = Integer.parseInt(temp[i][1]);
                tempstor[Integer.parseInt(temp[i][0]) - 5551].filename = temp[i][2];
                tempstor[Integer.parseInt(temp[i][0]) - 5551].switch_num = 2;
                Sr1.threadstart(Integer.parseInt(temp[i][0]));
                Sr2.threadstart(Integer.parseInt(temp[i][0]));
            }

        }catch (Exception io){}
    }
}
```

ServerThread

Within ServerThread, the code to run as a receiver or transmitter is included. The global variable switch_num is used as a flag to indicate which role a given peer will be fulfilling.

```
        try {
            if (iter < 11) {
                socket = new Socket(addr, port: SockNum - 1, addr, (SockNum + iter));
            }
            else{
                socket = new Socket(addr, SockNum, addr, (SockNum + iter));
            }
            out = new PrintWriter(socket.getOutputStream(), autoFlush: true);
            in = new Scanner(new InputStreamReader(socket.getInputStream()));
            out.print(1+" ");
            out.flush();
```

The connection code has been reworked to include an initial send and receive prior to the server socket being registered in Sthread. A value of 1 indicates that a connection is

local to the address space that a particular SThread instance operates on. When 1 is sent, Sthread will not immediately attempt to connect to another peer and will wait for the threadStart function to be called externally from TCPClient_Server.

```
socket.setSoTimeout(500);
while (switch_num == 0){
    try {
        socket.getInputStream().read();
        switch_num = 1;
        break;
    }catch(Exception io){}
}
socket.setSoTimeout(0);
```

When start() is called on a peer, an entry in SThread will be made and then the thread will idle on read(). It will wait to be sent a single byte of data which will signal that the peer is to act as a server. This blocking is periodically interrupted by a time out, at which point the while loop will check to see if the variable ‘switch_num’ has changed. If so, it will cease blocking on read and move to the client section of the code, setting the wait time for the socket back to infinity.

```
if (i == 0) {
    portnum = Integer.parseInt(fromPeer);
    out.print((portnum) + " ");// initial send (IP of the destination Client)
    out.flush();
    fromPeer = in.next();// initial receive from router (verification of connection)
    i++;
```

The original server code automatically connected to a port number using a stable offset. The new code waits to be forwarded the port number by the client. This happens before the filename is sent.

```
System.out.println("Path of the file to be transmitted for peer at port: "+ (int)(SockNum+iter));
System.out.println(filename);
System.out.println("The port number of the receiving peer:");
System.out.println(remote_portnum);
```

The updated client code does not ask for the file or port number as this was taken care of in TCPClient_Server. It will instead display these values prior to transmitting.

SThread

```
// Constructor
SThread(Object [][] Table, Socket toClient, int index, ServerRouter parent) throws IOException
{
        out = new PrintWriter(toClient.getOutputStream(), autoFlush: true);
        in = new Scanner(new InputStreamReader(toClient.getInputStream()));
        RTable = Table;
        addr = String.valueOf(toClient.getPort());
        RTable[index][0] = addr; // IP addresses
        RTable[index][1] = toClient; // sockets for communication
        ind = index;
        Parent = parent;
        int hold = Integer.parseInt(in.next());
        if (hold == 1){
            return;
        }
        else{
            this.start();
        }
}
```

The SThread constructor has been updated to handle the initial send when registering a socket into the address table. The variable ‘hold’ stores whether or not the connection is local or external. A local connection is expected to be started in the TCPClient_Server code, or started by another SThread instance that forwards the connection between Sr1 and Sr2. If the connection is external, it is being created for another socket that forwards a connection between ServerRouters. This socket will need to communicate with the receiving peer immediately and requires that the SThread be started automatically. A reference to the calling ServerRouter is also included so that threadStart() can be called from within an SThread.

```java
public void outside_connect(int port){
    try {
        InetAddress addr1 = InetAddress.getLocalHost();
        Socket Socket = new Socket(addr1, port, addr1, localPort: 0);
        outSocket = (Socket) Socket;
        outTo = new PrintWriter(outSocket.getOutputStream(), autoFlush: true);
        outTo.print(2+" ");
        outTo.flush();

        outTo.print(destination + " ");// initial send (IP of the destination Client)
        outTo.flush();
        inFrom = new Scanner(new InputStreamReader(outSocket.getInputStream()));
        inFrom.next();// initial receive from router (verification of connection)

        while ((inputLine = in.next()) != null) {
            outputLine = inputLine;
            if (outSocket != null) {
                if (outputLine.equals("Bye.")) { // exit statement
                    outTo.print(outputLine + " ");
                    outTo.flush();
                    break;
                }
                // passes the input from the machine to the output string for the destination
                outTo.print(outputLine + " "); // writes to the destination
                outTo.flush();
            }
```

In the event that a client tries to connect to a server in the other ServerRouters address space, the address lookup section will call ‘outside_connect’. Outside_connect will create a new Socket and connect to the other ServerRouter. It sends 2 initially so that the new Sthread instance will begin working immediately. It forwards the ‘destination’ variable to the new Sthread as if it were local. outTo is changed to the new socket so that data from the client will be forwarded twice: once to its own Sthread instance that invoked outside_connect, and then to the external SThread that can directly access the server. This same process will be performed again when the server returns data back to the client.

```
        int i = 0;
        outTo.print('r');
        outTo.flush();
    while ((inputLine = in.next()) != null) {
        System.out.println("Client/Server said: " + inputLine);
        outputLine = inputLine;
        if (outSocket != null){
            if (outputLine.equals("Bye.")) { // exit statement
                outTo.print(outputLine+ " ");
                outTo.flush();
                break;
            }
            // passes the input from the machine to the output string for the destination
            outTo.print(outputLine + " "); // writes to the destination
            outTo.flush();
        }
        i++;
        if (i == 1){

            Parent.threadstart(Integer.parseInt(destination));
        }
```

The original Sthread connection loop includes an extra send to wake up the server. It also ensures that the Sthread which will return the data is active in the last statement. (The last line is effective if the SThread was activated by the socket created in outside_connect)

ServerRouter

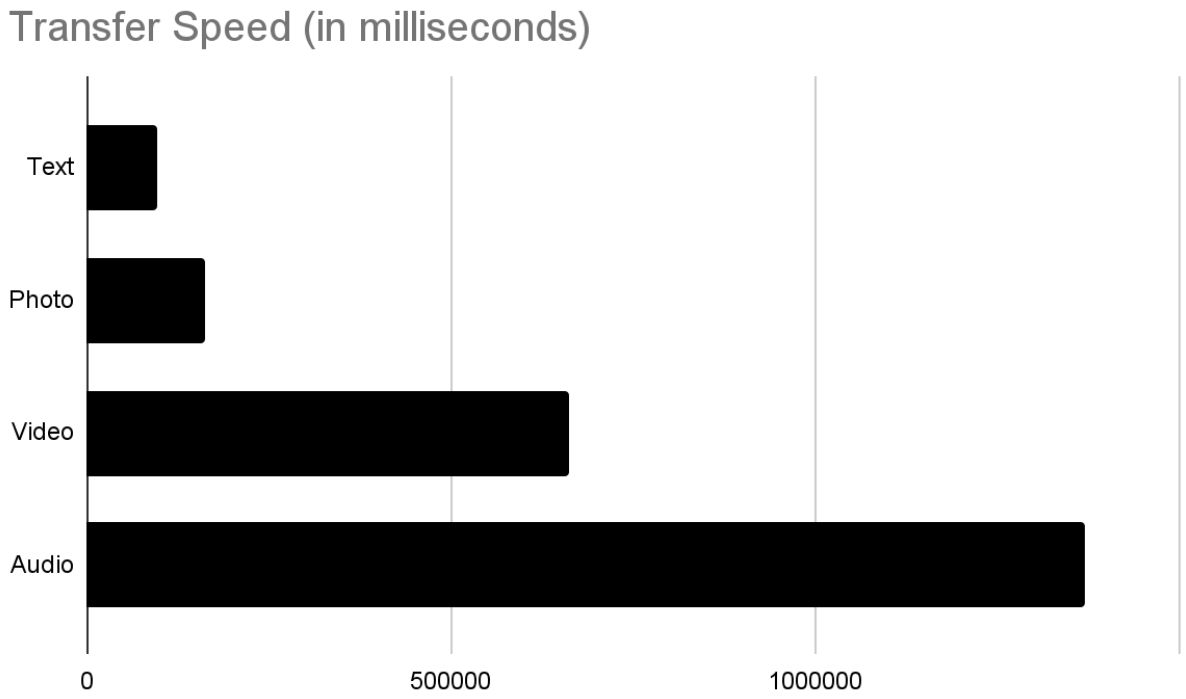
```
public class ServerRouter extends Thread {
    private int SockNum;
    private SThread[] threadStore;
    ServerRouter(int num){
        SockNum = num;
        threadStore = new SThread[20];
    }

    public void threadstart(int port){
        try {
            for (SThread i : threadStore) {
                if (i.addr.equals(String.valueOf(port)) && !i.isAlive()) {
                    i.start();
                    break;
                }
            }
        }
        catch (Exception io){}
    }
```

The ServerRouter has been updated to allow a Server sockets local port to be manually inputted in TCPClient_Server. The variable threadStore is an array of all of the Sthread objects that have been activated on the local ServerRouter. ThreadStart can activate start() for an Sthread instance by specifying the port number, with the condition that it has not already been started. This is used to activate the communication cycle from within TCPClient_Server and to activate remote communication cycles when starting from a different Server Router.

Simulation

Transfer Speeds Of Multiple File Types:



The files used in this test were a text file (517 kilobytes), a 960x679 image (868 kilobytes), a 01:27 video (3,392 kilobytes) and a 03:03 song (7,190 kilobytes). The text file took 98274 milliseconds to be transferred, the image took 163969 milliseconds, the video took 662125 milliseconds and the audio took 1368980 milliseconds. This test was done on a multi-core system and ten pairs were used (in the interest of time). The results were linear, relative to the size of the file.

User Guide

Configuration

All files should be downloaded and put into the same folder. If needed the port numbers for the Server-Router can be changed by editing the arguments of the ServerRouter object being created. The port numbers of the clients can be changed by editing portNum1 and portNum2. The file type can be changed by locating and changing “txt” further beneath the code pictured below. All of this is found in the TCP Client Server file.

```
temp = new String[200][3];
Scanner scan = new Scanner(System.in);
Sr1 = new ServerRouter( num: 5549);
Sr2 = new ServerRouter( num: 5550);
Sr1.start();
Sr2.start();
Thread.currentThread().sleep( millis: 1000);
int a = 0;

int j = 0;
int portNum1 = 5555, portNum2 = 5556;
System.out.println("Enter the file to be transmitted: ");
String path = scan.nextLine();
```

Execution

To execute, run TCP Client Server. Messages will appear prompting you to enter a port number for each of the clients. Once you enter those you will be prompted to enter a file path. After that you will be asked if you want to create another pair. If you don’t want to make another pair then enter “no”.