

CS 4504

Distributed Computing

Project: Phase 2

Report

Aman Bhimani

Evan Ross

Mehtab Chithiwala

Table of Contents

[Table of Contents](#)

[Abstract \(NEEDS UPDATE\)](#)

[Introduction \(NEEDS UPDATE\)](#)

[Design Approach](#)

[Implementation of Modules](#)

[TCP ServerRouter](#)

[TCP ServerRouter2](#)

[ClientNode](#)

[ServerNode](#)

[Simulation of Modules](#)

[Data and Analysis](#)

[Conclusions and Findings](#)

Abstract

This project is an enhanced version of our Phase I Project. The project is based on the idea of Distributed Computing, which involves multiple computers and devices interconnected to each other via a network. This network can be over the internet, LAN, bluetooth, or any number of protocols. For our project, we have decided to do this over a Local Area Network (LAN). This means that all computers and devices have to be connected to the same wireless (or wired) network.

Since this implementation focuses on the P2P Node to Node connection relationship, our implementation had numerous client/server nodes grouped into 2 clusters, and 2 routers that were used as middlemen to facilitate the connections and IP/port lookups. We used our personal laptops and computers in order to have sufficient nodes for the simulation.

To determine and analyze the speed and efficiency of our distributed system, we decided to send over various types of messages and file types. Initially, we used a text file with various lines of varying length. After that, we sent over video files of the .mp4 format of various resolutions. This also helped us compare differences in cycle time in relation to message size and video resolutions.

Going off of the initial design requirements given in the Phase 2 Description, we developed a smaller scale design consisting of hubs of multiple servers and routers each connected to their own router. We decided to not reuse any of the Phase 1 code and developed the implementation for this phase from scratch. We felt that this approach gave us more freedom to develop the code in a way that was more suitable to the design that we were trying to accomplish.

After running simulations and analyzing the data, we came to the conclusion that the amount of time it took to send over files was directly proportional to what we were sending over. The correlation between line length and cycle time was easy to see, as was the linear relation of increased video size (resolution) and cycle time.

Introduction

The core essence of this project starts with the two words, Distributed Computing. The name of this course perfectly describes this project because it is a matter of multiple computers contacting each other, asking for information or results.

The main purpose of this project is to simulate a networking environment using the APIs provided in the Java language, including Sockets. Since most of the code was already implemented, our project participants were responsible for understanding the code through debugging and reading comments. There were a few changes and additions needed to make the project worthy of a report which are discussed throughout the report.

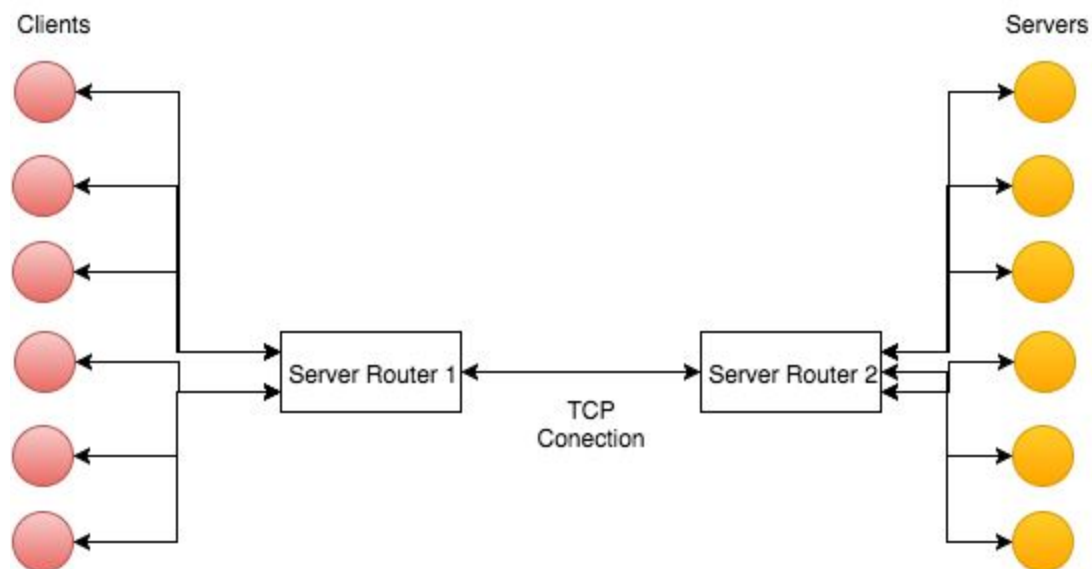
The first few days of the project time was spent understanding the requirements and reading through the documents like the User Guide and Project Specifications provided. Next, we all read through the code and the comments on the code to understand how it all works. Design and Implementation were done simultaneously. We all used our laptops where one was the ServerNode, and the other two were acting as the ClientNode. Finally, the code was edited to fit our needs and collect data such as message size and cycle time.

Design Approach

The design of this phase has changed a little bit from Phase I. In the first phase, there was only one ServerRouter, and all clients and servers connected to it. They were all on the same “subnet”. This means that the main ServerRouter knows the IP addresses of each of the clients, and each of the servers.

However, in Phase II, the design has changed so that there are two ServerRouters and they are both constantly connected via a TCP connection. One ServerRouter holds the IP addresses of Clients, and the other ServerRouter holds the IP addresses of the Servers.

A client must contact it's ServerRouter and get the IP address of the Server, then initiate a TCP Connection.



Implementation of Modules

For the second phase was broken down into four modules:

TCP ServerRouter

This ServerRouter has one main method which handles all the tasks and it is very systematic. First, the server router makes ServerSocket for accepting connections from the other ServerRouter, and also a Socket for outgoing connections and messages (for the client).

It then waits for any incoming connections. Once it receives a connection, it starts all servers and keeps them open for any requests.

Once it receives a request from a client to connect to a server, it fetches the IP of the a Server and sends it over to another ServerRouter.

TCP ServerRouter2

ServerRouter2 is responsible for connecting to the first ServerRouter. It is also responsible for processing requests from Clients to get Server IP and ports.

getServerIP(): This is a method that returns the Server IP received from ServerRouter2 to the Client.

getServerPort(): This method returns the Server port received from ServerRouter2 to the Client.

getInfo(): This method launches the ClientNode and registers it in the ServerRouter2.

ClientNode

The client node also only has one main method that handles all processes. First, it gets the IP address of the Server by using "getServerIP" from the ServerRouter. Then it connects to the said server, and initiates all DataOutputStream and DataInputStreams.

The while loop runs until the file is out of lines, and it sends all messages to the connected Server. Once the file is completed, the Client ends the connections.

ServerNode

The ServerNode first creates a ServerSocket and waits for an incoming connection. Once the connection is made, it creates a FileWriter and DataOutputStream and DataInputStreams.

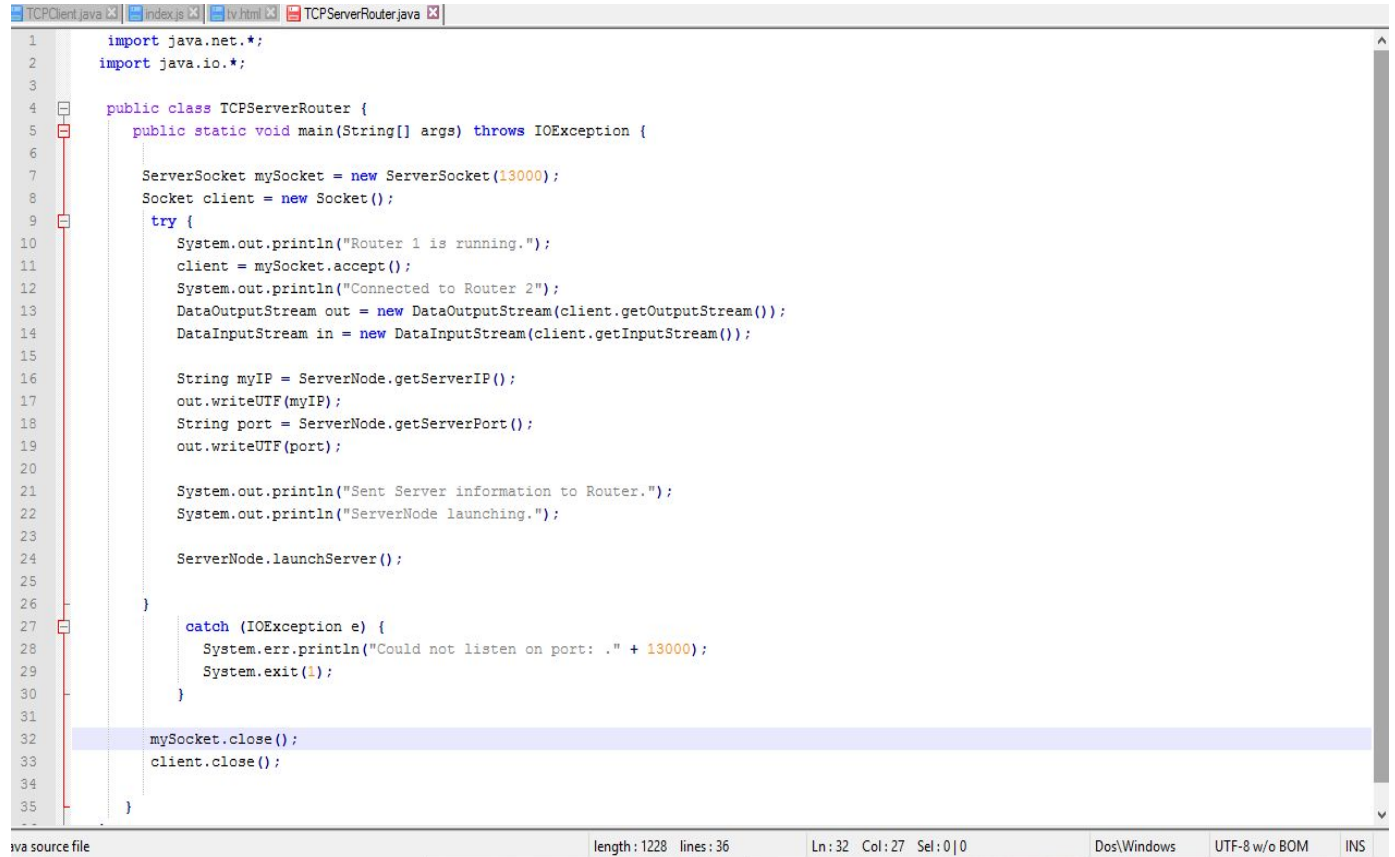
The FileWriter is made to make new files locally from the information that is coming in from DataInputStreams.

We have also implemented another version of the ServerNode that handles video files in mp4 format. The process is the same, but the file type is different.

Code Snippets

The code snippets below highlight the implementation of some of the modules mentioned above.

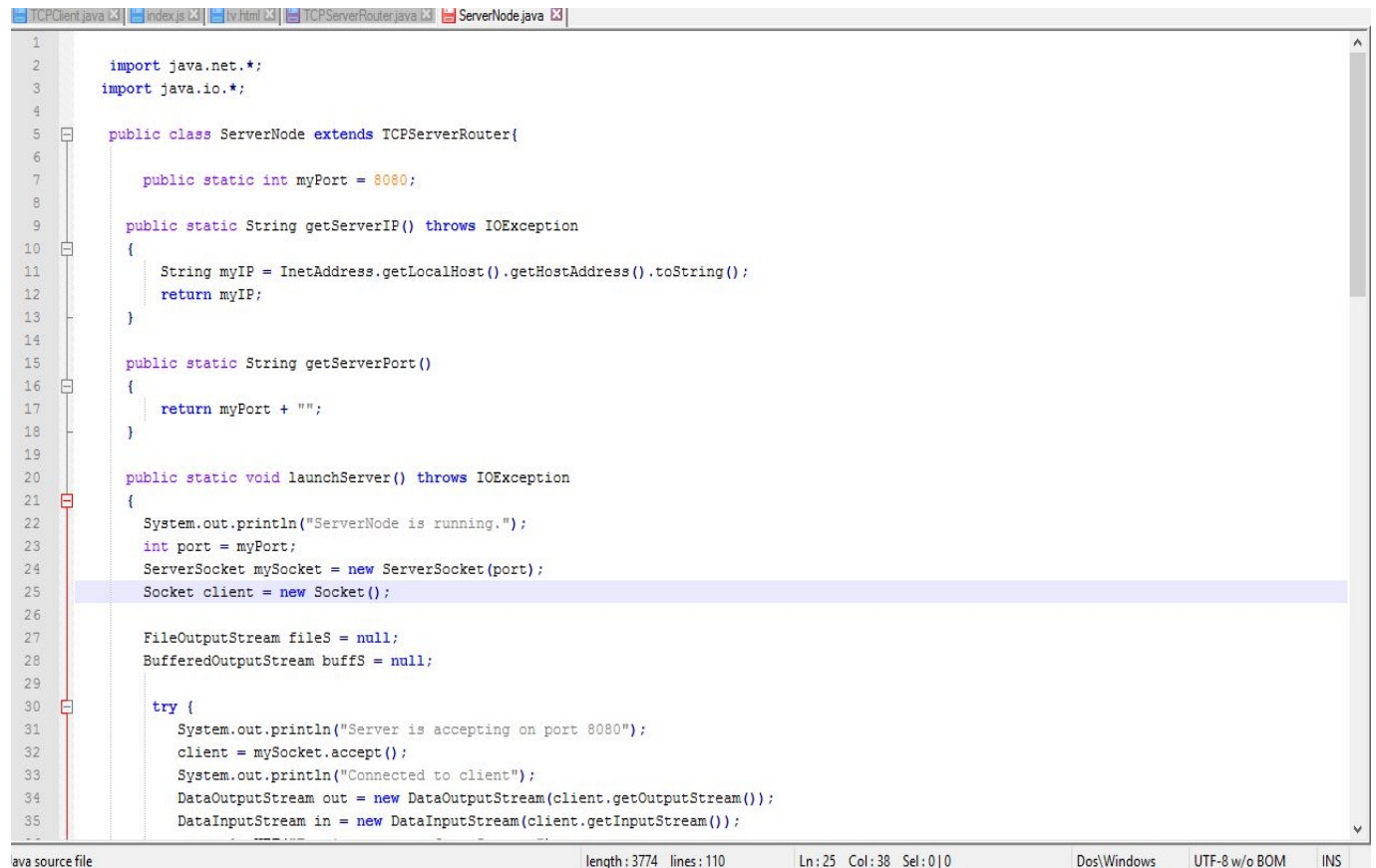
TCPServerRouter:



```
1  import java.net.*;
2  import java.io.*;
3
4  public class TCPServerRouter {
5      public static void main(String[] args) throws IOException {
6
7          ServerSocket mySocket = new ServerSocket(13000);
8          Socket client = new Socket();
9          try {
10             System.out.println("Router 1 is running.");
11             client = mySocket.accept();
12             System.out.println("Connected to Router 2");
13             DataOutputStream out = new DataOutputStream(client.getOutputStream());
14             DataInputStream in = new DataInputStream(client.getInputStream());
15
16             String myIP = ServerNode.getServerIP();
17             out.writeUTF(myIP);
18             String port = ServerNode.getServerPort();
19             out.writeUTF(port);
20
21             System.out.println("Sent Server information to Router.");
22             System.out.println("ServerNode launching.");
23
24             ServerNode.launchServer();
25
26         }
27         catch (IOException e) {
28             System.err.println("Could not listen on port: ." + 13000);
29             System.exit(1);
30         }
31
32         mySocket.close();
33         client.close();
34     }
35 }
```

java source file length: 1228 lines: 36 Ln: 32 Col: 27 Sel: 0 | 0 Dos/Windows UTF-8 w/o BOM INS

ServerNode:



```
1
2  import java.net.*;
3  import java.io.*;
4
5  public class ServerNode extends TCPServerRouter{
6
7      public static int myPort = 8080;
8
9      public static String getServerIP() throws IOException
10     {
11         String myIP = InetAddress.getLocalHost().getHostAddress().toString();
12         return myIP;
13     }
14
15     public static String getServerPort()
16     {
17         return myPort + "";
18     }
19
20     public static void launchServer() throws IOException
21     {
22         System.out.println("ServerNode is running.");
23         int port = myPort;
24         ServerSocket mySocket = new ServerSocket(port);
25         Socket client = new Socket();
26
27         FileOutputStream fileS = null;
28         BufferedOutputStream buffS = null;
29
30         try {
31             System.out.println("Server is accepting on port 8080");
32             client = mySocket.accept();
33             System.out.println("Connected to client");
34             DataOutputStream out = new DataOutputStream(client.getOutputStream());
35             DataInputStream in = new DataInputStream(client.getInputStream());
```

ava source file length: 3774 lines: 110 Ln: 25 Col: 38 Sel: 0 | 0 Dos/Windows UTF-8 w/o BOM INS

Simulation of Modules

The simulation of the second phase was quite similar to the first phase, except a small detail: there are two ServerRouters.

ServerRouter starts first and is waiting to accept connections from another ServerRouter. This is when ServerRouter2 is started and initiates a TCP connection to ServerRouter as long as the IP is known. Once the connection is made, both ServerRouters can send messages to each other directly.

Then, ServerRouter2 initiates the Server which runs indefinitely and accepts any incoming TCP connections. ServerRouter2 knows about all servers that are online and has a Routing Table that holds all server IPs and statuses.

Then, ServerRouter1 starts the Client who then sends a request to ServerRouter1 for a Server IP address. The ServerRouter1 sends the same request to ServerRouter2, which fetches the first server it finds in the Routing Table.

The IP is then sent back to ServerRouter1 and back down to the Client who first requested a server. The Client now has all information needed to process the request for a TCP connection.

After the TCP connection between Server and Client is made, the Client sends any file or text that is in the queue over to the Server and the server makes a new file to save it.

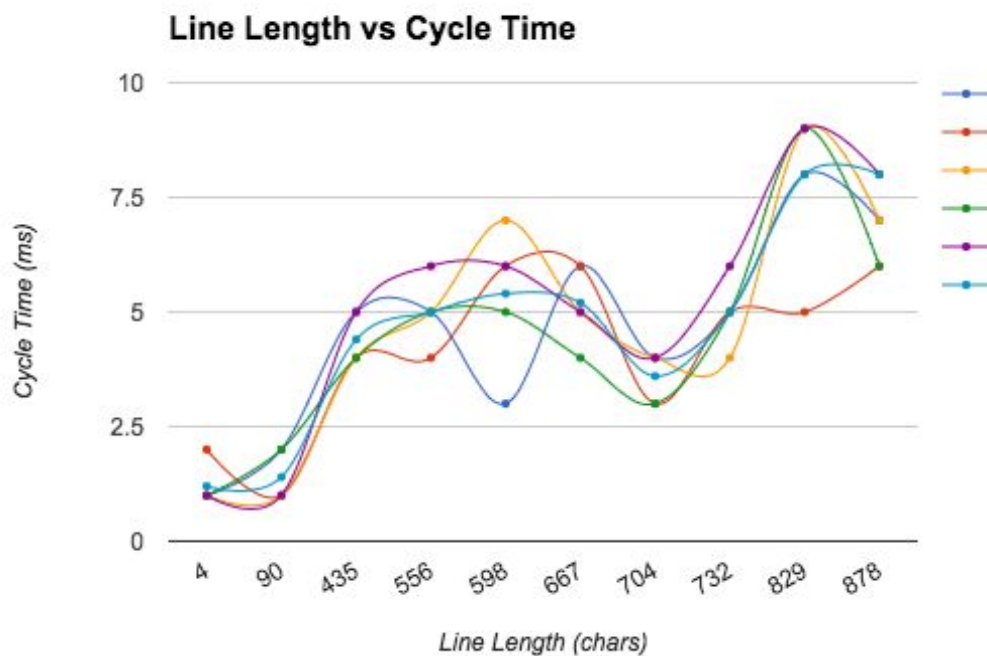
Data and Analysis

Text Files

Line Length (chars)	Cycle Time (ms)						Ratio (chars/ms)
	Trial 1	Trial 2	Trial 3	Trial 4	Trail 5	Avg	
4	1	2	1	1	1	1.2	3.33
90	2	1	1	2	1	1.4	64.29
435	5	4	4	4	5	4.4	98.86
556	5	4	5	5	6	5	111.20
598	3	6	7	5	6	5.4	110.74
667	6	6	5	4	5	5.2	128.27
704	4	3	4	3	4	3.6	195.56
732	5	5	4	5	6	5	146.40
829	8	5	9	9	9	8	103.63
878	7	6	7	6	8	8	109.75
					AVERAGE		125.55

The table here represents the multiple trials we have done over the text file provided to us in the project specifications. The line characters are ordered from lowest to largest character count to get a good understanding of the Average cycle time and the ratio of characters per millisecond.

Total of five trials were recorded and the average of those cycle times were taken. Then the ratio of the trial was recorded in the units of characters per millisecond. This lets us know how many characters can be sent per millisecond through the network.



Text file Data Analysis:

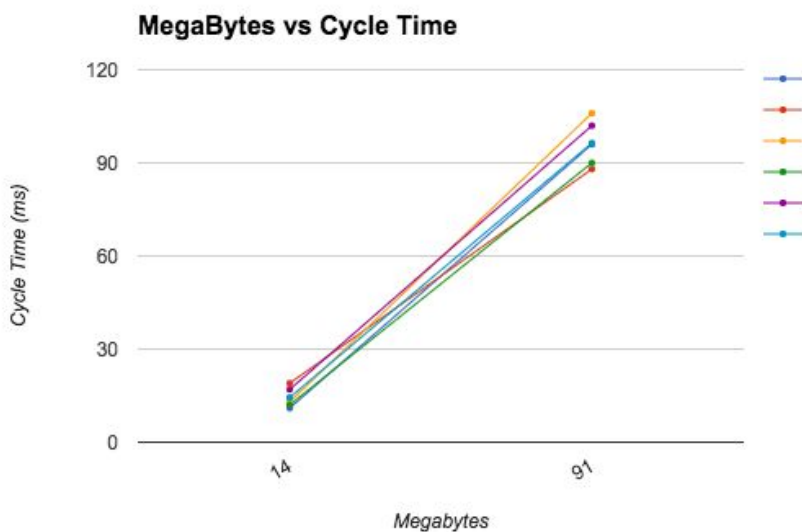
It should be noted that the character types do not matter since it is all written in unicode and each character is the same amount of data being sent. So it does not matter if you send a "?" or a "\". It requires the same amount of data. Therefore, only the number of characters matter, which is what we are investigating here.

As you can see the cycle time directly correlates to the number of characters in the message. The time taken consistently rises as the amount of text does. This makes sense because the more text we have transmitting through the network, the more amount of time it takes.

There are some inconsistencies, such as in the dark blue trail line, on 598 characters, it dips down to around 3 ms instead of a steady 6-8ms. This could be due to less traffic in the network. The rest of the data is quite consistent with each other.

Video File

MegaByte s	Cycle Time (s)						Ratio (MB/s)
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Avg	
14	11	19	13	12	17	14.4	0.97
91	96	88	106	90	102	96.4	0.94
					AVERAGE:		0.96



Video File Analysis

We have tested two videos, one 14MB and another 91MB. The times for transferring these files are consistent with each other. Both files were transferred to the other ServerNode at the rate of about 1MB per second. This was true for most trials in our case.

We believe that the video files are so consistent because they are not strings of text, rather bytes of data. The data does not have varying length and is sent over in a single stream one byte after another.

Conclusions and Findings

In this project, we designed and implemented a distributed system of three nodes (our laptops) to communicate with each other. The main goal of this project was to learn about how distributed systems work and actually see the result in implementation. A file was transmitted line by line from the Client to the Server and was converted into uppercase letters and sent back to the Client.

In this phase of the project we introduced a new server and clients to the original server and clients from phase one of the project. The data for the cycle times vs line lengths leads to the analysis that the differences in cycle times had little to do with the line lengths, and had more to do with network and hardware discrepancies. In the addition of the text file from phase one we transferred two different mp4 movie files of differing sizes.

We also ran into some problems while working on this project. A major issue we ran into was connecting to each other's computers remotely using wireless LAN so that we could work on the project while at home. Using a software like Hamachi didn't work either, so we resorted to just meeting more frequently and working in person. Another issue related to connection was that even when we were in person, one of the three laptops might connect to a different router. We resolved this by setting up a wireless hotspot and using that as our router. Another problem we encountered was deciding which IDE to use, and some problems with setting up configurations on IntelliJ. We resorted to using IntelliJ for the Router, as that was working fine with the configuration setup, and used BlueJ (a lightweight Java IDE) for the Client and Router.