**(Cover Page)**

**Progress Report:** The Design and Implementation of a Client-Server Paradigm using TCP

**Date: 9/29/24**

**Project Name:** CS 4504 PROJECT REPORT – PART1

**Report Prepared by:**

**Group 5 (Parallel Distributed Computing, Section W03, Fall 2024)**

|  |  |  |  |
| --- | --- | --- | --- |
| Courtney Faulkner  cfaulk16@students.kennesaw.edu | Nicholas Hodge  nhodge8@students.kennesaw.edu | Ashton Mahatoo  amahatoo@students.kennesaw.edu | Colson Sims  csims88@students.kennesaw.edu |
| Joshua Smith  jsmi1290@students.kennesaw.edu | Mike Tokura  mtokura@students.kennesaw.edu | Carinne Tzurdecker  ctzurdec@students.kennesaw.edu | Giovanni Zavala  gzavala1@students.kennesaw.edu |

**Report Submitted to:**

**Professor** Patrick O. Bobbie, PhD

Email: pbobbie@kennesaw.edu

Office Location: Atrium Bldg, J386

Office phone: 470.578.3810

CS 4504 PROJECT REPORT – PART1

Fall 2024

Names: Courtney Falkner/W03, Nicholas Hodge/W03, Ashton Mahatoo/W03, Colson Sims/W03, Joshua Smith/W03, Mike Tokura/W03, Carinne Tzurdecker/W03, Giovanni Zavala/W03

ABSTRACT

This document is a progress report for the first phase of the semester-long project in the Fall 2024 Parallel and Distributed Computing Course, Section W03, instructed by Dr. Bobbie. The report outlines our team’s approach to implementing communication between client and a server process via a server-router. In this phase, we modified the provided Java code to establish reliable routing and data transfer, while ensuring synchronization between machines. The report also includes a comprehensive analysis of the experiment, featuring tables that summarize the bytes of text transmitted, tabulated timing data, and corresponding graphs. Additionally, we discuss key observations related to performance, data handling, and potential bottlenecks. Finally, we will share our findings and insights from this stage of the project.

Introduction

The purpose of this project is to design and implement a Client-Server Paradigm using Transmission Control Protocol (TCP) to demonstrate how a client and a server can communicate through a router, functioning as a bridge between the two. The client and server will be connected in different network segments, and TCP will ensure reliable communication between them. This project is vital for understanding how network routing and TCP communication are handled in distributed systems, particularly in cybersecurity and network management scenarios.

In distributed systems, client-server communication is vital for transferring data and services across networks. This progress report details the implementation of communication between a client and a server via an intermediary router. The setup involves a client sending a lowercase message to the server through the router, where the server converts the message to uppercase and returns it to the client.

The system comprises three main components:

Client: Initiates communication by sending a message.

Router: Acts as an intermediary, forwarding messages between the client and server.

Server: Processes the client's message, converting it to uppercase, and sends it back to the client.

This document outlines the key components, implementation details, and functionality of each part of the system.

The rest of the document will consist of the design architecture, the implementation approaches, the simulation method, data analysis, conclusion, and references. In the design architecture section, we’ll explain the basis of how the client and server communicates with each other through our program. The following section, implementation approaches, elaborates on this further and explains exactly how we implemented the client-server interaction through ports, routers, and more. The simulation method section explains the process of setting up the program, and how we tested the program exactly. In the conclusion, it’s explained how our initial goal with this program was accomplished and what we’ll continue to do in the next part. In the references section you can find any sources we used for information on how to complete the given process.

Design Architecture

The architecture consists of a client, router, and server, each functioning as distinct entities. The client communicates with the router on a predefined port (4000), and the router forwards the client’s message to the server, which listens on a different port (5000). Upon receiving the message, the server processes it and sends the response back to the client via the router.

Architecture Flow:

Client: Sends a lowercase message to the router.

Router: Forwards the message to the server.

Server: Receives the message, converts it to uppercase, and sends the response back.

Router: Receives the server’s response and forwards it to the client.

Client: Receives the uppercase message from the server.

Each component communicates via sockets over designated ports, with the router acting as a bridge between the client and server.

Implementation Approaches:

Client Implementation:

The client is responsible for initiating communication by sending a lowercase message to the router. The communication occurs through a socket connection established with the router. Figure 1 shows how the client operates.

A screenshot of a computer program

Description automatically generated

Fig. : Client Java Class

In this implementation, the client:

- Connects to the router on port 4000.

- Sends a message to the router using a PrintWriter.

- Waits for a response from the router using a BufferedReader.

- Once the message is received, the client closes the connection.

Router Implementation:

The router acts as an intermediary between the client and server. It listens for messages from the client and forwards them to the server. Upon receiving a response from the server, the router sends it back to the client. Figure 2 shows a simplified version of the router's code.

A screenshot of a computer program

Description automatically generated

Fig. : Router Portion of Program (Simplified)

The router performs the following tasks:  
- Waits for a connection from the client on port 4000.  
- Once connected, it forwards the client’s message to the server on port 5000.   
- The router receives the server’s response and sends it back to the client.  
- It repeats the process indefinitely for any new client connections.

Server Implementation:

The server listens for incoming messages from the router, processes them, and returns a response. In this case, the server converts the client's message to uppercase before sending it back. Figure 3 displays the relevant code for the server.

A screenshot of a computer program

Description automatically generated

Fig. : Snippet of Server Code Portion

The server follows these steps:  
- Waits for a connection on port 5000.  
- Receives the client's message via the router.  
- Converts the message to uppercase and sends the response back.  
- Closes the connection after sending the response.

Simulation Method

The system was successfully tested with the following message flow:

The client sent a message: "client One, Hello Server!".

The router received the message from the client and forwarded it to the server.

The server converted the message to uppercase: "CLIENT ONE, HELLO SERVER" and sent the response back to the router.

The router received the server’s response and sent it to the client.

The client received the response from the server through the router: "Server Response: CLIENT ONE, HELLO SERVER".

This process demonstrated the correct functionality of the system and validated the communication between the client, router, and server. This is illustrated in Figure 4.

A screenshot of a computer screen

Description automatically generated

Fig. : Client, Router, and Server Sample Communication

Running the Program:

1. Open 3 different command line windows.

2. Navigate to the file where this program is located in each command line window.

3. Compile each Java file in its own command line window.

javac Router.java

javac Server.java

javac Client.java.

4. Start the Server, Router, and Client, in this order in there respective window:

java Server.java

java Router.java

java Client.java

5. Process for the text message:

The client will send the specified text to the router.

The router will get the text from the client and send it to the server.

The server will receive the text from the router, process it and send it back to the router.

The router will receive the text from the server and send it back to the client.

Time

The time it takes for communication between a client process and a server process, particularly through a router, is influenced by several factors, including network latency, the speed of socket creation, message transmission, and processing on both the client and server sides. In our case, the time begins when the client sends a message to the router, which acts as an intermediary, and ends when the client receives a response from the server. Each phase of this communication involves some overhead, such as establishing a socket connection, forwarding the message through the router, and the server processing the data before sending the response. Typically, this entire process can take milliseconds or even nanoseconds in a local setup, depending on the message size and processing power.

As the message length increases, the time taken for communication also increases. This is due to the extra time required to transmit larger chunks of data over the network and for the server to process them. For each additional character or word in the message, the transmission time, as well as the processing time on both the router and server, will rise. While these increments in time might be negligible for short messages, longer texts result in a proportional increase in communication time. In high-performance systems, this may still be in the range of nanoseconds, but the cumulative effect becomes noticeable with significantly larger data sizes.

Moreover, the server’s conversion of the message to uppercase adds a small, but measurable, processing time. The more characters the server needs to convert, the longer it takes to process the request and return a response. For each character, the server applies the toUpperCase() function, and the system’s efficiency in handling larger payloads can affect the overall communication delay. Although these increases in time are often measured in nanoseconds, they become more pronounced with the increasing length of the message, especially in systems with limited processing resources.

This section is illustrated in Figure 5.

A screenshot of a computer program

Description automatically generatedA screen shot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated

Fig. : Illustration of time of communication between Client, Server, and Router.

Conclusion

The implementation of communication between one client process and one server process via a server-router effectively simulates the core principles of client-server architecture. In this system, the client sends a message to the server through the router, and the server processes the message, converts it to uppercase, and sends it back. This simple interaction demonstrates the essential elements of request-response communication in distributed systems, relying on socket programming for message exchange. By breaking down the process into distinct components—client, router, and server—the project illustrates how intermediary routers can manage message routing and ensure that the data is correctly transmitted across the network.

This system also mimics some aspects of the TCP/IP protocol, particularly in how the client and server establish connections via sockets, akin to the handshake mechanism in TCP. The use of a router adds another layer of abstraction, simulating real-world network infrastructure where messages traverse multiple devices. The simulation ensures reliable message delivery between the client and server, with the router acting as a transparent middleman. However, the simplicity of the project means that it does not cover all features of the TCP/IP stack, such as congestion control, error detection, or packet fragmentation. Nonetheless, the fundamental concept of reliable message exchange is effectively demonstrated.

In terms of performance, the time it takes for a message to travel from the client to the server and back is relatively short, given the small scale of the system. However, in larger networks with more traffic, delays could become more significant due to factors like network congestion, routing overhead, and processing time on both the client and server sides. While this implementation works well for a single client-server interaction, scaling it for multiple clients or handling larger data loads could present challenges. Better approaches to improve performance could include multithreading on both the server and router to handle concurrent connections, optimizing message-passing protocols, and introducing non-blocking I/O to reduce delays associated with waiting for responses. Additionally, implementing more advanced error-handling and load-balancing mechanisms would improve the system’s robustness and efficiency.

References

[1] M. Shacklett, “TCP/IP: What is TCP/IP and how does it work?,” *TechTarget*, Jul. 2021. https://www.techtarget.com/searchnetworking/definition/TCP-IP

[2] P. Pandey, “Understanding of Client-Server Architecture and Communication Protocols,” Medium, Apr. 19, 2023. https://medium.com/@pankaj\_pandey/understanding-of-client-server-architecture-and-communication-protocols-486884ed5f5e

[3] Gaurangjotwani, “Creating a TCP Connection Between Client-Server in Java,” Medium, Aug. 11, 2023. https://medium.com/@gaurangjotwani/creating-a-tcp-connection-between-two-servers-in-java-27fabe53deaa