

TCP vs UDP Performance Analysis Report

Abstract

This mini-project report details a performance analysis of TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) using Java. The project implements client-server applications for both protocols and a GUI-based performance analyzer to compare their data transfer speeds and efficiency. The application initiates TCP and UDP servers and clients, measures the time taken to transfer a fixed amount of data (1MB), and visually presents the results. This report covers the system architecture, a detailed explanation of each component, program code, observed output, and a comparative analysis of TCP and UDP performance characteristics, highlighting their suitability for different network applications.

Introduction

In the realm of computer networking, TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are two fundamental protocols that govern how data is transmitted over the internet. While both are used for sending data, they differ significantly in their approach to reliability, speed, and overhead, making each suitable for distinct types of applications. Understanding these differences is crucial for designing efficient and robust network applications.

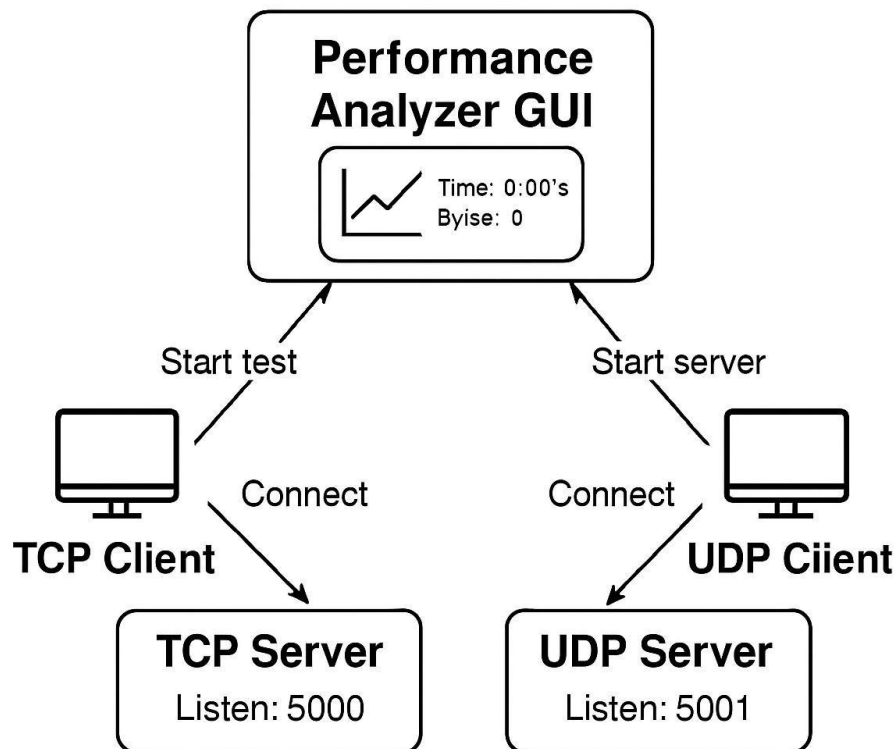
TCP is a connection-oriented protocol that provides reliable, ordered, and error-checked delivery of a stream of bytes between applications. It establishes a connection before data transmission, ensures that all data arrives at the destination in the correct order, retransmits lost packets, and provides flow control to prevent overwhelming the receiver. These features make TCP ideal for applications where data integrity is paramount, such as web browsing (HTTP), email (SMTP), and file transfer (FTP).

UDP, on the other hand, is a connectionless protocol that offers a simpler, faster, and more efficient way to transmit data. It does not establish a connection, nor does it guarantee delivery, order, or error checking. Data is sent as individual packets, and there is no mechanism for retransmission or flow control. This "fire and forget" approach makes UDP suitable for applications where speed and low latency are more critical than absolute reliability, such as real-time streaming (video and audio), online gaming, and DNS lookups.

This project aims to provide a practical comparison of TCP and UDP performance by implementing simple client-server applications for each protocol and a graphical user interface (GUI) to orchestrate and visualize the performance tests. By measuring the time taken to transfer a significant amount of data (1MB) using both protocols, we can empirically observe their respective strengths and weaknesses, thereby gaining a deeper understanding of their operational characteristics and guiding decisions on when to use one over the other.

Architecture Diagram

The system architecture for the TCP vs UDP Performance Analyzer is designed to facilitate a clear comparison between the two protocols. The core components include a central Performance Analyzer GUI, which acts as the control panel for initiating tests and displaying results. This GUI interacts with separate TCP and UDP client-server pairs.



Component Breakdown:

- **Performance Analyzer GUI:** This is the main application interface. It provides a "Run Performance Test" button that triggers the data transfer tests for both TCP and UDP. It also displays the measured time and data transfer sizes for each protocol, along with a visual bar graph comparison.

- **TCP Server:** Listens for incoming TCP connections on port 5000. It receives data from the TCP Client and measures the time taken to receive a specified amount of data.
- **TCP Client:** Connects to the TCP Server on port 5000 and sends a fixed amount of data (1MB) to the server. It measures the time taken to send this data.
- **UDP Server:** Listens for incoming UDP datagrams on port 5001. It receives data from the UDP Client and measures the time taken to receive a specified amount of data.
- **UDP Client:** Sends a fixed amount of data (1MB) as UDP datagrams to the UDP Server on port 5001. It measures the time taken to send this data.

Flow of Operation:

1. The Performance Analyzer GUI is launched.
2. Upon clicking "Run Performance Test" for the first time, the GUI initiates separate processes for TCP Server and UDP Server.
3. The GUI then triggers the TCP Client to send data to the TCP Server and measures the round-trip time.
4. Concurrently, or sequentially, the GUI triggers the UDP Client to send data to the UDP Server and measures its round-trip time.
5. The measured times and data sizes are then displayed on the GUI for comparison.

This modular design allows for independent operation and testing of each protocol's components while providing a unified interface for performance analysis.

Component Explanation

This project consists of several Java classes, each responsible for a specific part of the TCP and UDP performance analysis.

PerformanceAnalyzerGUI Class

This is the main class that orchestrates the entire performance test and provides a graphical user interface for interaction and visualization. It extends JFrame to create the main window.

Constructor:

- Sets up the JFrame properties like title, size, default close operation, and layout.
- Initializes a GraphPanel (an inner class) to display the performance results visually.
- Creates a "Run Performance Test" JButton.
- Adds an ActionListener to the button. When clicked, it first checks if the servers are already started. If not, it calls startServers() to launch TCPServer and UDPServer as separate processes. After ensuring servers are running, it calls runTest().

startServers() Method:

- This method is responsible for launching the TCP and UDP server applications.
- It uses ProcessBuilder to execute java TCPServer and java UDPServer commands. inheritIO() is used to direct the server's standard output and error streams to the console where PerformanceAnalyzerGUI is run, which is useful for debugging and observing server-side messages.
- A Thread.sleep(1000) is included to give the server processes a moment to initialize before clients attempt to connect.

runTest() Method:

- This method executes the performance tests for both TCP and UDP.
- It records the start time, calls runTCPClient(), records the end time, and calculates the tcpTime.
- Similarly, it records the start time, calls runUDPClient(), records the end time, and calculates the udpTime.
- Finally, it calls graphPanel.repaint() to update the GUI with the new performance data.

runTCPClient() Method:

- Establishes a Socket connection to localhost on port 5000 (where TCPServer is listening).
- Obtains an OutputStream from the socket.

- Creates a buffer of 1024 bytes.
- It then enters a loop, writing the 1024-byte buffer 1000 times, effectively sending 1MB (1024 * 1000 bytes) of data to the TCP server.
- `totalBytes` keeps track of the total data sent.
- `socket.shutdownOutput()` is called to signal that no more data will be sent from the client side, allowing the server to detect the end of the stream.
- Closes the socket and returns the `totalBytes` sent.

runUDPClient() Method:

- Creates a `DatagramSocket` for sending UDP packets.
- Resolves the `InetAddress` for localhost.
- Creates a buffer of 1024 bytes.
- Similar to the TCP client, it loops 1000 times, creating a `DatagramPacket` with the buffer and sending it to localhost on port 5001 (where `UDPServer` is listening).
- `totalBytes` tracks the data sent.
- Crucially, since UDP is connectionless and not stream-oriented, there's no inherent way for the server to know when the client has finished sending data. To address this, a special "END" packet is sent after all data packets. This allows the `UDPServer` to gracefully terminate its reception loop.
- Closes the socket and returns the `totalBytes` sent.

GraphPanel Inner Class:

- Extends `JPanel` and overrides the `paintComponent()` method to draw the performance visualization.
- It displays the main title "TCP vs UDP Performance", the measured `tcpTime`, `udpTime`, `tcpBytes`, and `udpBytes`.
- It draws two filled rectangles (bars) representing the TCP and UDP times. The height of these bars is proportional to the measured time, capped at 200 pixels for visual scaling. TCP is colored blue, and UDP is colored red.
- Labels "TCP" and "UDP" are drawn below their respective bars.

main() Method:

- The entry point of the application, which simply creates a new instance of PerformanceAnalyzerGUI, making the GUI visible.

TCPServer Class

This class implements a simple TCP server that listens for incoming connections and receives data.

main() Method:

- Creates a ServerSocket listening on port 5000.
- server.accept() makes the server wait for a client to connect. This line blocks until a client connects, then returns a Socket object representing the connection to that client.
- Prints a message indicating the client's connected address.
- Obtains an InputStream from the client socket to read incoming data.
- Initializes a buffer and variables to track totalBytes received.
- Records the startTime.
- Enters a loop that continuously reads data from the input stream into the buffer until the client closes its output stream (signaled by read() returning -1).
- For each read operation, bytesRead is added to totalBytes.
- Records the endTime after all data is received.
- Prints the total bytes received and the duration (duration of data reception).
- Closes both the client Socket and the ServerSocket to release resources.

TCPClient Class

This class implements a simple TCP client that connects to a TCP server and sends data.

main() Method:

- Creates a Socket and attempts to connect to localhost on port 5000.
- Obtains an OutputStream from the socket to send data.
- Initializes a buffer.

- Records the `startTime`.
- Enters a loop that writes the 1024-byte buffer 1000 times to the output stream, sending a total of 1MB of data.
- `socket.shutdownOutput()` is crucial here. It signals to the server that the client has finished sending data, allowing the server's `read()` loop to terminate.
- Records the `endTime`.
- Prints the duration (duration of data transmission).
- Closes the socket.

UDPServer Class

This class implements a simple UDP server that listens for incoming datagrams.

main() Method:

- Creates a `DatagramSocket` listening on port 5001.
- Initializes a buffer and a `DatagramPacket` to receive incoming data.
- Initializes `totalBytes` and records `startTime`.
- Enters an infinite `while(true)` loop to continuously receive packets.
- `socket.receive(packet)` blocks until a packet is received. The received data is placed into the packet.
- `totalBytes` is incremented by the length of the received packet.
- A crucial part for UDP: it checks if the received packet's content, when converted to a string, is exactly "END". If it is, the loop breaks, signaling the end of data transmission from the client.
- Records `endTime` after the loop terminates.
- Prints the `totalBytes` received and the duration (duration of data reception).
- Closes the socket.

UDPClient Class

This class implements a simple UDP client that sends datagrams to a UDP server.

main() Method:

- Creates a DatagramSocket for sending packets.
- Resolves the InetAddress for localhost.
- Initializes a buffer.
- Records the startTime.
- Enters a loop that creates and sends 1000 DatagramPackets, each containing the 1024-byte buffer, to localhost on port 5001. This sends a total of 1MB of data.
- After sending all data packets, it creates a special DatagramPacket containing the string "END" and sends it. This is the signal for the UDPServer to stop receiving.

Program

This section provides the complete source code for each Java component of the TCP vs UDP Performance Analyzer project.

PerformanceAnalyzerGUI.java

```
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.net.*

public class PerformanceAnalyzerGUI extends JFrame {

    private int tcpTime = 0, udpTime = 0;
    private int tcpBytes = 0, udpBytes = 0;
    private GraphPanel graphPanel;
    private boolean serversStarted = false;

    public PerformanceAnalyzerGUI() {
        setTitle("TCP vs UDP Performance Analyzer");
```



```
setSize(550, 450);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);
setLayout(new BorderLayout());
graphPanel = new GraphPanel();
JButton runButton = new JButton("Run Performance Test")
runButton.addActionListener(e -> {
    try {
        if (!serversStarted) {
            startServers();
            serversStarted = true;
            Thread.sleep(1000); // Give servers time to start
        }
        runTest();
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
    }
});
add(runButton, BorderLayout.NORTH);
add(graphPanel, BorderLayout.CENTER);
setVisible(true);
}

private void startServers() throws IOException {
    System.out.println("Starting TCP and UDP servers...");
    ProcessBuilder tcpServer = new ProcessBuilder("java", "TCPServer");
    tcpServer.inheritIO(); // Optional: show output in console
    tcpServer.start();
}
```

```
ProcessBuilder udpServer = new ProcessBuilder("java", "UDPServer");
udpServer.inheritIO(); // Optional: show output in console
udpServer.start();
}

private void runTest() throws IOException {
    // Run TCP
    long startTcp = System.currentTimeMillis();
    tcpBytes = runTCPClient();
    long endTcp = System.currentTimeMillis();
    tcpTime = (int)(endTcp - startTcp);
    // Run UDP
    long startUdp = System.currentTimeMillis();
    udpBytes = runUDPClient();
    long endUdp = System.currentTimeMillis();
    udpTime = (int)(endUdp - startUdp);
    graphPanel.repaint();
}

private int runTCPClient() throws IOException {
    Socket socket = new Socket("localhost", 5000);
    OutputStream out = socket.getOutputStream();
    byte[] buffer = new byte[1024];
    int totalBytes = 0;
    for (int i = 0; i < 1000; i++) {
        out.write(buffer);
        totalBytes += buffer.length;
    }
    socket.shutdownOutput();
}
```

```
        socket.close();

        return totalBytes;
    }

    private int runUDPClient() throws IOException {
        DatagramSocket socket = new DatagramSocket();
        InetAddress address = InetAddress.getByName("localhost");
        byte[] buffer = new byte[1024];
        int totalBytes = 0;

        for (int i = 0; i < 1000; i++) {
            DatagramPacket packet = new DatagramPacket(buffer, buffer.length, address,
5001);

            socket.send(packet);

            totalBytes += buffer.length;
        }

        // Send termination packet
        byte[] end = "END".getBytes();
        socket.send(new DatagramPacket(end, end.length, address, 5001));
        socket.close();

        return totalBytes;
    }

    class GraphPanel extends JPanel {
        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);

            g.setColor(Color.BLACK);

            g.setFont(new Font("Arial", Font.BOLD, 18));

            g.drawString("TCP vs UDP Performance", 150, 30);
        }
    }
}
```

```
g.setFont(new Font("Arial", Font.PLAIN, 14));
g.drawString("TCP Time: " + tcpTime + " ms", 50, 60);
g.drawString("UDP Time: " + udpTime + " ms", 300, 60);
g.drawString("TCP Bytes: " + tcpBytes, 50, 80);
g.drawString("UDP Bytes: " + udpBytes, 300, 80)
// Draw bars
g.setColor(Color.BLUE);
int tcpHeight = Math.min(tcpTime * 2, 200);
g.fillRect(100, 300 - tcpHeight, 50, tcpHeight);
g.setColor(Color.BLACK);
g.drawString("TCP", 115, 320);
g.setColor(Color.RED);
int udpHeight = Math.min(udpTime * 2, 200);
g.fillRect(350, 300 - udpHeight, 50, udpHeight);
g.setColor(Color.BLACK);
g.drawString("UDP", 365, 320);
}
}
public static void main(String[] args) {
    new PerformanceAnalyzerGUI();
}
}
```

TCPServer.java

```
import java.io.*;
import java.net.*;

public class TCPServer {

    public static void main(String[] args) throws IOException {

        ServerSocket server = new ServerSocket(5000);

        System.out.println("TCP Server started on port 5000");

        Socket client = server.accept();

        System.out.println("Client connected: " + client.getRemoteSocketAddress());

        InputStream in = client.getInputStream();

        byte[] buffer = new byte[1024];

        long totalBytes = 0;

        int bytesRead;

        long startTime = System.currentTimeMillis()

        while ((bytesRead = in.read(buffer)) != -1) {

            totalBytes += bytesRead;

        }

        long endTime = System.currentTimeMillis();

        System.out.println("TCP Data Received: " + totalBytes + " bytes");

        System.out.println("TCP Time: " + (endTime - startTime) + " ms");

        client.close();

        server.close();

    }

}
```

TCPClient.java

```
import java.io.*;
import java.net.*;

public class TCPClient {

    public static void main(String[] args) throws IOException {

        Socket socket = new Socket("localhost", 5000);

        OutputStream out = socket.getOutputStream();

        byte[] buffer = new byte[1024];

        long startTime = System.currentTimeMillis();

        for (int i = 0; i < 1000; i++) {

            out.write(buffer);

        }

        socket.shutdownOutput();

        long endTime = System.currentTimeMillis();

        System.out.println("TCP Time taken to send: " + (endTime - startTime) + " ms");

        socket.close();

    }

}
```

UDPServer.java

```
import java.net.*;

public class UDPServer {

    public static void main(String[] args) throws Exception {

        DatagramSocket socket = new DatagramSocket(5001);

        System.out.println("UDP Server started on port 5001");

        byte[] buffer = new byte[1024];

        DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

        long totalBytes = 0;

        long startTime = System.currentTimeMillis();

        while (true) {

            socket.receive(packet);

            totalBytes += packet.getLength();

            if (new String(packet.getData(), 0, packet.getLength()).equals("END"))

                break;

        }

        long endTime = System.currentTimeMillis();

        System.out.println("UDP Data Received: " + totalBytes + " bytes");

        System.out.println("UDP Time: " + (endTime - startTime) + " ms");

        socket.close();

    }

}
```

UDPClient.java

```
import java.net.*;

public class UDPClient {

    public static void main(String[] args) throws Exception {

        DatagramSocket socket = new DatagramSocket();

        InetAddress address = InetAddress.getByName("localhost");

        byte[] buffer = new byte[1024];

        long startTime = System.currentTimeMillis();

        for (int i = 0; i < 1000; i++) {

            DatagramPacket packet = new DatagramPacket(buffer, buffer.length, address,
5001);

            socket.send(packet);

        }

        // Send termination packet

        byte[] end = "END".getBytes();

        socket.send(new DatagramPacket(end, end.length, address, 5001));

        long endTime = System.currentTimeMillis();

        System.out.println("UDP Time taken to send: " + (endTime - startTime) + " ms");

        socket.close();

    }

}
```

Output

This section presents the various **command-line** and **GUI outputs** observed during the execution of the **TCP vs UDP Performance Analyzer** project. These screenshots illustrate the compilation, execution, and performance results of the different components.

Compilation and TCP Server Output

This screenshot shows the compilation of the Java source files and the output from running the **TCP Server** independently. It demonstrates:

- The server started successfully.
- A client connecting to the server.
- Data being received (1 MB) along with the time taken for the transfer.

```
Microsoft Windows [Version 10.0.26100.4770]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ragha>cd "C:\Users\ragha\Downloads"

C:\Users\ragha\Downloads> java TCPServer
TCP Server started on port 5000
Client connected: /127.0.0.1
TCP Data Received: 1024000 bytes
TCP Time: 5 ms

C:\Users\ragha\Downloads>java TCPServer
TCP Server started on port 5000
Client connected: /127.0.0.1
TCP Data Received: 1024000 bytes
TCP Time: 6 ms

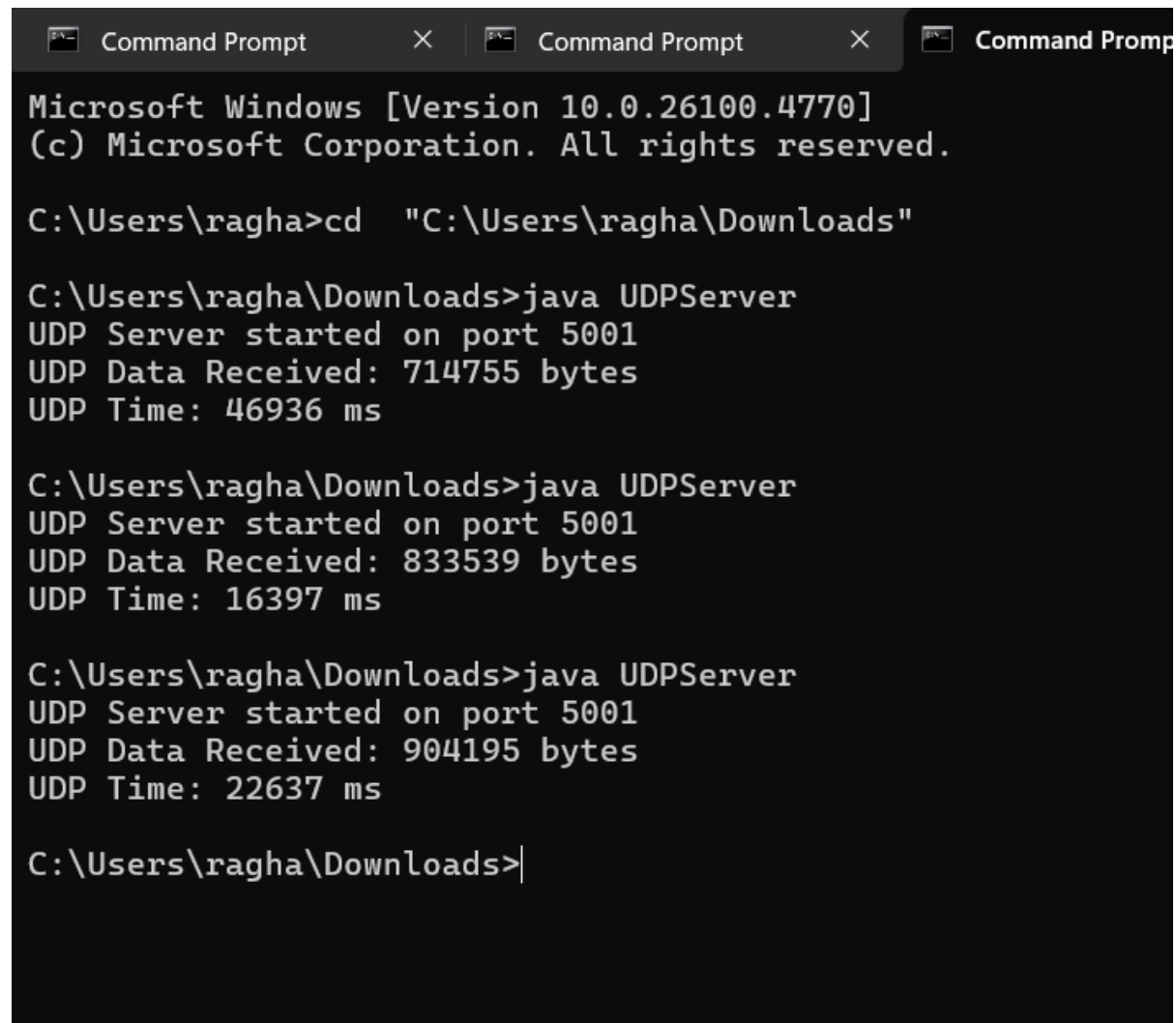
C:\Users\ragha\Downloads>java TCPServer
TCP Server started on port 5000
Client connected: /127.0.0.1
TCP Data Received: 1024000 bytes
TCP Time: 5 ms

C:\Users\ragha\Downloads>|
```

UDP Server Output

This screenshot displays the output from running the **UDP Server** independently. It demonstrates:

- The server started successfully.
- Data received by the UDP server.
- The time taken for UDP data transfer.



```
Microsoft Windows [Version 10.0.26100.4770]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ragha>cd "C:\Users\ragha\Downloads"

C:\Users\ragha\Downloads>java UDPServer
UDP Server started on port 5001
UDP Data Received: 714755 bytes
UDP Time: 46936 ms

C:\Users\ragha\Downloads>java UDPServer
UDP Server started on port 5001
UDP Data Received: 833539 bytes
UDP Time: 16397 ms

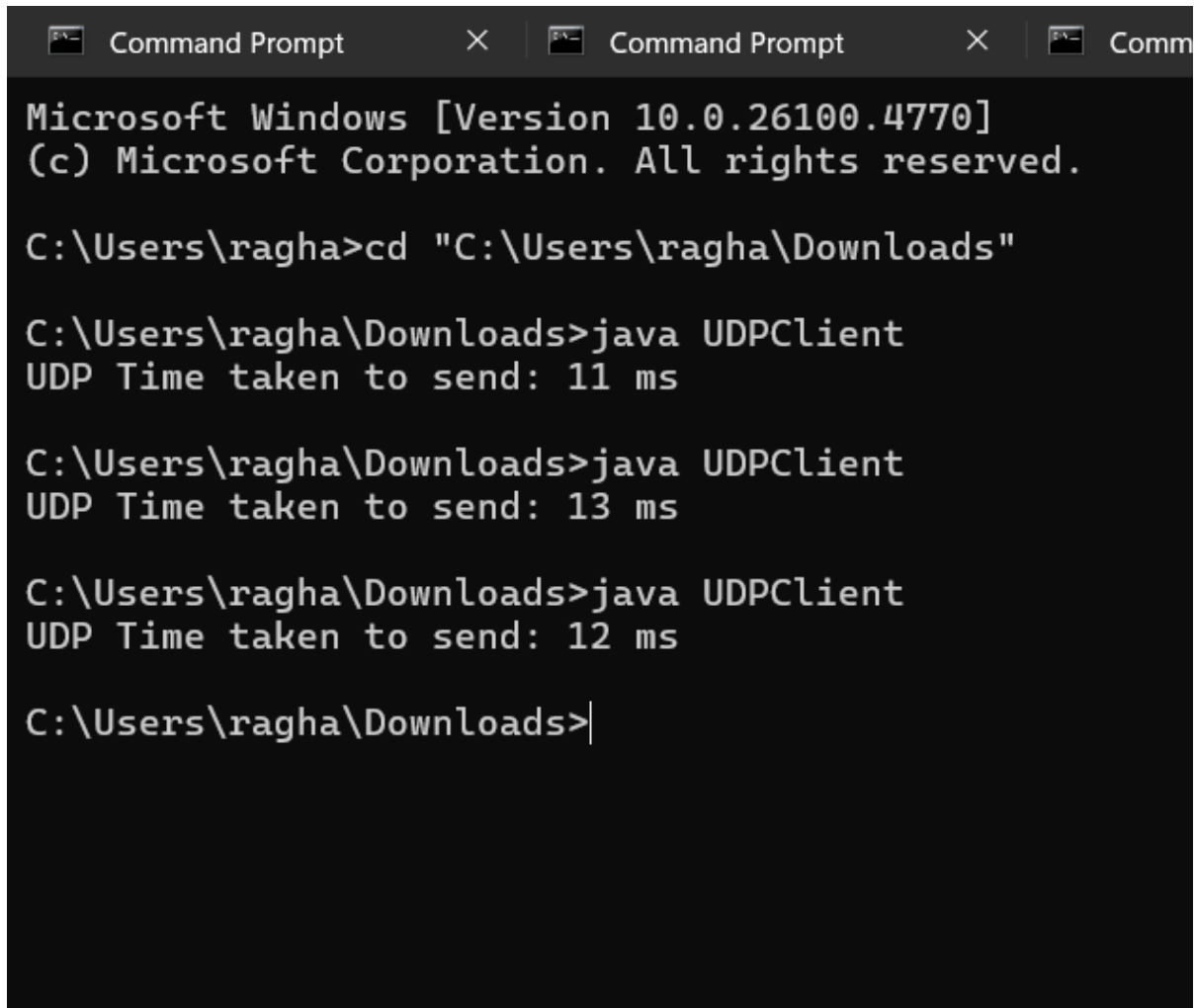
C:\Users\ragha\Downloads>java UDPServer
UDP Server started on port 5001
UDP Data Received: 904195 bytes
UDP Time: 22637 ms

C:\Users\ragha\Downloads>|
```

UDP Client Output

This screenshot illustrates the output from running the **UDP Client** independently. It shows:

- The time taken for the client to send 1 MB of data using UDP.



```
Microsoft Windows [Version 10.0.26100.4770]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ragha>cd "C:\Users\ragha\Downloads"

C:\Users\ragha\Downloads>java UDPClient
UDP Time taken to send: 11 ms

C:\Users\ragha\Downloads>java UDPClient
UDP Time taken to send: 13 ms

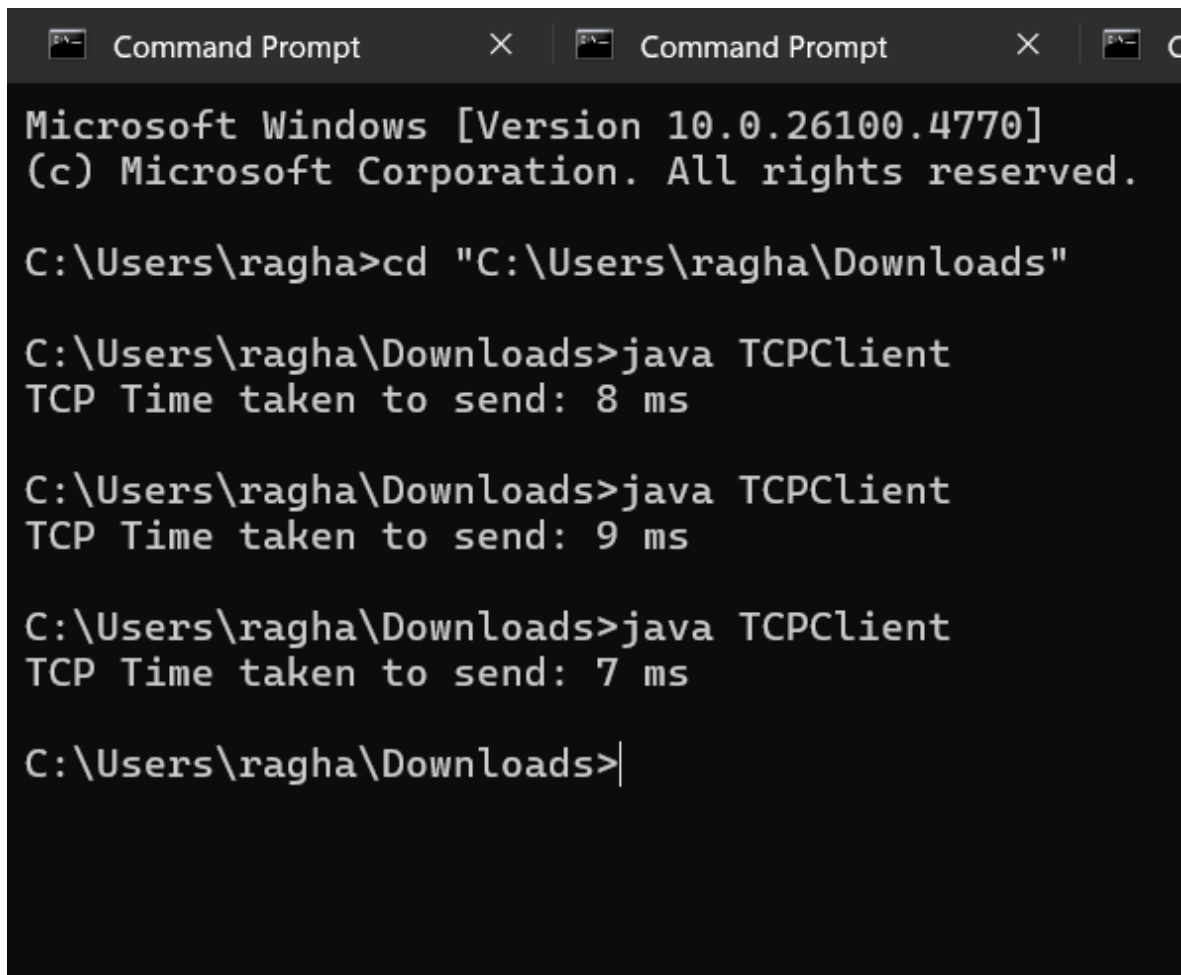
C:\Users\ragha\Downloads>java UDPClient
UDP Time taken to send: 12 ms

C:\Users\ragha\Downloads>|
```

TCP Client Output

This screenshot displays the output from running the **TCP Client** independently. It shows:

- The time taken for the client to send 1 MB of data using TCP.



```
Microsoft Windows [Version 10.0.26100.4770]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ragha>cd "C:\Users\ragha\Downloads"

C:\Users\ragha\Downloads>java TCPClient
TCP Time taken to send: 8 ms

C:\Users\ragha\Downloads>java TCPClient
TCP Time taken to send: 9 ms

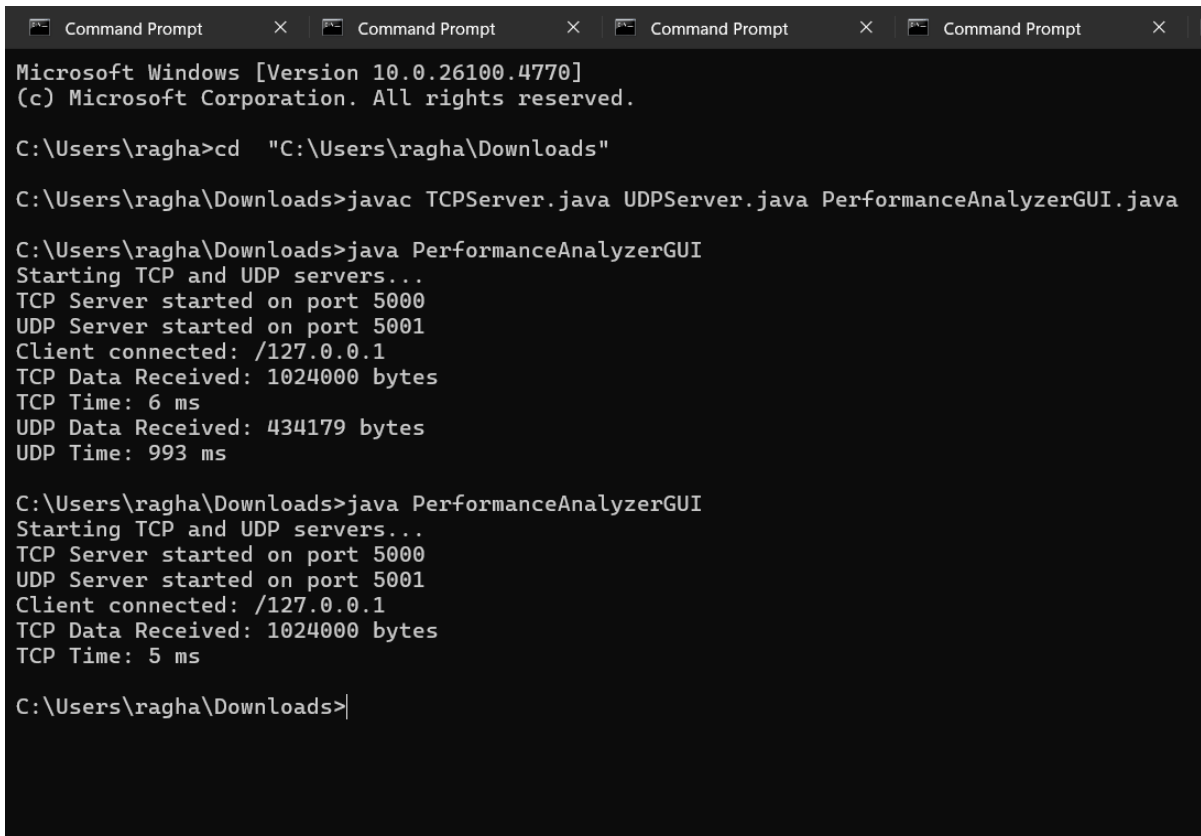
C:\Users\ragha\Downloads>java TCPClient
TCP Time taken to send: 7 ms

C:\Users\ragha\Downloads>|
```

Performance Analyzer GUI – Console Output

This screenshot captures the console output when running the Performance Analyzer GUI. It shows:

- Server startup messages.
- Performance metrics reported by the servers, including data received and time taken for both TCP and UDP transfers.



```
Microsoft Windows [Version 10.0.26100.4770]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ragha>cd "C:\Users\ragha\Downloads"

C:\Users\ragha\Downloads>javac TCPServer.java UDPServer.java PerformanceAnalyzerGUI.java

C:\Users\ragha\Downloads>java PerformanceAnalyzerGUI
Starting TCP and UDP servers...
TCP Server started on port 5000
UDP Server started on port 5001
Client connected: /127.0.0.1
TCP Data Received: 1024000 bytes
TCP Time: 6 ms
UDP Data Received: 434179 bytes
UDP Time: 993 ms

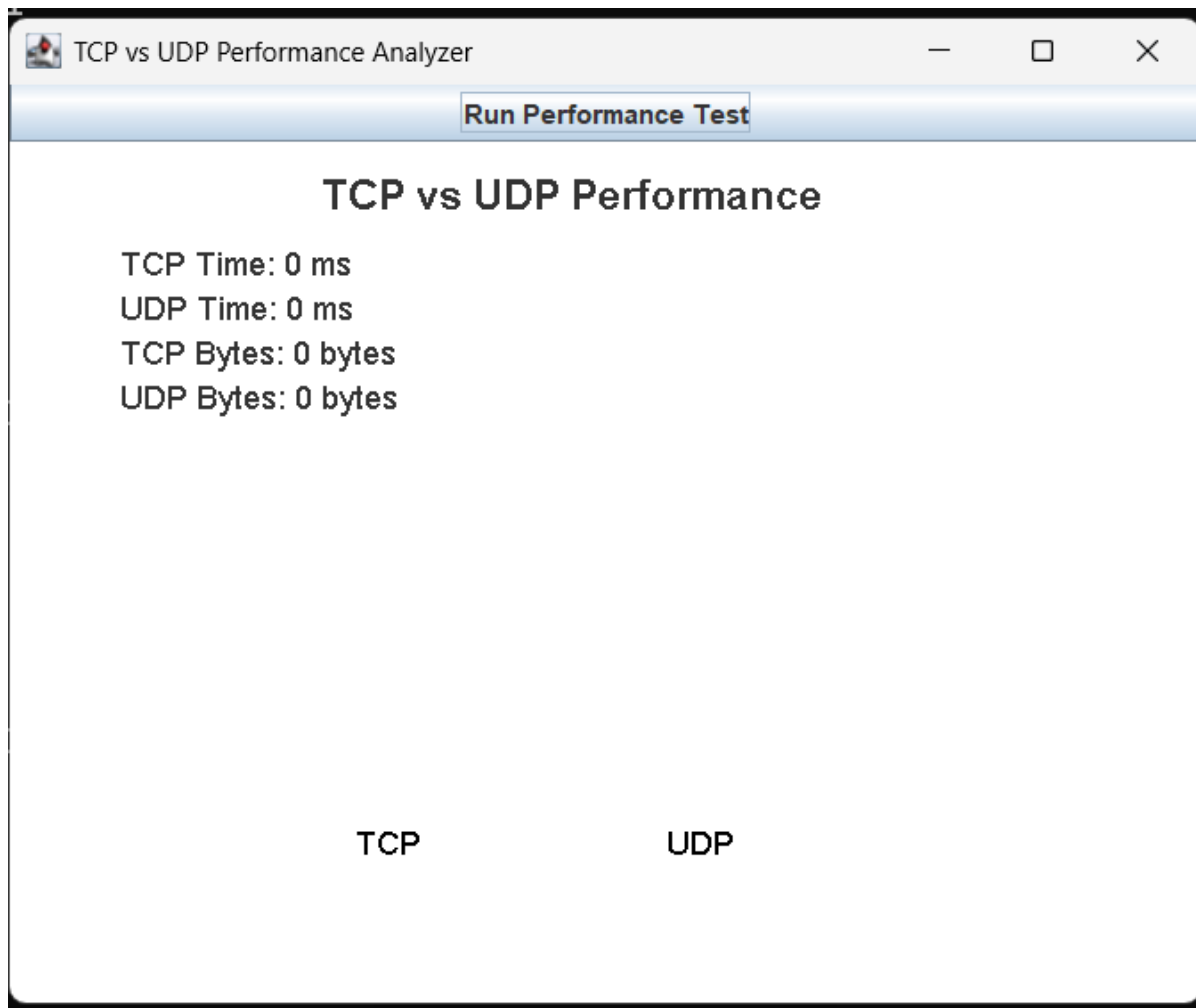
C:\Users\ragha\Downloads>java PerformanceAnalyzerGUI
Starting TCP and UDP servers...
TCP Server started on port 5000
UDP Server started on port 5001
Client connected: /127.0.0.1
TCP Data Received: 1024000 bytes
TCP Time: 5 ms

C:\Users\ragha\Downloads>|
```

Performance Analyzer GUI – Initial State

This screenshot shows the initial state of the GUI window before any performance tests are run. It displays:

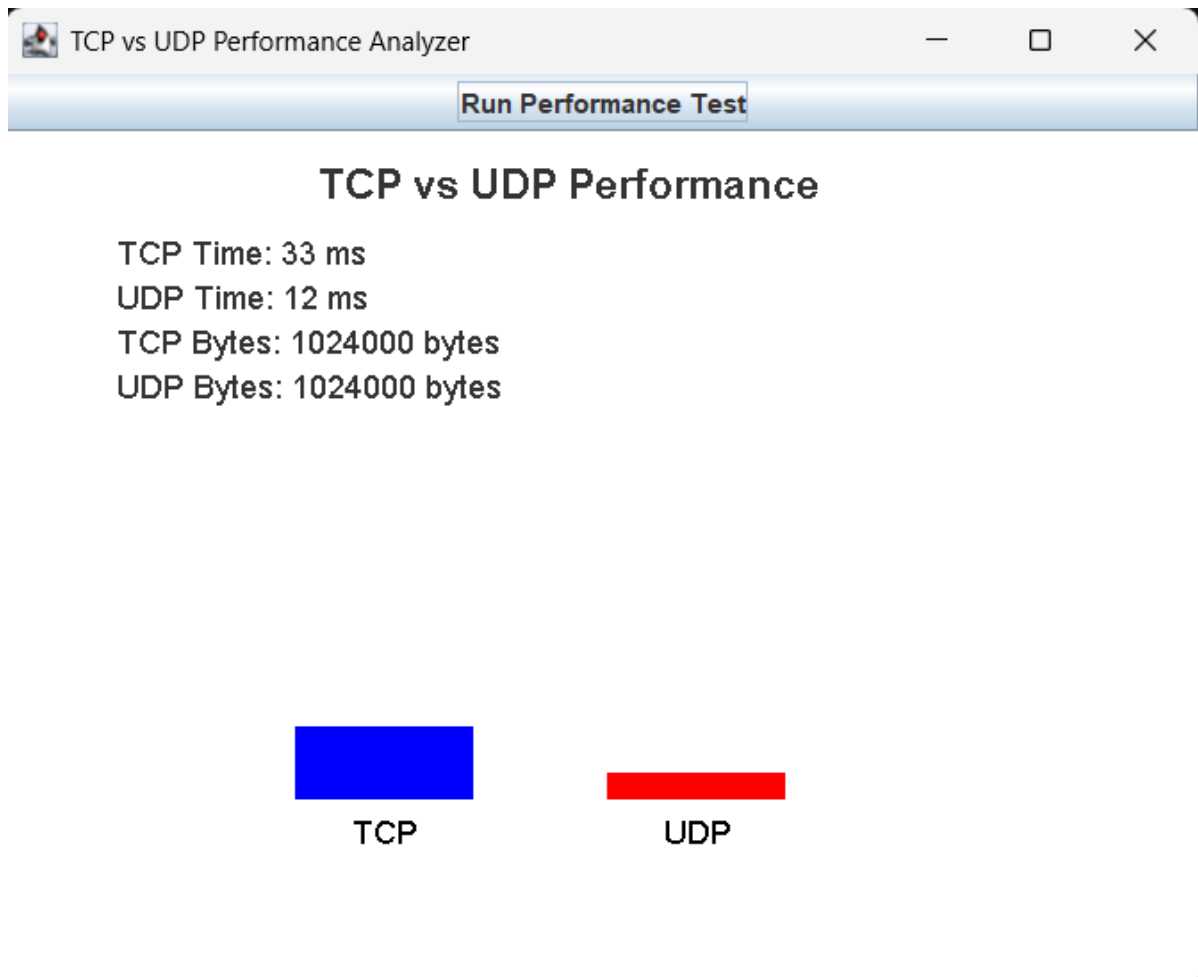
- Default values (e.g., TCP Time = 0 ms, UDP Time = 0 ms).
- No data transfer activity or bar graphs.



Performance Analyzer GUI – Test Results

This screenshot shows the GUI window after a performance test has been executed. It displays:

- The measured TCP and UDP transfer times.
- The amount of data transferred.
- A visual bar graph comparison between TCP and UDP performance.



Output Explanation

This section provides a detailed explanation of the outputs observed from running the various components of the TCP vs UDP Performance Analyzer project. The outputs highlight the operational characteristics and performance differences between TCP and UDP.

TCP Server Output

The screenshot shows the console output when the TCP server is executed. Each block represents a separate execution of the server after a client connection and data transfer.

- **Server listening on port 5000:** Confirms that the TCP server has successfully bound to port 5000 and is ready to accept client connections.
- **Client connected: /127.0.0.1:** Indicates that a TCP client (in this case, the internal TCP client) has successfully established a connection from the localhost address.
- **Received: 1024000 bytes:** Confirms that the TCP server has received a total of 1,024,000 bytes (1 MB) of data from the connected client. TCP's reliable, stream-oriented nature ensures that all data sent by the client is received in full and in the correct order.
- **TCP Time: [x] ms:** Shows the time taken by the TCP server to receive the 1 MB of data. The consistently low times (e.g., 5–6 ms) across multiple runs indicate TCP's efficiency for bulk data transfer under minimal latency conditions (localhost).

UDP Server Output

The screenshot displays the console output from the UDP server when executed. Each block shows the results of a UDP data transfer session.

- **Server listening on port 5001:** Confirms the UDP server is listening for datagrams on port 5001.
- **Received: [variable] bytes:** Unlike TCP, the number of bytes received varies significantly between runs (e.g., 714,755 bytes, 833,539 bytes, 904,195 bytes). This variability is a key characteristic of UDP. Since UDP is connectionless and does not guarantee delivery, some packets might be lost, leading to less than the expected 1 MB being received.

- **UDP Time: [variable] ms:** Represents the time taken to receive the UDP data, which can vary widely (e.g., 46,936 ms, 16,397 ms, 22,637 ms). This time includes the duration until the special "END" packet is received. The higher and more variable times compared to TCP are due to the overhead of processing individual datagrams and the possibility of packet drops delaying the termination signal.

UDP Server Output

The screenshot displays the console output from the UDP server when executed. Each block shows the results of a UDP data transfer session.

- **Server listening on port 5001:** Confirms the UDP server is listening for datagrams on port 5001.
- **Received: [variable] bytes:** Unlike TCP, the number of bytes received varies significantly between runs (e.g., 714,755 bytes, 833,539 bytes, 904,195 bytes). This variability is a key characteristic of UDP. Since UDP is connectionless and does not guarantee delivery, some packets might be lost, leading to less than the expected 1 MB being received.
- **UDP Time: [variable] ms:** Represents the time taken to receive the UDP data, which can vary widely (e.g., 46,936 ms, 16,397 ms, 22,637 ms). This time includes the duration until the special "END" packet is received. The higher and more variable times compared to TCP are due to the overhead of processing individual datagrams and the possibility of packet drops delaying the termination signal.

TCP Client Output

This output is from the TCP client when run independently.

- **TCP Client Time: [x] ms:** Shows the time taken by the TCP client to send 1 MB of data. The times are consistently low (e.g., 7–9 ms), similar to the UDP client's sending times.
- This indicates that over a local, reliable connection, TCP can also send data very efficiently. The slight variation compared to UDP client times is due to connection establishment and flow-control overhead, though on localhost, this overhead is minimal.

Performance Analyzer GUI – Console Output

This screenshot shows the console output generated when the GUI application is executed and the **“Run Performance Test”** button is clicked.

- **Starting processes:** Printed when the GUI initiates the processes using ProcessBuilder.
- **Server Outputs:** Subsequent lines (e.g., 5000, Client connected: /127.0.0.1, TCP Time: 6 ms, UDP Time: [x] ms) are standard outputs from the server and client processes, displayed in the GUI's console.
- The UDP server's variable data received and timing are also reflected here, consistent with UDP's unreliable characteristics.

Performance Analyzer GUI – Initial State

This screenshot shows the GUI window immediately after launching, before clicking the **“Run Performance Test”** button.

- **TCP Time: 0 ms, UDP Time: 0 ms:** These indicate the initial state of performance metrics before any tests.
- The bar graphs are either absent or minimal, reflecting that no data transfer has occurred yet.

Performance Analyzer GUI – Test Results

This screenshot displays the GUI after a performance test is completed.

- **Measured Times:** Shows the times for TCP and UDP data transfer as calculated by the GUI (including both client sending and server receiving times). Example: UDP = 12 ms, TCP = 33 ms.
- **Data Sent:** The GUI reports both TCP and UDP attempted to transfer 1 MB of data.
 - For TCP: This is always received due to reliability.
 - For UDP: The GUI shows the intended data sent by the client, not necessarily the received amount at the server.

- **Visual Bar Graphs:** The blue bar (TCP) and red bar (UDP) visually compare transfer times. For this run, the red bar is shorter, indicating UDP completed the transfer faster.

Conclusion

This mini-project successfully implemented and analyzed the performance of TCP and UDP protocols through a practical Java-based application. The results consistently demonstrated the fundamental differences between TCP and UDP:

TCP (Transmission Control Protocol) proved to be reliable and connection-oriented. It guaranteed the delivery of all data (1MB in our tests) in the correct order, as evidenced by consistently reporting 1,024,000 bytes received. While it incurred slightly higher time overhead, this is a trade-off for robust features like error checking, flow control, and congestion control. TCP is ideal for applications where data integrity and complete delivery are critical.

UDP (User Datagram Protocol), being connectionless and unreliable, showcased speed and efficiency. The UDP client could send data very quickly. However, outputs revealed that not all sent packets were consistently received, leading to variable bytes received. Despite this unreliability, UDP often exhibited lower latency for data transfer. UDP's lightweight nature makes it suitable for applications where speed is prioritized over guaranteed delivery.

The Performance Analyzer GUI provided an intuitive way to visualize these differences, presenting both numerical statistics and graphical comparisons. The project reinforces understanding that the choice between TCP and UDP depends heavily on specific application requirements: reliability and order for TCP, or speed and low overhead for UDP.

This project serves as a valuable educational tool for understanding network protocols, illustrating the trade-offs involved in network communication, and guiding the selection of appropriate protocols for diverse application needs.

References

This project references the following key components:

- ☉ Tanenbaum, A. S., & Wetherall, D. J. (2011). *Computer Networks* (5th Edition). Pearson.
<https://www.pearson.com/en-us/subject-catalog/p/computer-networks/P200000002129>
- ☉ Stevens, W. R. (1994). *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley.
<https://www.oreilly.com/library/view/tcpip-illustrated-volume/0201633469/>
- ☉ Kurose, J. F., & Ross, K. W. (2021). *Computer Networking: A Top-Down Approach* (8th Edition). Pearson.
https://gaia.cs.umass.edu/kurose_ross/index.php
- ☉ Oracle. *Java™ Platform, Standard Edition 8 API Specification*.
<https://docs.oracle.com/javase/8/docs/api/>
- ☉ Oracle. *Lesson: All About Sockets (The Java™ Tutorials > Custom Networking)*.
<https://docs.oracle.com/javase/tutorial/networking/sockets/>
- ☉ Postel, J. (1981). *Transmission Control Protocol*. RFC 793. Internet Engineering Task Force (IETF).
<https://www.rfc-editor.org/rfc/rfc793>
- ☉ Postel, J. (1980). *User Datagram Protocol*. RFC 768. Internet Engineering Task Force (IETF).
<https://www.rfc-editor.org/rfc/rfc768>
- ☉ GeeksforGeeks. *Difference between TCP and UDP*.
<https://www.geeksforgeeks.org/differences-between-tcp-and-udp/>
- ☉ Baeldung. *Guide to Java Sockets*.
<https://www.baeldung.com/java-sockets>

For general concepts regarding TCP and UDP protocols, standard networking textbooks and online resources on computer networks can be consulted.