Load Balancer Content Providers

Description

This project demonstrates a network application that showcases static and dynamic load balancer implementations using Weighted Round Robin and Round Robin algorithms over both UDP and TCP protocols. The application highlights two main functionalities:

- Video streaming via the UDP port.
- Text file transfer via the TCP port.

The project aims to illustrate how load balancing can be effectively managed in a network environment, ensuring efficient resource utilization and optimized performance.

Features

- **Static Load Balancer with Round Robin Algorithm**: Implements a static load balancing mechanism where the incoming requests are distributed equally among available servers using the Round Robin algorithm. This ensures a fair distribution of load but does not account for server performance or load.
- **Dynamic Load Balancer with Weighted Round Robin Algorithm**: Utilizes a dynamic load balancing strategy where servers are assigned weights based on their capabilities or current load. The Weighted Round Robin algorithm distributes requests proportionally, considering the server weights, leading to more efficient load distribution.
- **Video Streaming from Server to Client**: Demonstrates a server-client model where video content is streamed from the server to the client over a UDP connection. This part of the project highlights the low-latency benefits of using UDP for streaming applications.
- **TCP Server-Client Model for File Transfer**: Implements a reliable file transfer mechanism over TCP. Text files are sent from the server to the client, ensuring data integrity and delivery reliability inherent in the TCP protocol.
- **Simple Server Competition**: The project includes a basic server competition scenario where multiple servers compete to handle incoming client requests based on the implemented load balancing algorithms.

Requirements

- **Java Development Kit (JDK) 8 or Higher**: The project is developed in Java and requires JDK 8 or higher to compile and run. It leverages Java's socket programming capabilities for network communication.
- **Basic Understanding of Socket Programming**: Users should have a fundamental knowledge of socket programming concepts, including how to create server and client sockets, handle connections, and manage data transmission over the network.

Common Issues

- **Port Conflict Due to Full Ports**: One common issue encountered in this project is the problem of port conflicts. After a certain number of transactions, the ports may become congested with pending transactions, leading to conflicts. This issue arises because ports may remain occupied for some time after the connection is closed, due to the TIME_WAIT state in TCP connections. It is essential to manage port usage and connection lifetimes carefully to mitigate this problem.
    - **Solution**: Implementing proper connection handling techniques, such as reusing existing connections, setting appropriate socket options (e.g., SO_REUSEADDR), and ensuring timely release of resources can help alleviate port conflicts. Additionally,

monitoring and managing port usage dynamically can prevent the accumulation of transactions on specific ports.

By incorporating these features and considerations, the project provides a comprehensive demonstration of load balancing in network applications, leveraging both UDP and TCP protocols for different types of data transmission.