**Network Load Balancer**

**Submitted By**

Muhammad Abdullah

22k-5156

Sameer Ahmed

22k-5193

**Department of Computer Science**

**National University of Computer & Emerging Sciences**

# Motivation

The increasing complexity and scale of distributed applications demand efficient request handling and fault tolerance. This project was initiated to simulate and visualize how load balancing systems operate, monitor server health, and handle failovers. It reflects our practical interest in real-world networking systems and prepares us for scenarios faced in modern web infrastructure.

# Overview

This project replicates the foundational principles of real-world load balancers such as HAProxy and Nginx using Python, from scratch. It offers an interactive and visual simulation for understanding dynamic traffic routing, failure detection, and recovery across multiple servers.  
  
This simulation empowers students to visualize request handling in distributed systems, gain insight into backend traffic flow, and experiment with load balancing strategies. Practically, these principles are widely used in production systems like web servers, databases, and cloud infrastructure. It holds academic value as a bridge between theoretical networking concepts and their applied implementation.  
  
If successful, this project can serve as an educational tool for students studying distributed systems, DevOps, or networking. It encourages experimentation and promotes understanding of key architectural decisions in scalable systems.

# Description of the Project

The project simulates a network load balancer that:  
- Receives requests from multiple simulated clients.  
- Distributes them across several backend servers using load balancing algorithms like Round Robin, Least Connections, and IP Hashing.  
- Detects and handles server failures using periodic health checks.  
- Redirects traffic to alternative healthy servers if one becomes unresponsive.  
  
Users interact with the system through a GUI where they can:  
- Select the load balancing algorithm.  
- Choose how many clients and servers to simulate.  
- Set the frequency of health checks.  
- Define how often servers fail (as a percentage).  
- Monitor server responses and logs in real time.

# Background of the Project

The project builds upon concepts studied in Computer Networks, especially regarding:  
- Client-server models  
- Load balancing techniques  
- Health monitoring in distributed systems  
- TCP socket simulation  
- Failover mechanisms  
  
It also incorporates:  
- Web interface in HTML + Tailwind via Flask  
- Python threading for concurrency  
- Logging for observability  
  
References:  
- Kurose, J. F., & Ross, K. W. (2020). Computer Networking: A Top-Down Approach.  
- Official Python Flask documentation.  
- Real-world documentation from HAProxy and Nginx for algorithm concepts.

# Project Category

Product-based academic simulation project designed for educational and demonstration purposes.

# Features / Scope / Modules

1. Traffic Distribution using Load Balancing Algorithms  
2. Server Health Monitoring  
3. Dynamic Server Failure Simulation  
4. Client Request Simulation  
5. Modem and User-Friendly GUI  
6. Real-Time Logging + File Logging

# Project Planning

Week 1: Research load balancing algorithms and architecture  
Week 2: Build basic request routing system with threading  
Week 3: Implement Round Robin, Least Connections, IP Hash  
Week 4: Add server health checks and failure logic  
Week 5: Create full-featured GUI  
Week 6: Add delay, logging, server status display, and tuning options  
Week 7: Testing, logging file integration, and final report preparation

Each group member contributed to code, debugging, and documentation.

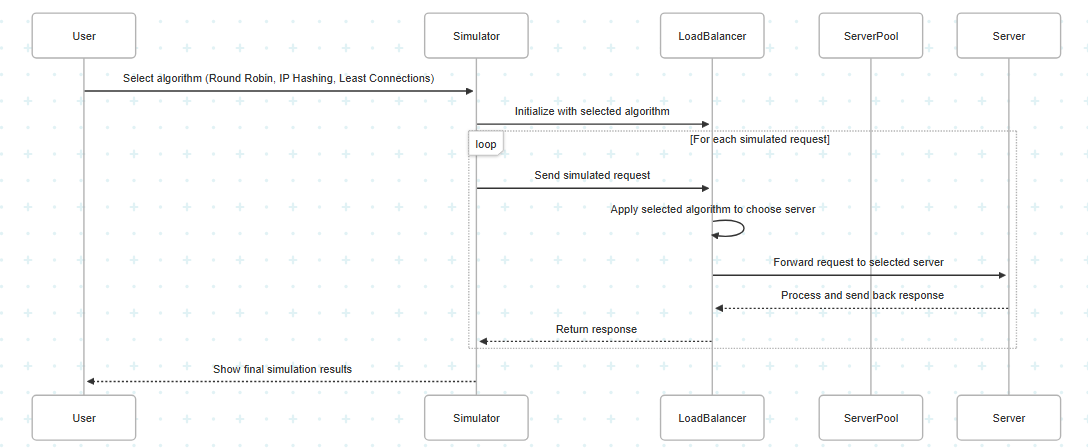
# Project Feasibility

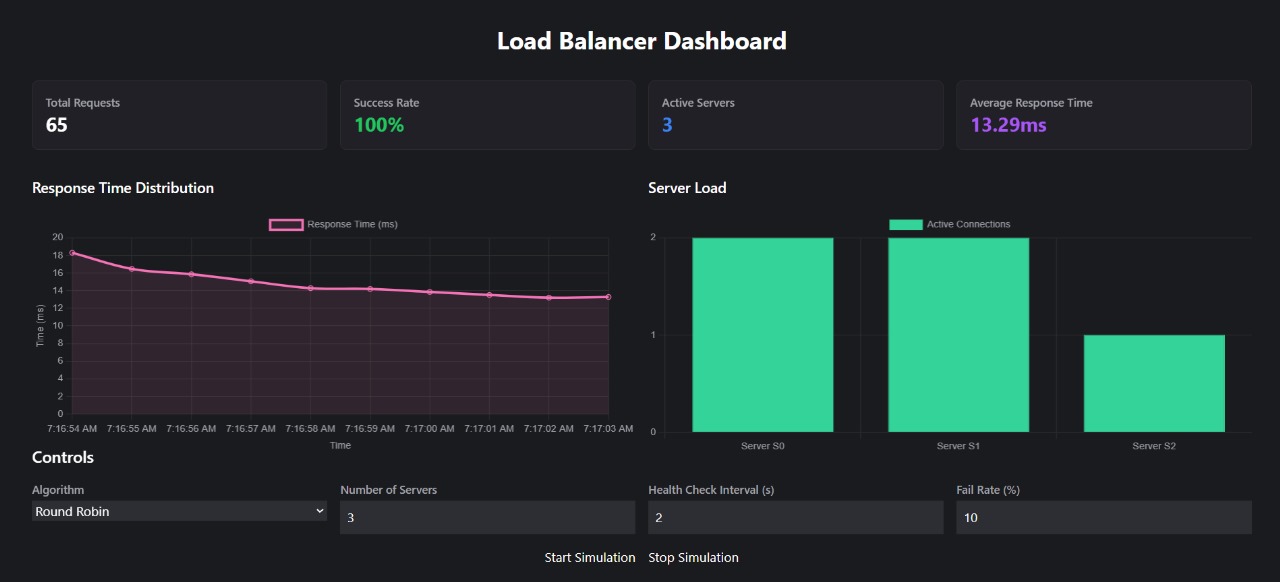
Technical Feasibility:  
All components are achievable using standard Python libraries (socket, threading, flask, logging). The architecture supports future extensions.  
  
Economic Feasibility:  
No external resources or licensing are required. Development was done on personal/student laptops with open-source tools.  
  
Schedule Feasibility:  
Completed within 6–7 weeks as per plan. Each week was used to implement and test modular components.

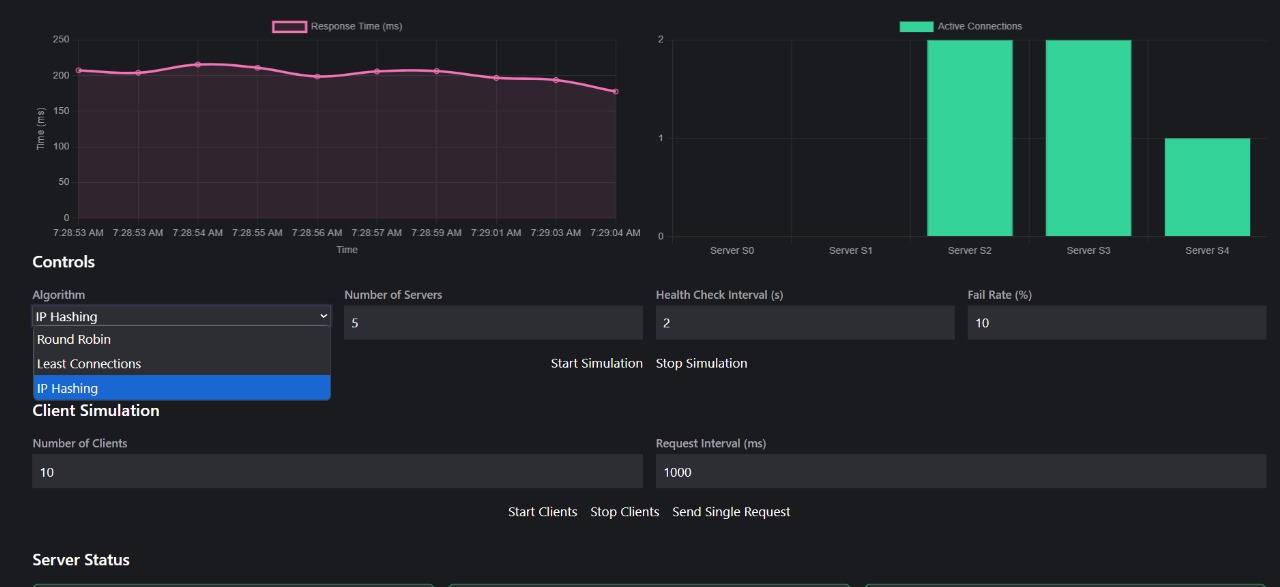
# Hardware and Software Requirements

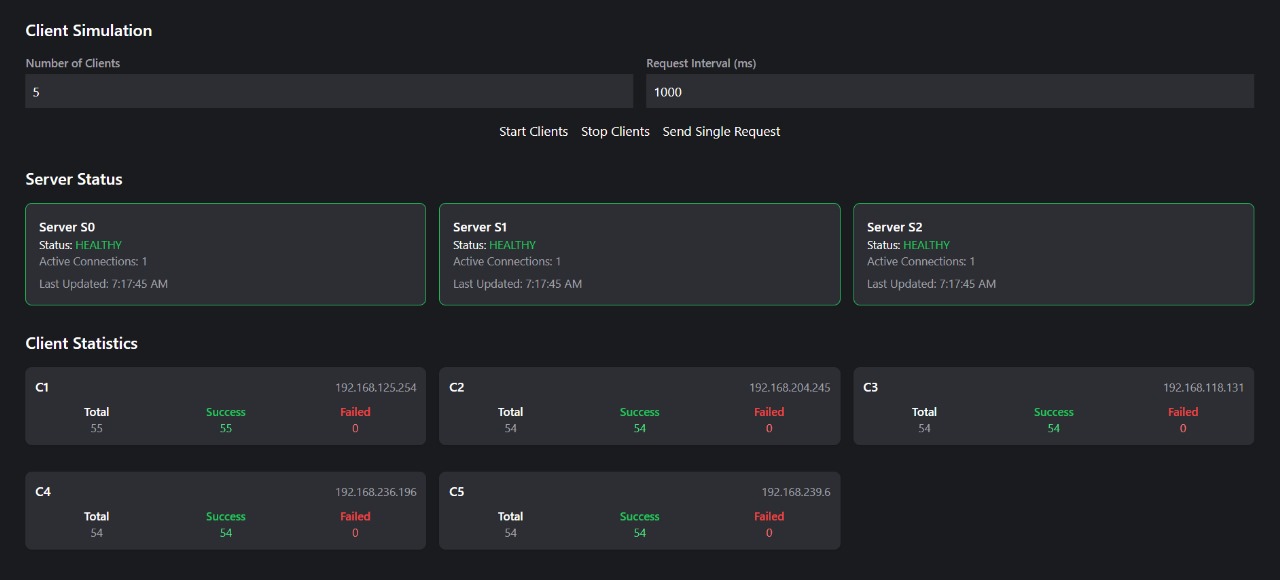
Software:  
- Python 3.x  
- HTML + Tailwind via Flask for GUI  
- threading, socket, logging (standard libraries)  
  
Hardware:  
- Any standard Windows

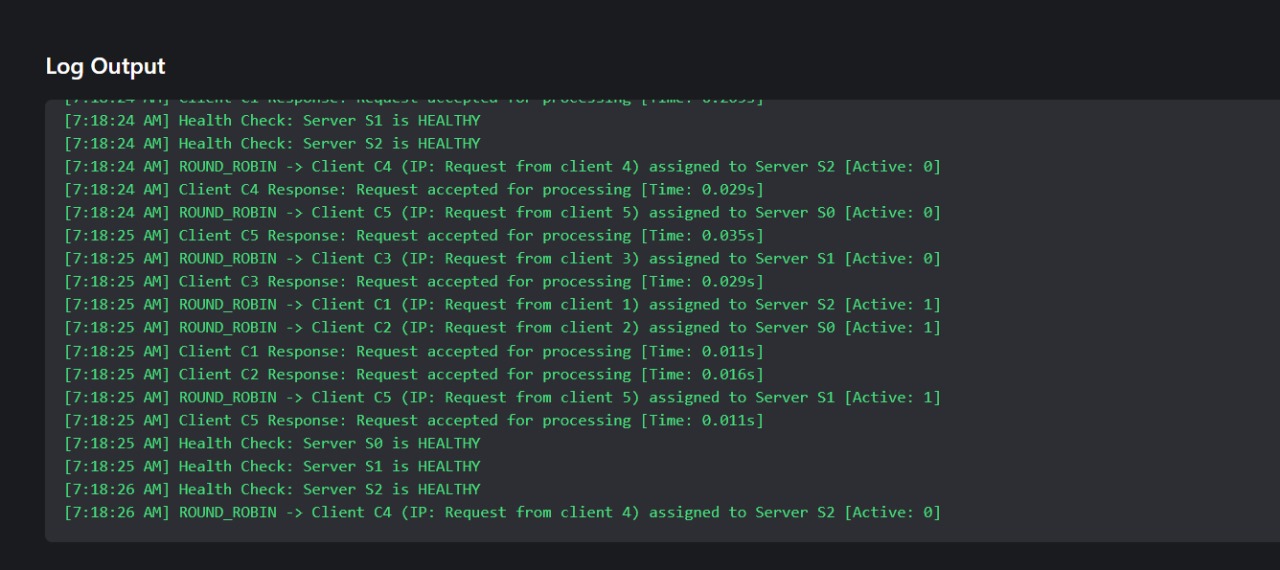
# Diagrammatic Representation of the Overall System











# References

[1] Kurose, J. F., & Ross, K. W. (2020). Computer Networking: A Top-Down Approach. Pearson.  
[2] Python Official Docs – https://docs.python.org/3/library/  
[3] NGINX Load Balancing Docs – https://docs.nginx.com/nginx/admin-guide/load-balancer/  
[4] HAProxy Documentation – https://www.haproxy.org/  
[5] GeeksForGeeks – Socket Programming in Python