Fastchat : Chmod+x

Ayush Agarwal, Sankalan Baidya, Soham Joshi

IIT Bombay

August 2022

Table of Contents

Introduction

Hello, this project, named “FastChat” is a course project of the course CS251 offered in the autumn semester at IIT Bombay. We have designed a working chat application complete with a GUI and a backend. It is complete with the ability to join different groups, send images, E2E encryption and is compatible with the three major platforms of modern computers : MacOS, Windows as well as Linux (Debian). So, without further ado let’s dive in !

GUI

For the purposes of GUI, we have used the tkinter module in python 3.8. Tkinter facilitates a user interface complete with buttons for creating and joining groups, sending image files stored on your machine, switching between different groups. Moreover, the tkinter interface is complete with a chatbox for viewing messages and a chatwindow for sending messages.

GUI : Implementation

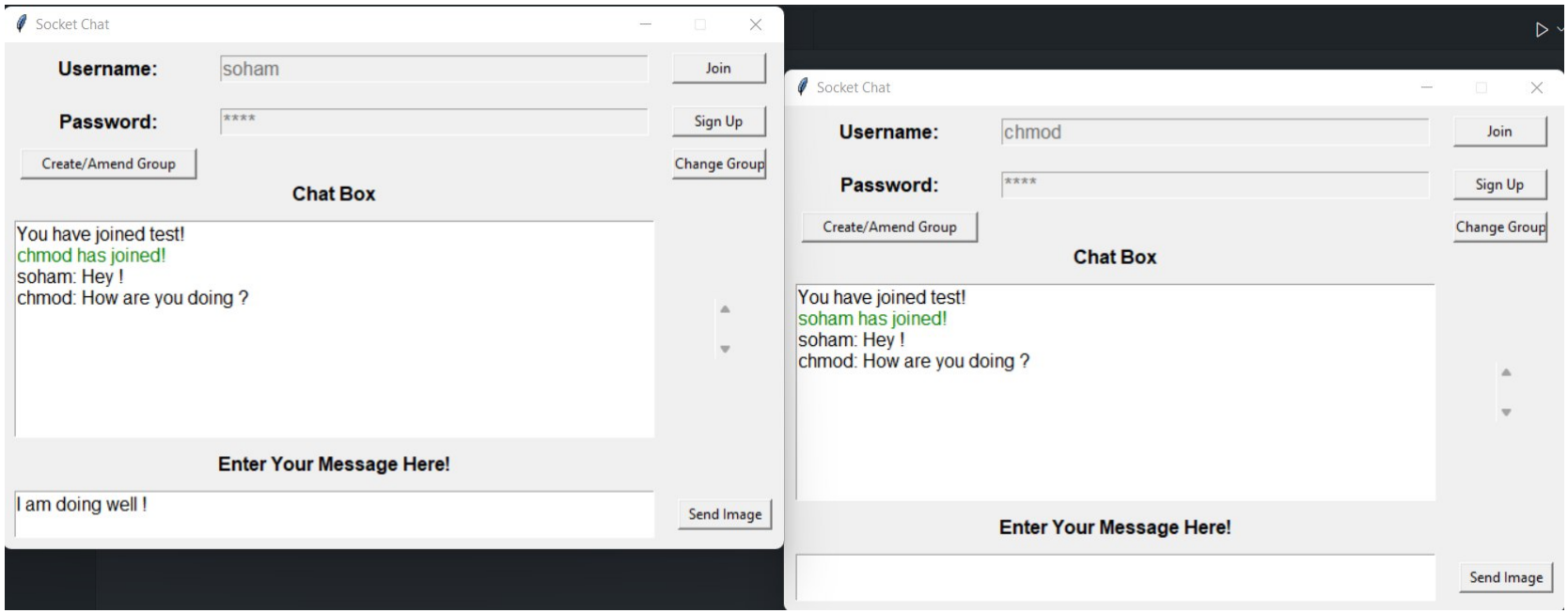


Figure: GUI Overview

Databasing

For the purposes of networking and storing information pertaining to the chat application, we have used PostGreSQL. PostGreSQL is useful especially for dealing with efficient handling of concurrency and having a "delocalised" database in some sense. We have prepared a database schema comprising of four tables.

Schema

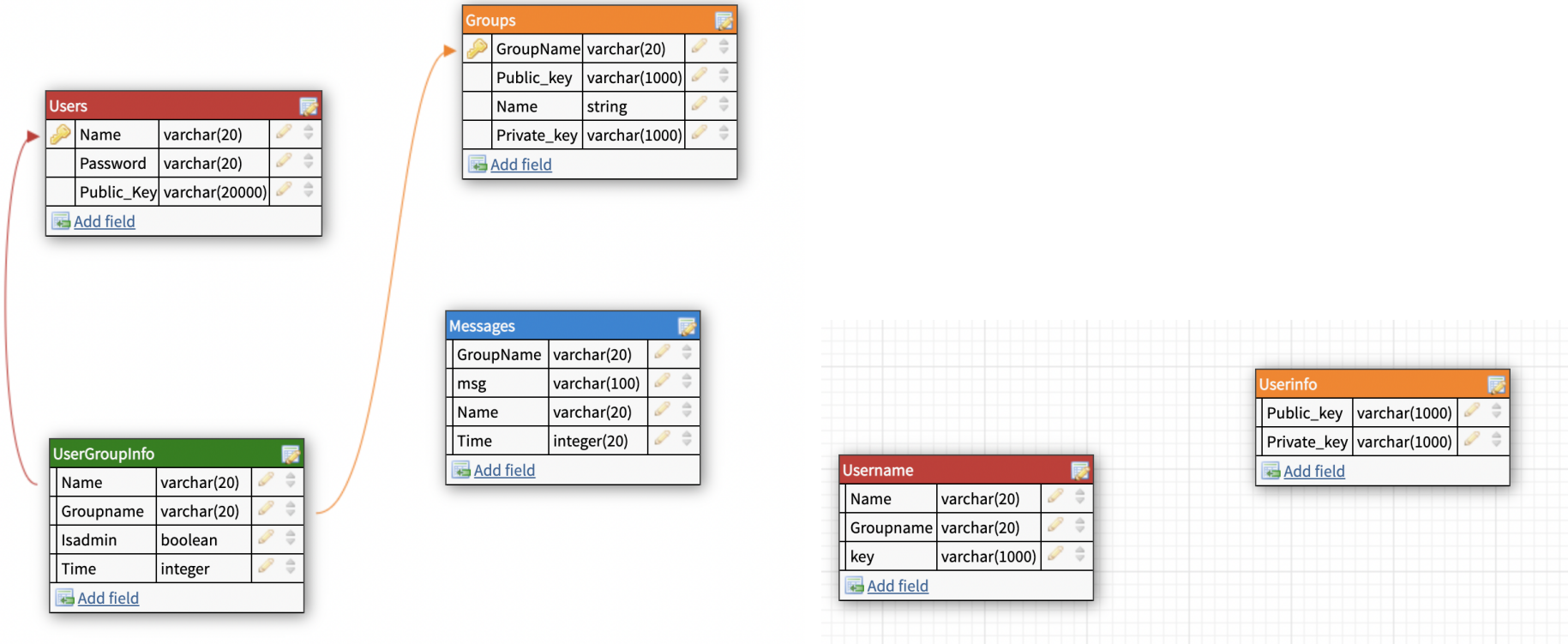


Figure: Database Schema at Server and Client

Login And Signup

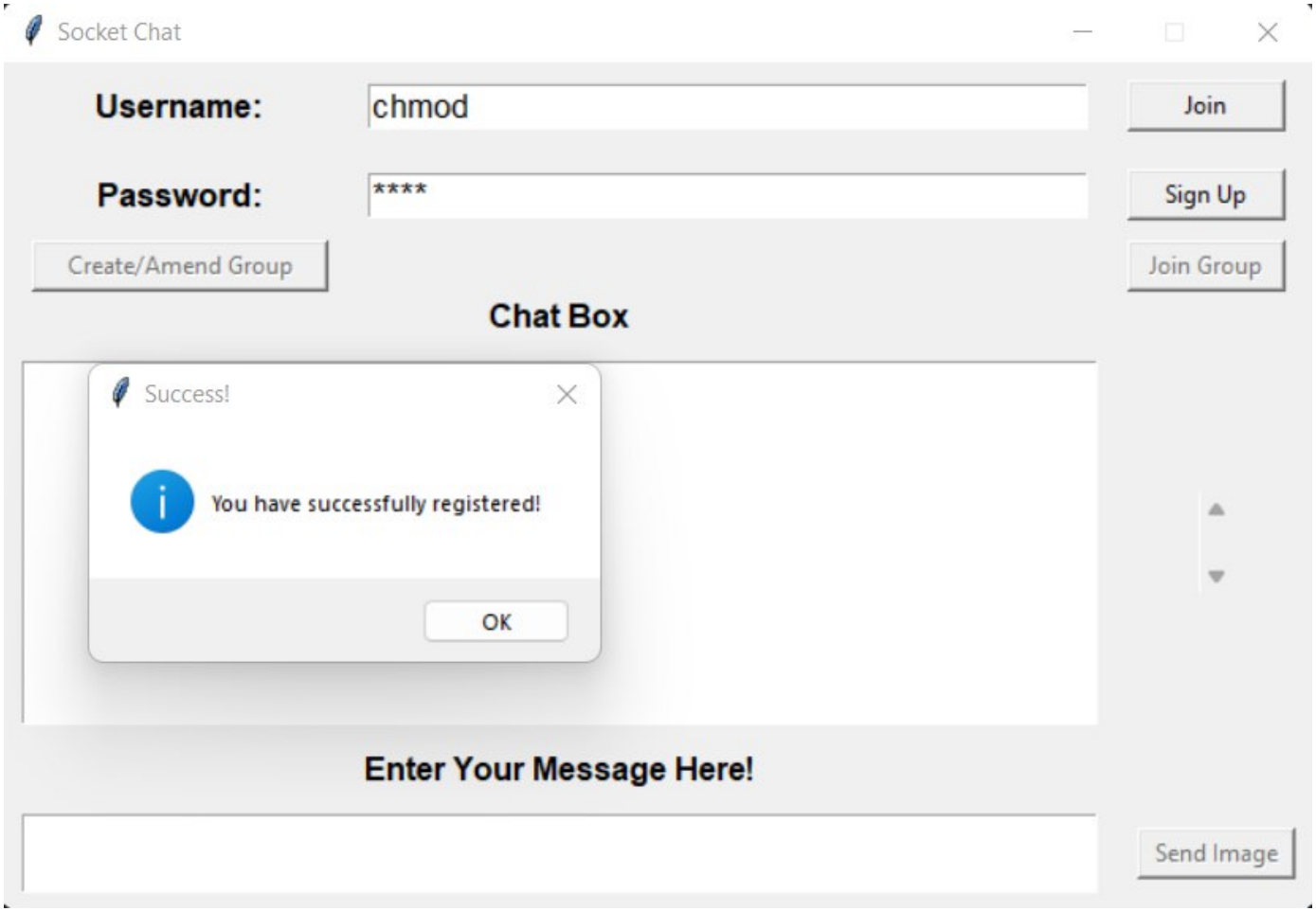


Figure: User registration

Create and Amend group

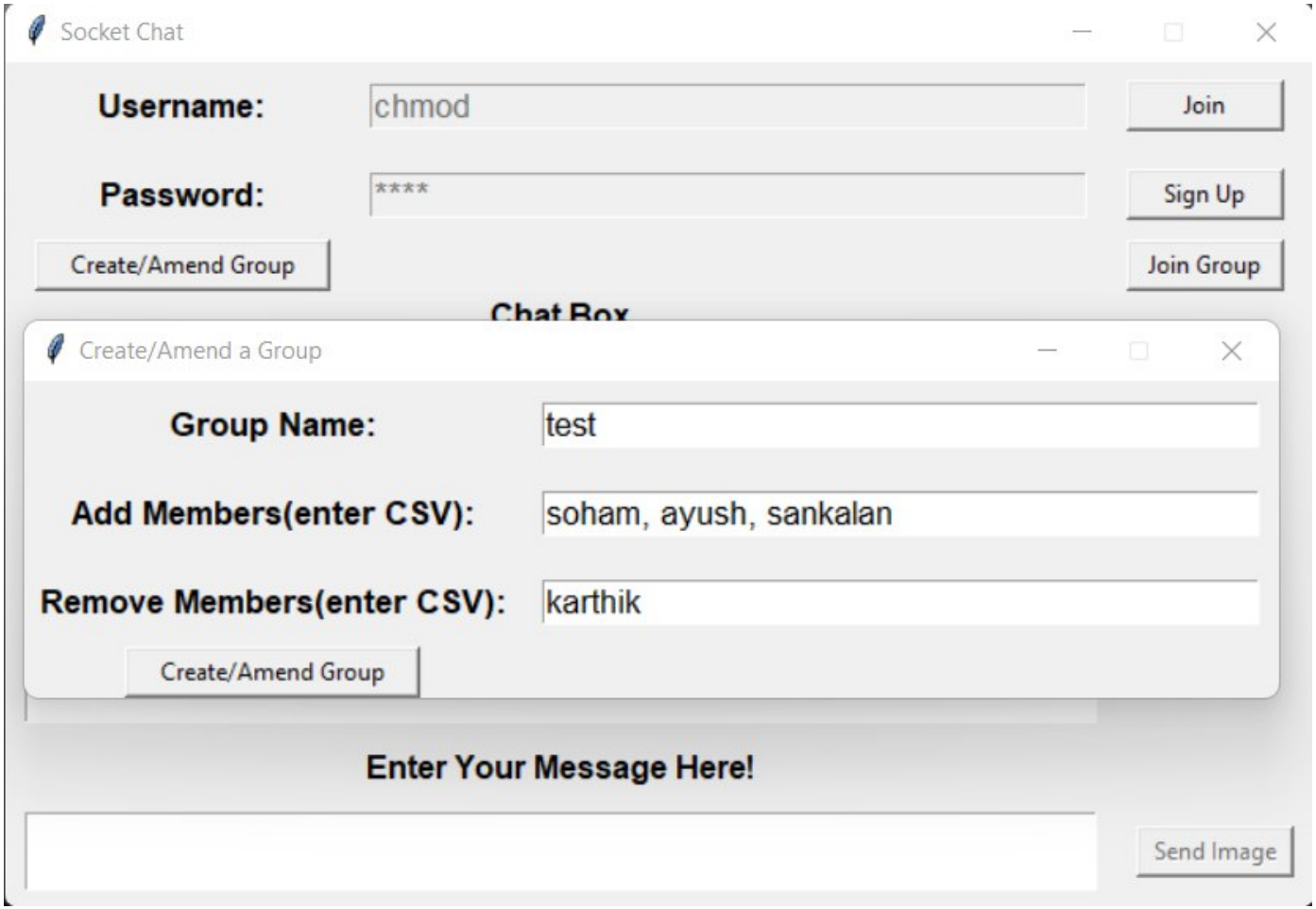


Figure: Group Creation

End-to-end encryption

We are handling end-to-end encryption via a combination of RSA and AES ciphers. Essentially, first every participant generates their own RSA public and private keys. Now, the fernet (AES) key is generated by the group creator and is encrypted using the public keys of the group participants. Now this encrypted message is sent to each of the group participants and they obtain the AES key by decrypting the message using their private key. Now that the keys are established, all messages of a group are symmetrically encrypted and decrypted using the fernet keys. Hence, we achieve semantic security.

Images interface

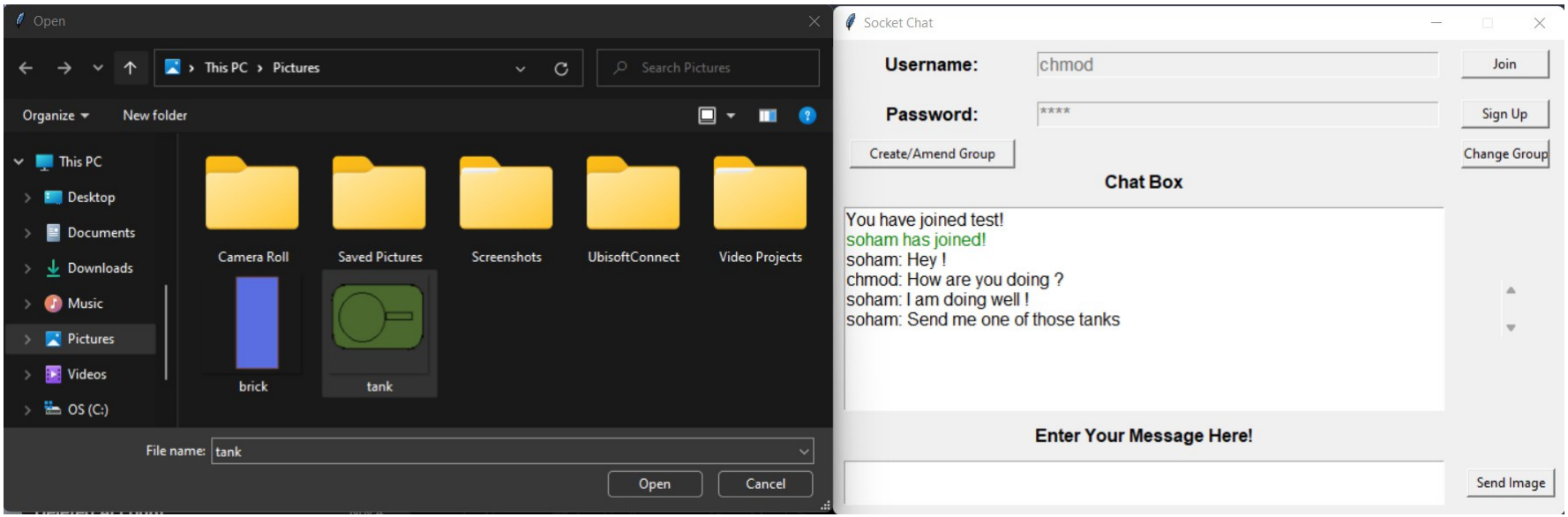


Figure: Images

Mechanism

The mechanism for sending images essentially remains the same as that of messages, with the addition of a few libraries. For the purpose of sending images. we first obtain the file path of the image to be sent using a graphic interface using tkinter. Now that the file path is obtained, we simply convert the images into bytes by reading the file and then reconstruct the image at the destination using the Bytes library.

Load Balance and Strategy

Standard Round Robin

In this strategy, the servers are accessed in periodic order according to their availability. Hence, the servers are accessed in cyclic order by the clients. The Load Balancer essentially acts as a database mediator, updating the id of the next available server

Round Robin Client Side

In this strategy, the servers are accessed in periodic order by each client. Essentially, each client accesses each of the servers in cyclic order. This strategy does not need a load balancer.

Load Balance and Strategy

Random Robin

This is essentially random accessing of servers. Good randomised algorithms essentially ensure that the load among servers is distributed evenly. This strategy does not need a load balancer.

CPU Utilisation

Here, the servers with lesser CPU Usage are preferentially accessed. Here, a load balancer monitors the CPU usage of each of the servers using databasing and it returns the least occupied server when queried for the same by a client

Data Generation

For the purpose of data generation, we have used the following libraries in python3 :

1. os,sys
2. threading
3. multiprocessing
4. pwntools, pwn
5. numpy
6. functools

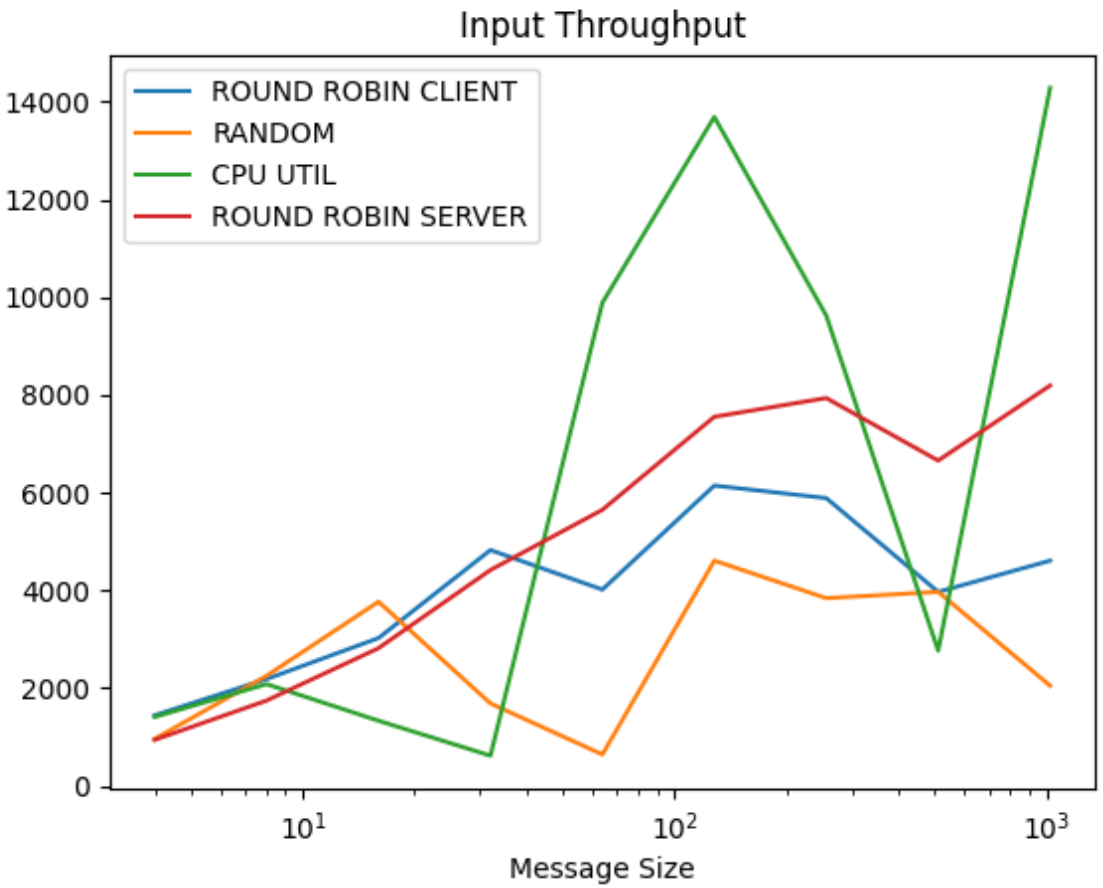
We have creating python scripts in order to automate testing for varying number of servers, clients, message lengths, and different load balancing strategies. In order to replicate our results, one can run the script superanalysis.py from our github repository

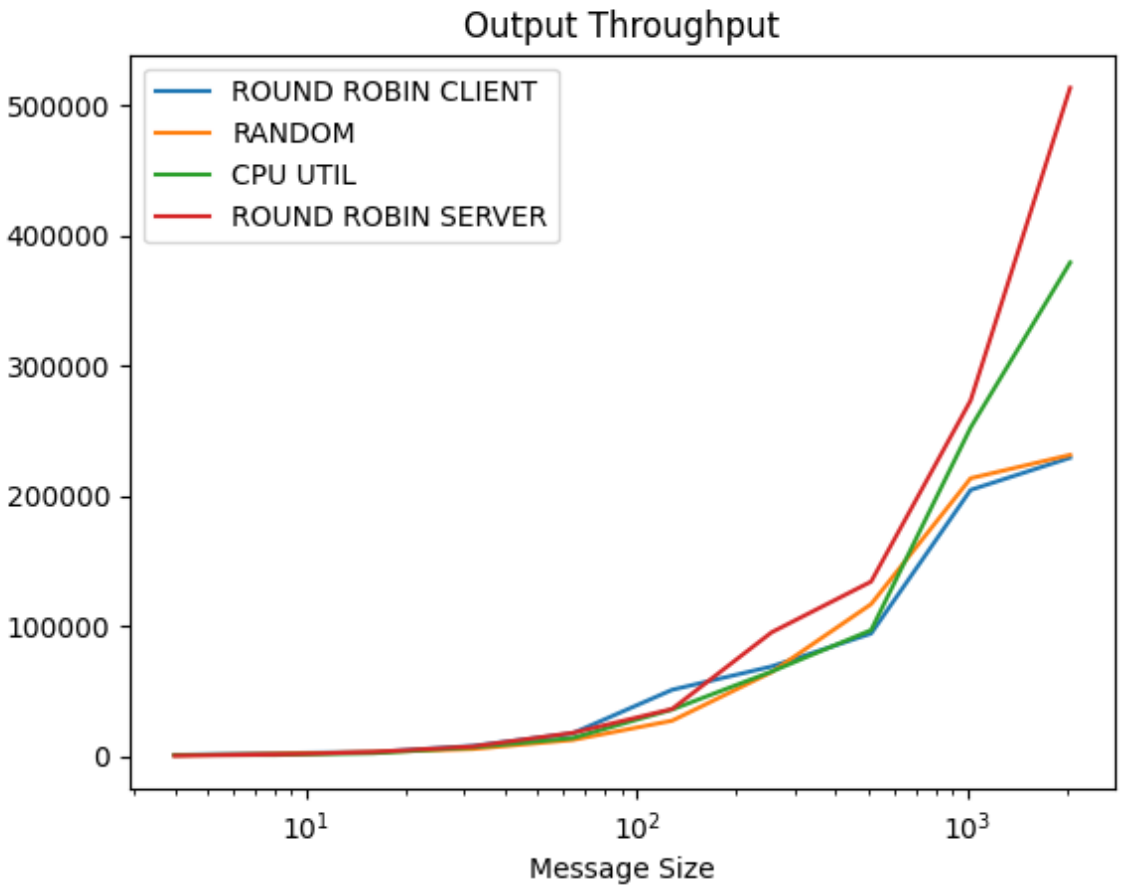
Data Analysis

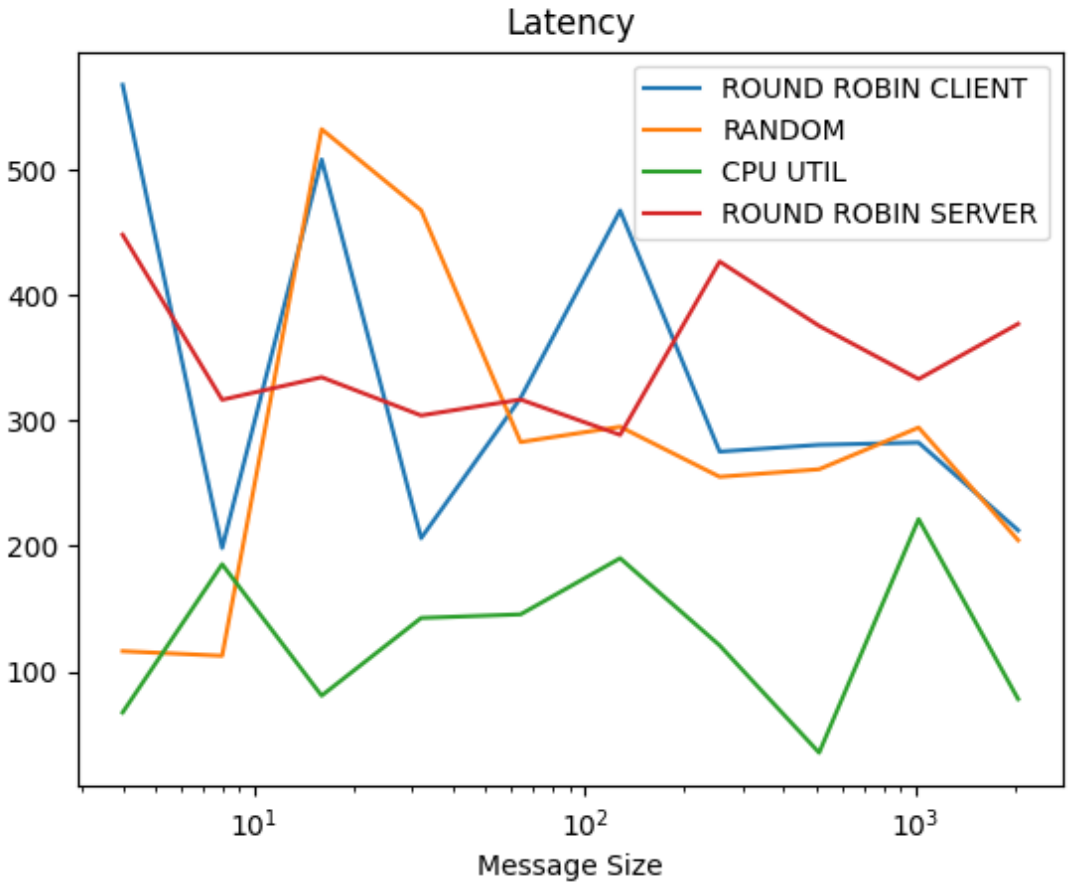
We have analysed the variation of latency and throughputs in the following cases :

1. Varying message sizes for fixed number of clients and servers
2. Varying number of clients
3. Varying number of servers
4. Different Load Balancing Strategies

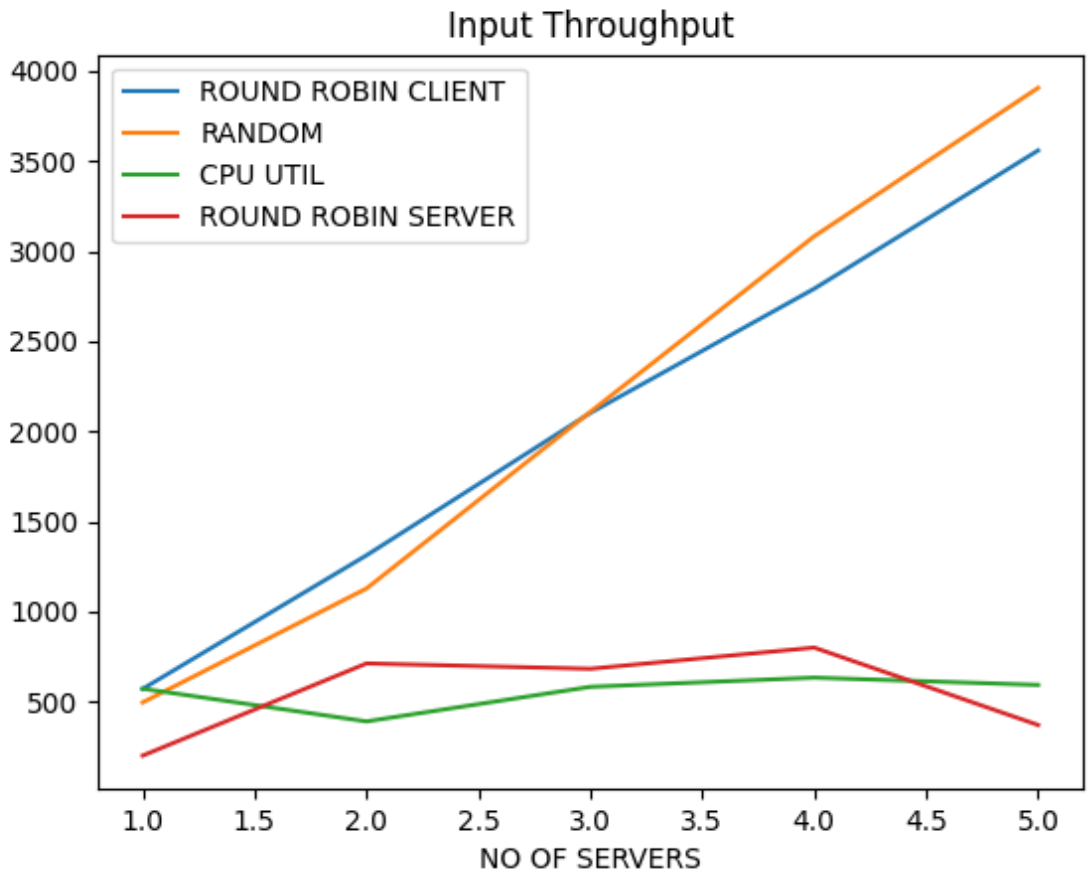
Varying message sizes

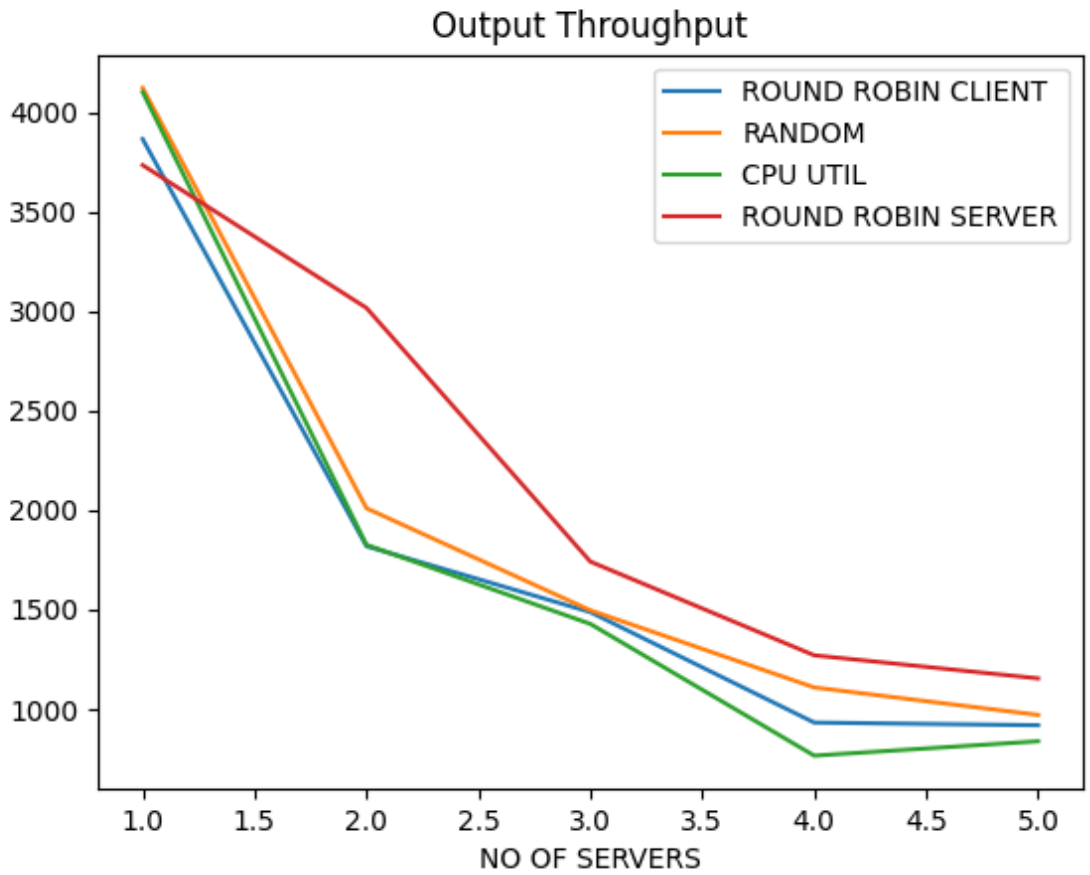


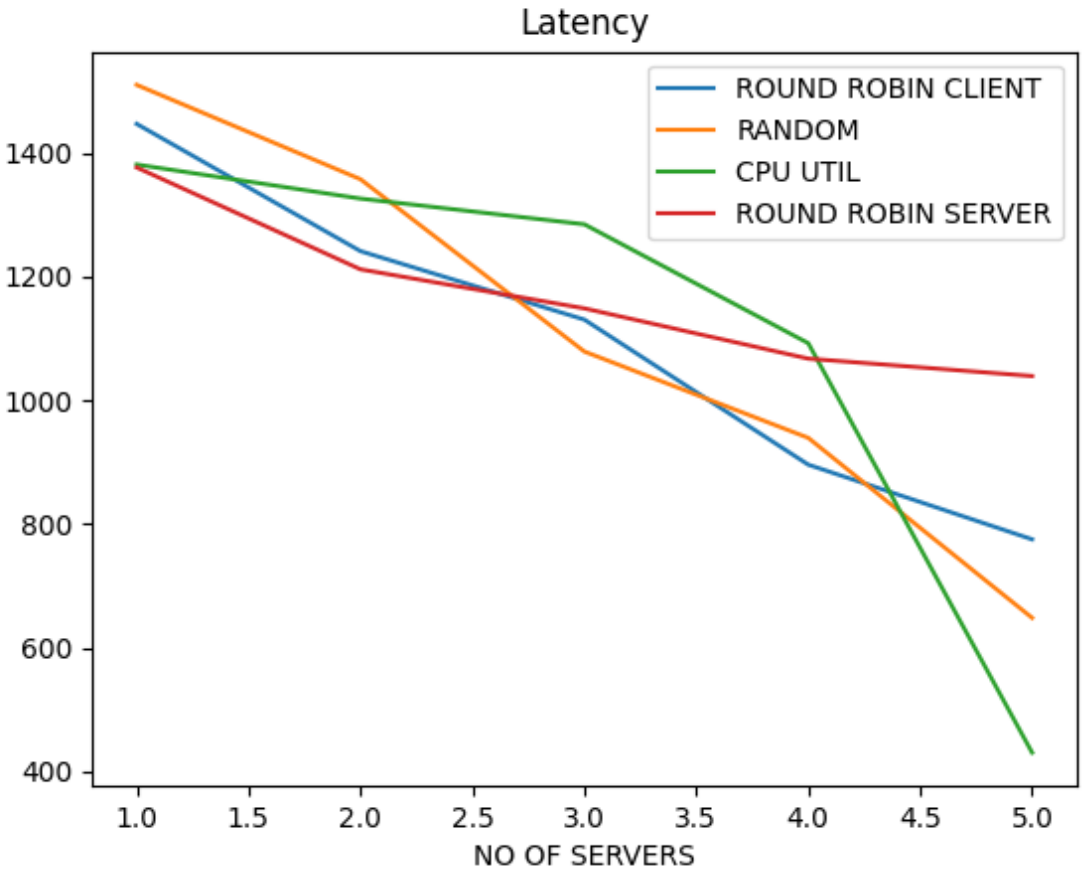

Output Throughput
ROUND ROBIN CLIENT
RANDOM
CPU UTIL
ROUND ROBIN SERVER
500000
400000
300000
200000
100000
0
10^1
10^2
10^3
Message Size


Latency
ROUND ROBIN CLIENT
RANDOM
CPU UTIL
ROUND ROBIN SERVER
500
400
300
200
100
10^1
10^2
10^3
Message Size

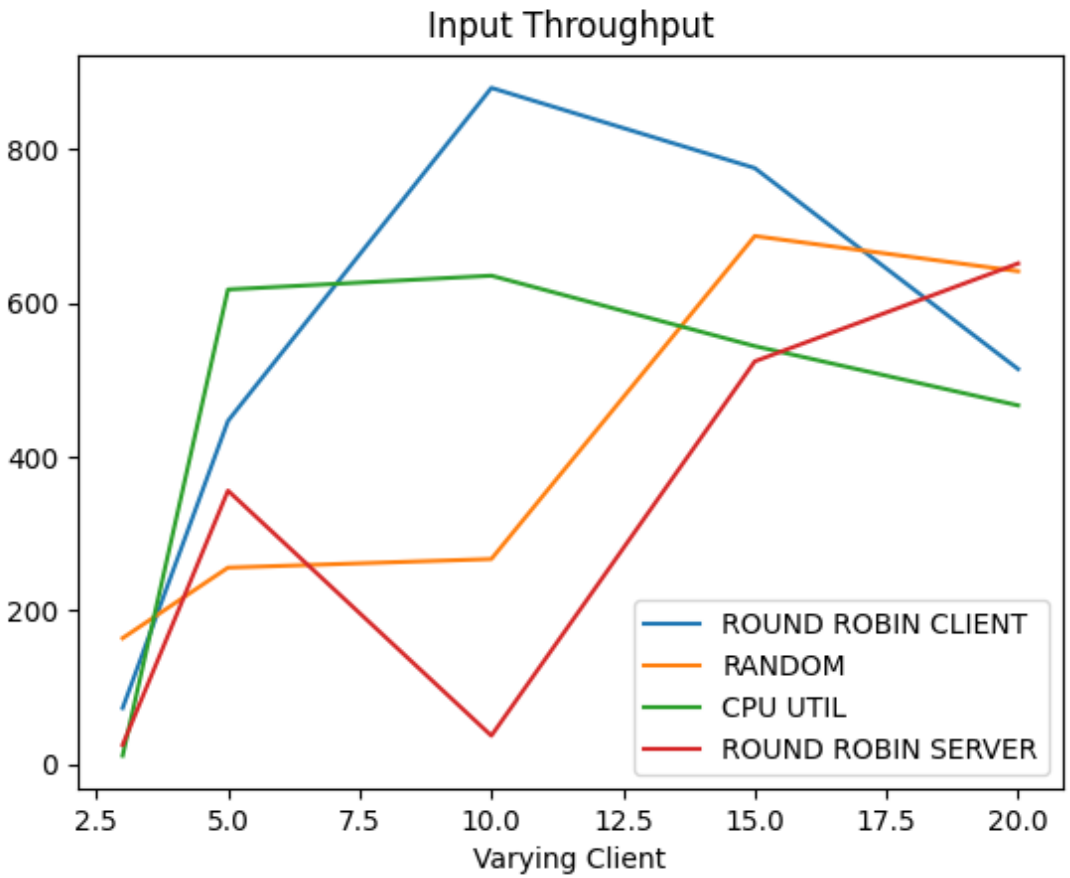
Varying Number of Servers

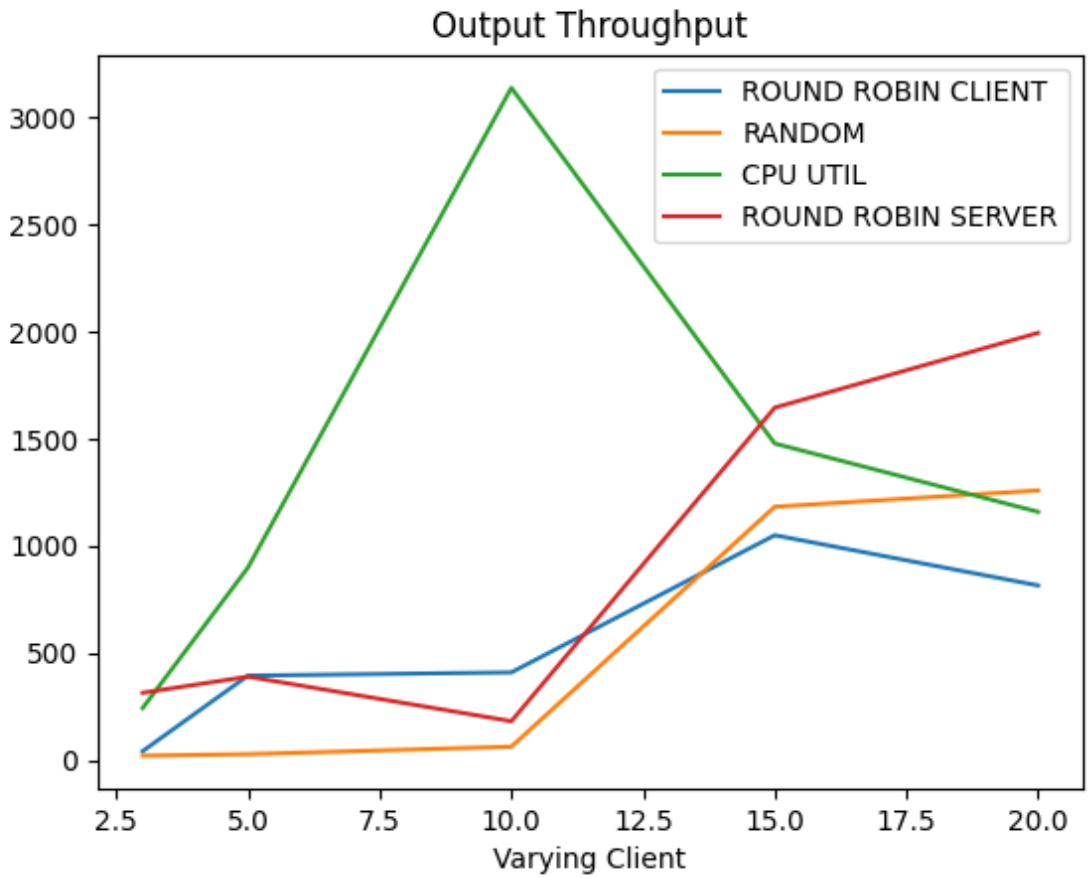


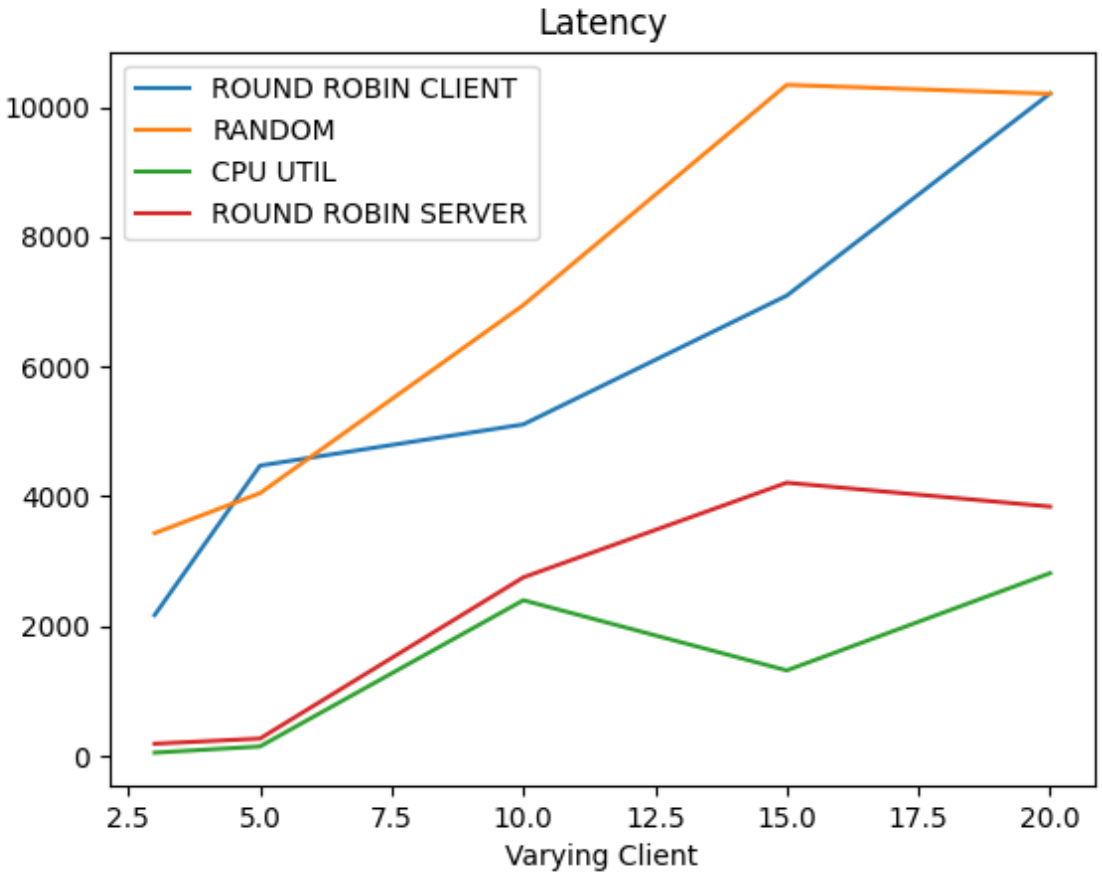

Output Throughput
ROUND ROBIN CLIENT
RANDOM
CPU UTIL
ROUND ROBIN SERVER
4000
3500
3000
2500
2000
1500
1000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
NO OF SERVERS


Latency
ROUND ROBIN CLIENT
RANDOM
CPU UTIL
ROUND ROBIN SERVER
1400
1200
1000
800
600
400
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
NO OF SERVERS

Varying Number of Clients




Output Throughput
ROUND ROBIN CLIENT
RANDOM
CPU UTIL
ROUND ROBIN SERVER
3000
2500
2000
1500
1000
500
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Varying Client


Latency
ROUND ROBIN CLIENT
RANDOM
CPU UTIL
ROUND ROBIN SERVER
10000
8000
6000
4000
2000
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Varying Client

Critiqué

Latency

In our observations, message size seems to have minimal to zero effect on Latency. As the number of servers increases, or the number of clients decreases, the latency decreases. This matches well intuitively, since, the packet size of the message itself is 1024, and hence effects would be observable truly on relatively large messages or images. Moreover, as the number of servers increases or the number of clients decreases, the "queue" at each server reduces, hence reducing latency.

Performance

The Latency varies from a few milliseconds to the few tenths of a second depending upon the aforementioned factors

Critiqué

Output Throughput

We observe that the output throughput increases significantly with the size of the message. Moreover, it decreases with increasing number of servers. The output throughput increases with increasing number of clients as well, but that is caused in our opinion by the delay between sending out the initial flood of messages and the time when the listening starts. This behaviour can be justified intuitively as well. As the number of servers increases, the load balancer takes more time to offer choices to the client, hence, outgoing messages are delayed.

Critiqué

Input Throughput

We observe that the input throughput increases with the size of the message. Moreover, it increases with the number of servers for two of the strategies, and it roughly increases with number of clients because of more number of messages going across clients. Less clients cause the received messages to be sparse. We think that the input throughput increases with message size more bytes are carried in each message.

Cross-strategy comparison

For this, it is clear that low latency, high throughput should be the metric upon which this order is to be decided. By observing graphs, in almost all of them **CPU Utilisation** based load balancing seems to be the best strategy. It is followed by the **Standard Round Robin**, and this is followed almost equally by the **random strategy** and **client-side round robin**.