# Distributed Systems
# COMP90015 2018 SM1
# Project 1 Multi-server Network

**Group Name:** Knight Coders

**Group Members:**
Hyesoo Kim - hyesook (881330)
hyesook@student.unimelb.edu.au
Kuldeep Suhag - ksuhag (919397)
ksuhag@student.unimelb.edu.au
Teenu Thomas Thaliath - tthaliath (959956)
tthaliath@student.unimelb.edu.au
Vadiraj Kulkarni - vkulkarni (950587)
vkulkarni@student.unimelb.edu.au

**Mentor:** Irum Bukhari

## 1. INTRODUCTION

The purpose of this project was building the multi-server broadcasting system and communicating between multiple clients by following the instruction specified. Clients are able to register their username and secret password and it allows them to login and logout as anonymous or using registered a username/secret set. When clients communicate, all messages are sent as JSON objects. There are several features in this project. First of all, the system keeps load-balancing between servers. Also, there are synchronisation strategies to control persistent client registration. Lastly, there are authentication features to protect the system from vicious server. The most challenging part throughout the project was handling load balancing between servers and user registration. We have achieved several things. To be specific, we figured out how to pass JSON object between two systems. Also, we understood how to handle server load balancing.

## 2. SERVER FAILURE MODEL

When a server failure occurs, we can assume a fail-stop module for servers, where a server that crashes or quits never starts up again, or we may assume that servers that crash or quit may eventually restart, at the same address and using the same port number.
In this project, it is assumed that the servers never crash but they may quit gracefully by broadcasting a quit message. For implementing this, we could add a JSON message to the existing commands like "SERVER_QUIT". In this way, the server can exit without disconnecting the servers connected to it.
In Figure 1, we can see one handling mechanism i.e. before quitting Server 1 will give its parent's address/port for its child servers to connect. However, complications arise when the server that is quitting is the root server. In such cases, one solution is to make one of its child servers the root server.

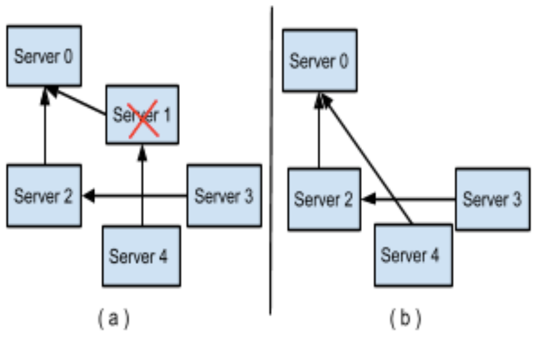

*Figure 1: Handling Server Failure. Server architecture (a) before server quits and (b) after server quits.*

The second case for server failure is when servers crash without warning. The tough part in this scenario is to know if and when a server has crashed. For this, TCP/IP protocol has a timeout period that gives information when a server is crashed. One solution for it is to hold the network topology structure somewhere so that a server can get it easily when it needs to reconnect.
The scenario where a server may restart at the same address and port number doesn't seem to be the best approach. However, when such structure is a requirement then this could be done by continuous attempt to reconnect. The best approach would be to connect to a different server in the network.
In the tree of servers, if one of the servers quits, then the tree would not remain as a single tree it may split in two or more. Therefore, if two clients register with the same username and secret in two different trees, then both clients will get registered. If the server that was failed is re-initiated, then the tree gets the original shape that would mean that there are two users with the same username and secret, then the system cannot differentiate the users.

## 3. CONCURRENCY ISSUES

The protocol does not address all the issues that arise while the clients are connecting to the server. If two clients try to connect simultaneously with the same username and different secrets. Both the clients try to register on different servers. Both the servers send LOCK_REQUEST to every other server after some delay. After some time, both the lock request command will clash which will result in LOCK_DENIED to one of the client. However, until this happens there is a short time when one client tries to register to a server which is only aware of their LOCK_REQUEST and register that user. Thus, one user may be logged in as the other user which is successfully registered with servers.
The load balancer does not redirect the clients that are already connected to the servers that were started before new servers are started. The load balancer kicks in when there is an available server and the

client tries to connect to the one which already has many client connections that can make the server busy at one point of time.
The solution for registering the user issues would be to keep a list of pending clients on each server. When there is lock request the client gets added to this list instead. Also, the server sends lock request command to every server. Now the server which sends the lock request command gets lock allowed command from other servers and it can send a final broadcast message to every server (CONFIRMED_REGISTER). After the servers get this announcement, it adds that client who has successfully registered to their own list.

## 4. SCALABILITY

The network is modelled in a tree like structure with each new server connected to only one other server throughout. With "SERVER_ANNOUNCE", this system broadcast from every server to every other server on a regular basis, once every 5 seconds by default, i.e. all servers connected will sent a message to other servers that they are connected for every 5 seconds interval. Therefore, for a tree like structure, there will be $n - 1$ edges (communication lines) for $n$ nodes (servers) i.e. $n * (n - 1)$ messages will be sent in the network every 5 seconds. Hence, the complexity of the messages is $O(n^2)$.
When a client tries to register with a username and a secret, the server broadcast to all other servers. In return, the servers will reply all the servers with either a "LOCK_ALLOWED" or "LOCK_DENIED" message. Eventually, as the numbers of servers grows, the number of messages in the server network will also grow. To overcome this issue, a possible solution is to make the main server using the dynamic allocation policy. Yongbing Zhang [1] has shown that the static allocation policies like round robin are negligibly inferior to their dynamic counter parts. A downside of having a main server is that, the main server is the bottleneck of the network and can become a single point of failure. However, theoretically the main server can be extended to become an abstract service. As different servers are connected to the main server, the user registration can be handled in different way as stated below, which can help in elimination of some announcements. The more appropriate way in which user registration can be modified is to utilize the tree structure of the network. We propose a modified way of user registration protocol that acts recursively with servers to rely on the lock information received by immediate neighbours instead of getting lock information from every server in the network. Suppose a root server receives a REGISTER command from client, and if that server does not know about the requested username, then it will send a single LOCK_REQUEST to its immediate neighbours or immediate children server

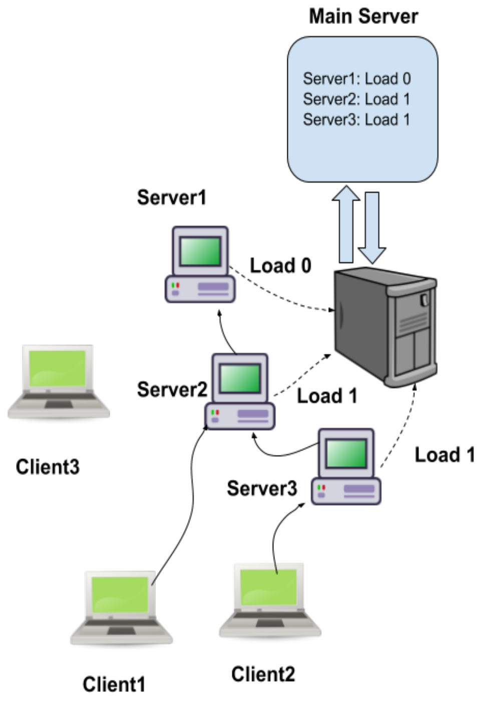

*Figure 2: represents how each server in network updates its load with the master server. If a new Client3 wants to join the network, then it will be redirected to Server1 as it has 0 no of load.*

and will wait to get the LOCK_ALLOWED messages from them. If the servers receiving the LOCK_REQUEST message doesn't know about the username, then it sends the LOCK_REQUEST message to its immediate children. When all the expected LOCK_REQUEST messages are received the server can respond back to its parent server. When the parent server receives all the LOCK_ALLOWED or a single LOCK_DENIED message from its children servers. At this point, a server has to get a permission from its parent server before registering a new client. Therefore, we propose a unique way of registering when a requesting server receives all the LOCK_ALLOWED or LOCK_DENIED commands, it will either announce REGISTER_SUCCESSFUL or REGISTER_FAILED respectively. With this proposed method, a user will only require maximum $O(n)$ LOCK_REQUEST messages, $O(n)$ LOCK_ALLOWED messages and $O(n)$ REGISTER_SUCCESSFUL messages.

## REFERENCES

[1] Y. Zhang, H. Kameda and S. L. Hung, "*Comparison of dynamic and static load-balancing strategies in heterogeneous distributed systems*" in IEE Proceedings -Computers and Digital Techniques, vol. 144, no. 2, pp. 100-106, Mar 1997.

## Project Plan

### AIM
This project is building a multi-server system for broadcasting activity objects between a number of clients.

### Task Breakdown

| Task | Assigned To | Expected Date |
|---|---|---|
| Server Design and Transmission | Teenu Thomas Thaliath, Kuldeep Suhag | 10 April 2018 |
| Client Design | Hyesoo Kim, Vadiraj Kulkarni | 8 April 2018 |
| Register | Vadiraj Kulkarni | 16 April 2018 |
| Login/Logout | Kuldeep Suhag | 16 April 2018 |
| Server Announce | Hyesoo Kim | 16 April 2018 |
| Activity Broadcast | Teenu Thomas Thaliath, Vadiraj Kulkarni | 16 April 2018 |
| Authentication | Kuldeep Suhag | 16 April 2018 |
| Lock Command | Teenu Thomas Thaliath | 16 April 2018 |
| Implementation of command on Client | Teenu Thomas Thaliath, Vadiraj Kulkarni | 25 April 2018 |
| Implementation of command on Server and handling Failure Model | Hyesoo Kim, Kuldeep Suhag | 25 April 2018 |
| Testing | Hyesoo Kim, Kuldeep Suhag, Teenu Thomas Thaliath, Vadiraj Kulkarni | 26 April 2018 |
| Report | Hyesoo Kim, Kuldeep Suhag, Teenu Thomas Thaliath, Vadiraj Kulkarni | 28 April 2018 |