

---

## FACULTY OF SCIENCE AND TECHNOLOGY

COURSEWORK FOR THE BSC (HONS) INFORMATION TECHNOLOGY; BSC (HONS) COMPUTER SCIENCE; YEAR 2

ACADEMIC SESSION 2017; SEMESTER 3

NET3204: Distributed System

Project

DEADLINE: Week 12

---

### INSTRUCTIONS TO CANDIDATES

- This assignment will contribute 30% to your final grade.
- This is a Group assignment of 5 members.

### IMPORTANT

The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorized late submission of work.

- Coursework submitted after the deadline but within 1 week will be accepted for a maximum mark of 40%.
- Work handed in following the extension of 1 week after the original deadline will be regarded as a non-submission and marked zero.

**Lecturer's Remark** (Use additional sheet if required)

**Choong Kai Wern**      **15053648**

I..... (Name) .....std. ID received the assignment and read the comments..... **19/11/17** (Signature/date)

### Academic Honesty Acknowledgement

**Choong Kai Wern**

"I .....(student name). verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties (refer to page 16, 5.5, Appendix 2, page 44 of the student handbook diploma and undergraduate programme) for any kind of copying or collaboration on any assignment."

..... **19/11/17**  
..... (Student's signature / Date)

---

**FACULTY OF SCIENCE AND TECHNOLOGY**

**COURSEWORK FOR THE BSC (HONS) INFORMATION TECHNOLOGY; BSC (HONS) COMPUTER SCIENCE; YEAR 2**

**ACADEMIC SESSION 2017; SEMESTER 3**

**NET3204: Distributed System**

**Project**

**DEADLINE: Week 12**

---

**INSTRUCTIONS TO CANDIDATES**

- This assignment will contribute 30% to your final grade.
- This is a Group assignment of 5 members.

**IMPORTANT**

The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorized late submission of work.

- Coursework submitted after the deadline but within 1 week will be accepted for a maximum mark of 40%.
- Work handed in following the extension of 1 week after the original deadline will be regarded as a non-submission and marked zero.

**Lecturer's Remark** (Use additional sheet if required)

**Lim Shi Hern**

**15075153**

I..... (Name) .....std. ID received the assignment and read the comments.....  
19/11/17 (Signature/date)

**Academic Honesty Acknowledgement**

**Lim Shi Hern**

"I.....(student name). verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties (refer to page 16, 5.5, Appendix 2, page 44 of the student handbook diploma and undergraduate programme) for any kind of copying or collaboration on any assignment."

.....  
19/11/17 (Student's signature / Date)

---

**FACULTY OF SCIENCE AND TECHNOLOGY**

**COURSEWORK FOR THE BSC (HONS) INFORMATION TECHNOLOGY; BSC (HONS) COMPUTER SCIENCE; YEAR 2**

**ACADEMIC SESSION 2017; SEMESTER 3**

**NET3204: Distributed System**

**Project**

**DEADLINE: Week 12**

---

**INSTRUCTIONS TO CANDIDATES**

- This assignment will contribute **30%** to your final grade.
- This is a Group assignment **of 5 members**.

**IMPORTANT**

The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorized late submission of work.


- Coursework submitted after the deadline but within 1 week will be accepted for a maximum mark of 40%.
- Work handed in following the extension of 1 week after the original deadline will be regarded as a non-submission and marked zero.

**Lecturer's Remark** (Use additional sheet if required)

**Mah Qi Hao** **16080111**  
I..... (Name) .....std. ID received the assignment and read the  
comments..... **19/11/17** (Signature/date)

**Academic Honesty Acknowledgement**

**Mah Qi Hao**  
"I .....(student name). verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties (refer to page 16, 5.5, Appendix 2, page 44 of the student handbook diploma and undergraduate programme) for any kind of copying or collaboration on any assignment."

 **19/11/17**  
..... (Student's signature / Date)

---

**FACULTY OF SCIENCE AND TECHNOLOGY**

**COURSEWORK FOR THE BSC (HONS) INFORMATION TECHNOLOGY; BSC (HONS) COMPUTER SCIENCE; YEAR 2**

**ACADEMIC SESSION 2017; SEMESTER 3**

**NET3204: Distributed System**

**Project**

**DEADLINE: Week 12**

---

**INSTRUCTIONS TO CANDIDATES**

- This assignment will contribute **30%** to your final grade.
- This is a Group assignment **of 5 members**.

**IMPORTANT**

The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorized late submission of work.

- Coursework submitted after the deadline but within 1 week will be accepted for a maximum mark of 40%.
- Work handed in following the extension of 1 week after the original deadline will be regarded as a non-submission and marked zero.

**Lecturer's Remark** (Use additional sheet if required)

**Mu Chun Khang**      **13079272**  
I..... (Name) .....std. ID received the assignment and read the  
comments..... **19/11/17** ..... (Signature/date)

**Academic Honesty Acknowledgement**

**Mu Chun Khang**  
"I .....(student name). verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties (refer to page 16, 5.5, Appendix 2, page 44 of the student handbook diploma and undergraduate programme) for any kind of copying or collaboration on any assignment."

..... **19/11/17** ..... (Student's signature / Date)

---

## FACULTY OF SCIENCE AND TECHNOLOGY

COURSEWORK FOR THE BSC (HONS) INFORMATION TECHNOLOGY; BSC (HONS) COMPUTER SCIENCE; YEAR 2

ACADEMIC SESSION 2017; SEMESTER 3

NET3204: Distributed System

Project

DEADLINE: Week 12

---

### INSTRUCTIONS TO CANDIDATES

- This assignment will contribute 30% to your final grade.
- This is a Group assignment of 5 members.

### IMPORTANT

The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorized late submission of work.


- Coursework submitted after the deadline but within 1 week will be accepted for a maximum mark of 40%.
- Work handed in following the extension of 1 week after the original deadline will be regarded as a non-submission and marked zero.

**Lecturer's Remark** (Use additional sheet if required)

**Ong Li Sheng**                      **15071863**  
I..... (Name) .....std. ID received the assignment and read the  
comments..... **19/11/17** ..... (Signature/date)

### Academic Honesty Acknowledgement

**Ong Li Sheng**  
"I .....(student name). verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties (refer to page 16, 5.5, Appendix 2, page 44 of the student handbook diploma and undergraduate programme) for any kind of copying or collaboration on any assignment."

 **19/11/17**  
..... (Student's signature / Date)

# **DISTRIBUTED CHAT SYSTEM**

## **NET 3204** Distributed System

**Choong Kai Wern (100%)**  
15053648

**Lim Shi Hern (100%)**  
15075153

**Mah Qi Hao (100%)**  
16080111

**Mu Chun Khang (100%)**  
13079272

**Ong Li Sheng (100%)**  
15071863

# Table of Contents

<b>Table of Contents</b> .....	7
<b>Introduction</b> .....	8
<b>Physical and Interactive Models</b> .....	8
Physical Model .....	8
Interactive Model .....	9
<b>Scalability and Reliability</b> .....	11
Scalability .....	11
Reliability .....	11
<b>Test Case</b> .....	12
Process Omission Failure .....	12
Communication Omission Failure .....	13
<b>Reflections</b> .....	14
Choong Kai Wern .....	14
Lim Shi Hern .....	16
Mah Qi Hao .....	18
Mu Chun Khang .....	22
Ong Li Sheng .....	24

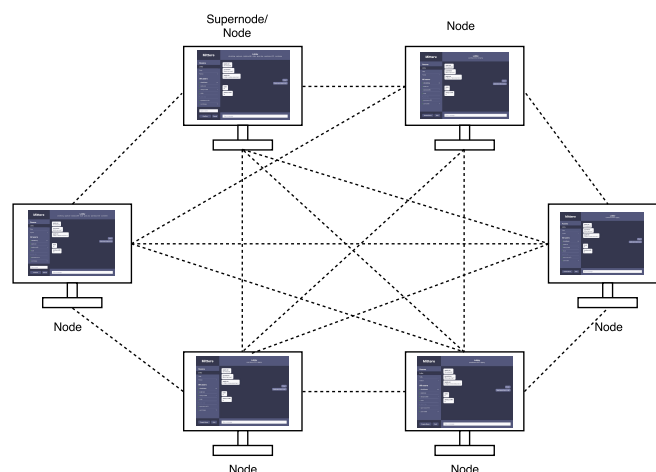
# Introduction

The project involves the building of a distributed chat system called Mittere. The chat system allows users to send messages to each other, either in the form of personal messages or group messages. The novelty of Mittere lies in the utilisation of the peer-to-peer system architecture. Unlike applications that employ the client-server architecture, Mittere operates on the premise of intercommunicating nodes instead of a traditional server to handle all requests from clients. Herein, the chat system is comprised of nodes distributed across the same network that can freely communicate with one another, all managed by an assigned supernode. The supernode acts as the entry point for new nodes trying to join the system, introducing them to all existing nodes. In terms of technologies, the project employs Scala as the high-level programming language, ScalaFX as the user interface (UI) domain-specific language (DSL), and Akka as the concurrency and distributed framework.

## Physical and Interactive Models

### Physical Model

The physical model of Mittere is based on the peer-to-peer system architecture. Specifically, it comprises of a system of nodes where each node can directly communicate with other nodes. Among all the nodes in the system, there will always be one supernode. To start a chat system, a node must first join itself to become a supernode. Then, new nodes must join the supernode to be in the same system. Figure 1 shows the system architecture diagram.

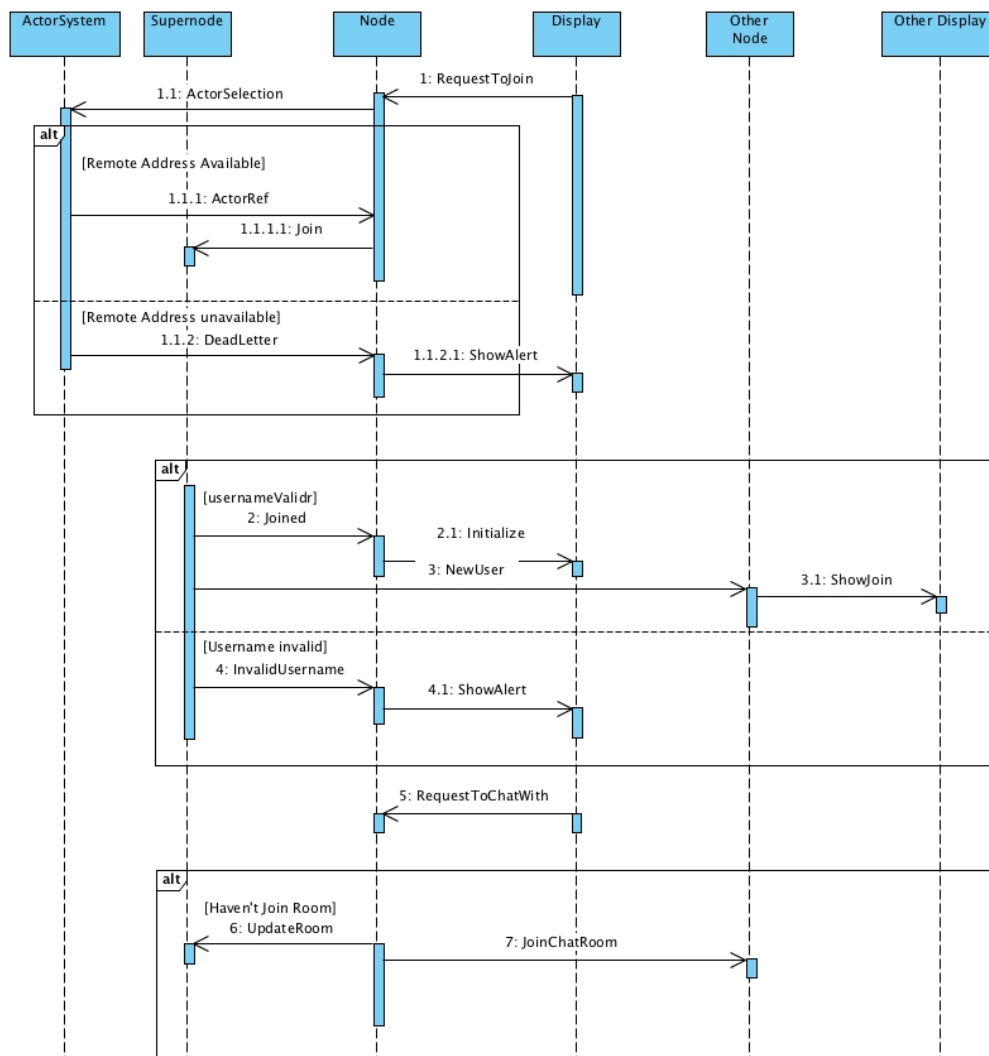


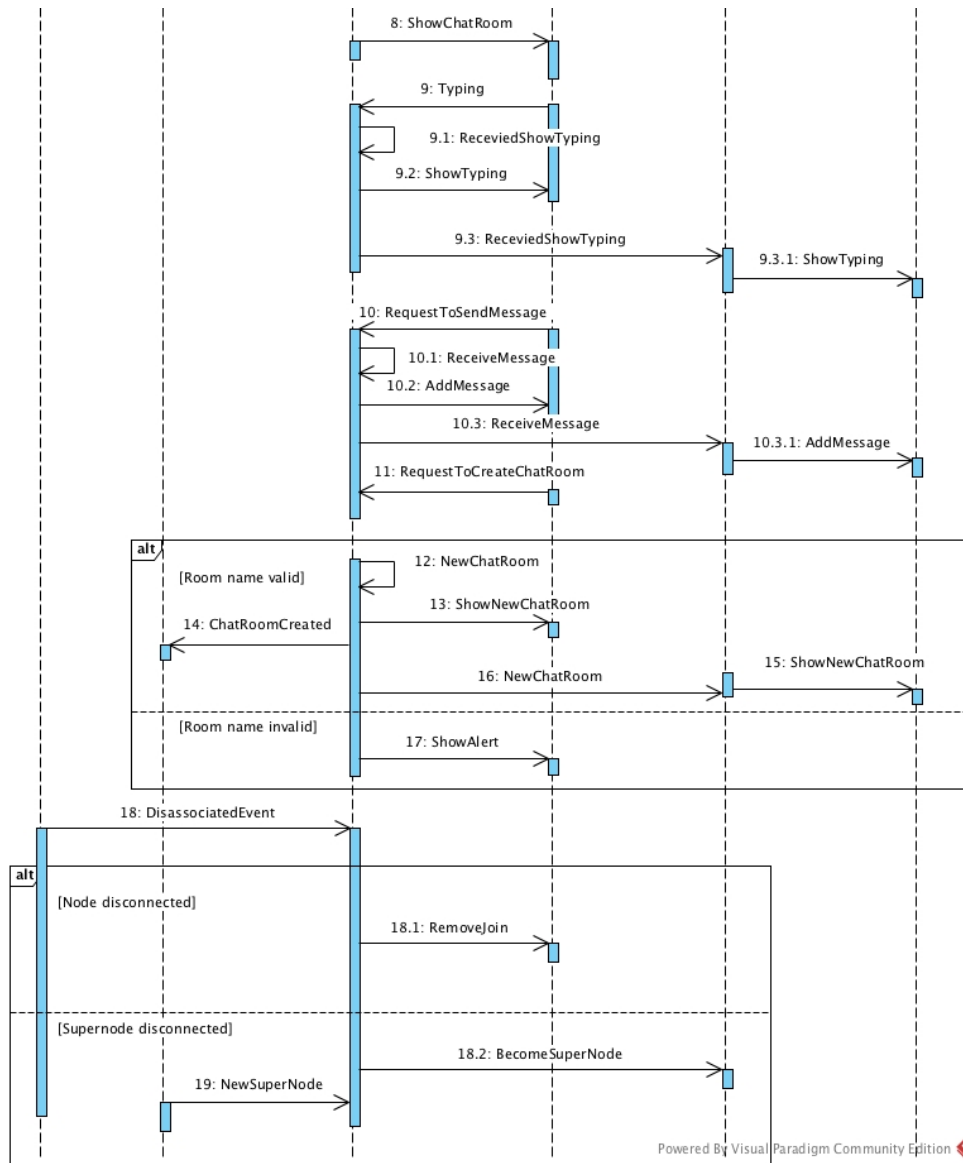
**Figure 1:** System architecture diagram for Mittere



## Interactive Model

Mittere is comprised of four components, namely ActorSystem, Supernode, Node and Display. At this juncture, ActorSystem is responsible for handling the remote networking aspect of the system. It is subscribed to remote events such as DisassociatedEvent and DeadLetters to inform the node about the status of remote nodes. On the other hand, Supernode passes information regarding the chat system environment such as list of users and rooms. Meanwhile, Node handles all the core functionalities of the chat system which include keeping track of users, rooms and messages. Display is responsible for managing the graphical user interface (GUI) of the system. The interactive model of Mittere is based on an asynchronous distributed system. It has no assumed bounds on process execution speed, message transmission delay and clock drift rates. Figure 2 shows the communication protocol sequence diagram of Mittere. It should be noted that the diagram also shows how the nodes check to see whether they should be the new supernode when the current supernode disconnects or crashes.





**Figure 2:** Communication protocol sequence diagram for Mittere

# Scalability and Reliability

## Scalability

Scalability is important in a distributed system, especially a communication system with many users. A system is scalable if it remains effective with significant increases in resources and users. At this juncture, the peer-to-peer system architecture is adopted because it promotes scalability. In Mittere, each node is only concerned with fulfilling its own responsibilities of sending and receiving messages. In other words, a node will never handle a message that it does not intend to send or receive. Hence, the system can be scaled up to accommodate more nodes because no single node is responsible for everything. In Mittere, the nodes each need to store a local copy of all nodes in the system to communicate with one another. Nonetheless, Akka is a high-performing framework with a small memory footprint. Utilising Akka actors permits Mittere to be scaled up to multiple servers easily without significantly affecting the core functionalities or changing the codebase. Furthermore, best practices for scalability were followed strictly in the codebase. For instance, only “tell” is used to send messages as it is non-blocking and delivers the best performance in the context of concurrency and scalability.

## Reliability

On the other hand, reliability is also essential in distributed systems. In Mittere, there will always be a supernode assigned. Herein, the supernode oversees the handling new nodes that just joined the system. Specifically, it keeps track of all the available rooms and nodes in the system. When a new node enters the system through the supernode, its reference is broadcasted to all existing nodes in the system. The new node is also provided with the references to all other nodes in the system. To ensure reliability in this context, the supernode's role is passed on to another node in the system if the current supernode disconnects or crashes. Herein, the new supernode is always chosen based on alphabetical order. It will then broadcast itself to other nodes in the system to inform them. This way, a new node can always join the system even when the original supernode has disconnected or crashed. After joining the system, a node no longer relies on the supernode to communicate with other nodes. Since a node has a copy of all current nodes that will always be updated by the supernode, it can start chatting with other nodes without further assistance. Even if the supernode disconnects or crashes, the node can still communicate with the rest of the nodes.

## Test Case

In terms of Mittere's reliability and fault tolerance, a few test cases were designed to ensure that the system can handle both process and communication omission failures effectively.

### Process Omission Failure

The process omission failure in Mittere can be categorised as fail-stop. In this context, when one node in the system fails, the rest of the nodes subsequently gets notified by the system.

#### Scenario 1: Disconnection of node

Steps taken to simulate scenario:

1. Start several nodes that connect to the same supernode.
2. Quit one of the nodes' application.

Expected results:

1. Other nodes should see the disconnected node removed from the user list.
2. Any node chatting with the disconnected node should see it become offline; no messages can be sent to the disconnected node anymore

The expected results were achieved.

#### Scenario 2: Disconnection of supernode

Steps taken to simulate scenario:

1. Start several nodes that connect to the same supernode.
2. Quit the supernode's application.

Expected results:

1. One of the remaining nodes should become the new supernode.
2. Each node's application should display an alert to inform about the assignment of the new supernode.

The expected results were achieved.

## **Communication Omission Failure**

One possible communication omission failure is when the message fails to be sent out to or received by the intended node.

### Scenario 3: Loss of internet connection

Steps taken to simulate scenario:

1. Start several nodes that connect to the same supernode.
2. Use one of the nodes to start a chat with another node.
3. Send a message out.
4. Immediately disconnect from the internet.

Expected result:

1. The sender node's application should display an alert to inform about the message failing to send out.

The expected result was achieved.

## Reflections

### Choong Kai Wern

Through the assignment I have learned a few key concepts of distributed system. The first one is related to the importance of the architectural model of a distributed system to produce a more reliable system. The architectural model of the system describes the high level representation of how each component communicate with each other. It is important to select the right architectural model for our system. By implementing a peer-to-peer architecture, we can ensure that our system does not have a single point of failure.

Another concept I have learned is related to the communication paradigm of a distributed system. I have learned about message passing. Message passing is when one process pass messages to another process. It is widely used in interprocess communication. However, in our system, through the use of Akka framework, we use the same concept to pass message to remote actors asynchronously to produce a reactive system.

One of the main problem we faced in this assignment is unstructured and redundant code base. The original implementation is to implement all the functionality into one Node actor, which involves in several tasks such as handling join, creating chat room and chatting with other user. This approach is not maintainable in long term.

One of the possible approach is to use State. However, since our architecture is peer to peer, this approach is less suitable. In the chat system, there is only two possible state, which is Initial and Joined. However just splitting into these 2 state still result in a huge chunk of code. At the same time, we have to handle disassociated event in both state, which might results in redundant code.

Hence we use another approach, inspired from one of the Akka tutorial. In this approach, we utilize Scala trait to have a separation of concern. Our Node actor consists of three abstract methods joinManagement, sessionManagement and chatManagement which handle different category of messages separately. Those methods are implemented in a separate trait.

The strength of the distributed system is as followed:

- Heterogeneity
  - Works in different operating systems, because of JVM
- Fault Tolerance
  - Automatically reassign supernode if supernode fail or disconnect
- Reliability
  - Each node has data backup

The weakness of the distributed system is as followed:

- Data does not persist
  - All the data in the chat system is stored in memory, which will be gone after the application is closed
- Security
  - No end-to-end encryption is involved to secure the messages
  - No password is required to join

<b>Group Member</b>	<b>Contribution (%)</b>
Choong Kai Wern	100
Lim Shi Hern	100
Mah Qi Hao	100
Mu Chun Khang	100
Ong Li Sheng	100

## Lim Shi Hern

Distributed system concepts that I've understand and applied in this assignment is that the communication paradigms used here is remote message passing and the communicating entities are the nodes and supernodes. The architectural style used in our system is the peer-to-peer style. Which means that all processes involved in a task or activity play a similar role and they interact cooperatively as peers without any distinction between client and server processes or the computer on which they run.

Another concept understood and implemented in the system is the computer clocks and timing events. To elaborate, each computer in a distributed system has its own internal clock that can be read by local processes to obtain the value of the current time. Two computer can associate timestamps with their event based on their own internal clock. However, even the two process in each computer read their clocks at same time, their local clocks may supply different time values. Also, our system is an asynchronous distributed system. Which means that there are no assumptions about time.

During the original implementation, we focused on just getting node to chat with another node. In our implementation, we use the term user, which consists of username and ActorRef to identify each node. Basically, we just keep track of a key value pairs of "username" and "ActorRef", in order for us to look up the address so we can send the message to the remote `Node`. Each messages with other users is also tracked separately in a key value pairs of "username" and "messages", which is an "ArrayBuffer[Message]". Later on, we add in "Room" to allows for user to have group chat with other users. We tracked a separate key value pair of "roomName" and "Room" object. The "Room" class contains of "name", "Set[ActorRef]" and "ArrayBuffer[Message]". We have both "User" and "Room" class, with the trait "Chattable" at this stage. This makes our UI and "akka" implementation more complicated as the implementation is slightly different for both scenario.

If a user sends a message to a User, we look up the user's ActorRef and send to it. If a user sends a message to a Room, we look up the room object and loop through the users and send the message to each of the user. This implementation is redundant in some sense and increase the complexity of the code at both front end and back end part. Hence, we uniform the interface by just using Room alone. Since eventually, a one-to-one messaging is just a communication



within a Room with just 2 users. We implement the changes by creating Room for each online user in a Node. The Room are then identified by the name of both user, joined with ":". To ensure that the identifier is consistent, the first part of the identifier (before ":") will always be the smaller String. With this approach, we are able to make the code more straightforward and simple, hence improve the maintainability of the chat system.

Strength: Fault masking

Weakness Data does not persist

<b>Group Member</b>	<b>Contribution (%)</b>
Choong Kai Wern	100
Lim Shi Hern	100
Mah Qi Hao	100
Mu Chun Khang	100
Ong Li Sheng	100

## **Mah Qi Hao**

As we always defined, distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages. The Internet enables users throughout the world to access its services wherever they may be located. We have built a distributed chat system with such definition. The entire system is implemented through messages passing and without the uses of shared memory. As this system is distributed, each node will keep their own sets of data rather than accessing a centralized set of data.

Moreover, we have used peer-to-peer architecture where there is no server acting as a middleman. In this architecture, all of the processes involved in a task or activity play similar roles, interacting cooperatively as peers without any distinction between client and server processes or the computers on which they run. In practical terms, all participating processes run the same program and offer the same set of interfaces to each other. In our system, a peer will only need to join to the supernode to get the list of peers. After the list of peers is received, the peer can talk to all of the peers directly without the supernode.

Furthermore, our system is also fault tolerance because the system will be still working if the supernode crashed or disconnected. We are managed to solve this problem through the reassigned of supernode to another node and all the nodes will be updated. We have also used the disassociate event to manage the crashes of other nodes. If a node disconnected from the supernode, the supernode will process the disassociate event and remove name of the node from the system to avoid any new messages sent to the user or node.

On the other hand, heterogeneity is one of challenges of all distributed systems. As we know, heterogeneity means variety and difference of networks, computer hardware, operating systems, programming languages, implementations by different developers. In this system, we have used a middleware to map the heterogeneity of the underlying networks, hardware, operating systems and programming languages. The task of middleware is to provide a higher-level programming abstraction for the development of distributed systems and, through layering, to abstract over heterogeneity in the underlying infrastructure to promote interoperability and portability. Therefore, we have used Akka as our middleware of the system where every process is an individual actor. Each actor will have to communicate with each other through messages passing.

Although all of the necessary functions are implemented in our system, we are still obsessed in providing a system that ensure the quality of service. In this case, quality of service is measured through the performance, reliability, user interfaces and user experience of the system. We hoped that a system with good user interfaces and user experience can be developed besides other non-functional requirements. Therefore, we designed the interfaces nicely with responsive screen, intuitive user interactions and organized layout of elements.

One of the major challenges in creating the Graphical User Interface (GUI) is that the limited resources available online for scalafx. It is because the limited documentation for scalafx and limited number of blogs, forums and extra documentations. Hence, we are trying to search for javafx instead of scalafx because since scalafx is just wrapper for javafx and we are able to write scalafx according to javafx resources.

We also meet a problem in customizing the list cell of the list view of the chatroom. Since it is a chatroom, we hope to create an interface that is familiar to the user like all other chatrooms. When we are trying to achieve this, we ran into a problem which is the limited customization of default list cell. Therefore, we implemented our own custom list cell with custom font size, font family, padding, background colour, alignment, background radius, minimum width, and maximum width.

After we customized the list cell for chatroom, we found that changing the cell alignment to left and right is still insufficient for group chat because the sender is unable to be identified only through changing the cell alignment to left and right. Thus, we created another type of list cell that can show the sender of the message and use it in group chat.

Rather than sending single line messages, we are trying to send both single line and multi-line of messages. Meanwhile, we are trying to keep the "enter to send behaviour" for our text area. Therefore, we had customized the text area where it will add a newline while the key combination of shift and enter is pressed, and send the messages when enter is pressed.

Also, we also encountered a problem when we are trying to make the entire sy to be responsive because the components will be misaligned and shrink into inappropriate size. Therefore, we are making great efforts in modifying the layout settings and properties of every component to ensure responsiveness of the interfaces.

We think that user interface and user experiences are playing important roles in determining the successfulness of the entire system. As first, we have already designed the interfaces with the most basic layout and components. However, we decided to redesign and implement the entire system with a better design. We have even design the interface with the properties and layout of each components before the actual implementation of the interfaces. As we decided to majorly change the interfaces, we faced another challenge because every controller of the interface needed to be rewritten.

When we are nearly done with our system, we found one minor issue where the user cannot know when other people message him or her. Therefore, we implemented an unread function where the user can see the message that he or she received but have not read yet. Initially when implementing this function, we done it through refreshing the cells every time a user received a message and display it on the particular cell. However, we found it is very inefficient to refreshing the cells every time and practically a bad practice. Therefore, we decided add listener to every cell and they will always listen to the changes of the items they are displaying.

### Strengths

- Heterogeneity
  - Works in different operating systems, because of JVM
- Fault Tolerance
  - Fault masking
- Reliability
  - Each node has data backup

### Weaknesses

- Data does not persist
- Security

<b>Group Member</b>	<b>Contribution (%)</b>
Choong Kai Wern	100
Lim Shi Hern	100
Mah Qi Hao	100
Mu Chun Khang	100
Ong Li Sheng	100

## **Mu Chun Khang**

Distributed system is defined as hardware or software components located at networked computers that communicate and coordinate their actions by passing messages. Herein, the system's core functionalities are provided entirely through message passing. One of the most important distributed system concepts applied in the project is independent failures. Basically, each component of the system can fail independently without affecting the whole system. Besides that, failure handling is also another concept applied. Specifically, the system needs to be able to both detect and recover from failures.

In terms of how the distributed concepts are applied in the project, Mittere employs Akka, a framework that relies on the concept of actors and message passing to build a concurrent, distributed system. Meanwhile, independent failures are ensured since Mittere is composed of a system of independent nodes. Each node can disconnect or crash independently without affecting the other nodes in the chat system. On the other hand, failure handling is implemented with the notion of disassociation and supernodes. Herein, the Akka actor system will send the `DisassociatedEvent` message to all remaining nodes when a node fails. Nonetheless, the failed node can still rejoin the system by joining the supernode.

The implementation of Mittere was faced with several problems. Namely, I faced a huge problem when dealing with fault tolerance. During the initial implementation, whenever the host node disconnected, the remaining nodes could still communicate with each other because they still retain a local copy of node references each. However, no new nodes can join the system since the entry point is lost. To rectify this problem, I proposed the concept of supernode whereby one of the nodes in the system will always be the entry point. The supernode is responsible in getting new nodes up to date with the chat environment and informing existing nodes about the arrival of new nodes. Herein, when the supernode fails, all remaining nodes will be notified. Subsequently, they will reassign a new supernode among themselves by selecting the node with the smallest name alphabetically to ensure consistency. With that, the nodes will send a message to the new supernode to acknowledge it. Meanwhile, the new supernode will also send messages to the other nodes to inform them of its role.

Of course, Mittere has its strengths and weaknesses. One of the strengths of the chat system is heterogeneity. Mittere works equally on different operating systems due to the underlying Java Virtual Machine (JVM). Besides that, Mittere is also decentralised. This is because it employs a peer-to-peer system architecture whereby there is no single point of failure. On the other hand, one weakness of Mittere is the lack of data persistency. Since the messages received by each node are not stored in local databases or logs, chat history is effectively lost when the node disconnects or crashes before joining the same system again. Also, Mittere offers poor security. In this context, the chat messages are not encrypted in any way, and therefore can be easily intercepted and read by any malicious party.

<b>Group Member</b>	<b>Contribution (%)</b>
Choong Kai Wern	100
Lim Shi Hern	100
Mah Qi Hao	100
Mu Chun Khang	100
Ong Li Sheng	100

## **Ong Li Sheng**

I have learned a lot throughout this assignment. One of the few concepts that I learnt is heterogeneity. Heterogeneity states that the components in a distributed system should be interoperate across different operating systems, hardware architecture, programming languages and more. Our system should be able to work in different operating system.

I have learned that the architectural model is the structure of a system that specifies its components and their relationship between the components. Architecture model is important for making the system reliable, manageable, adaptable and cost-effective. Peer-to-peer architecture is used in this chat system. Peer-to-peer architecture states that all of the peer have similar responsibility and there is no client separation of responsibility such as in client and server architecture. In practice, all the participating peers should behave the same way and have same set of interfaces.

Failure model helps us to understand the effects of failure by describing the ways in how failure may happen. In the real world, the distributed system must be ready for any expected failure. The main reason of a process omission failure of a process is to crash. When we said the process crashed, meaning that the process has stopped working and will not execute any further steps. A process crash is called fail-stop if other processes can detect that the current process has crashed.

Since we are building a Scala application, Scala application runs on a Java Virtual Machine(JVM), which makes our application able to work on different operating systems. Windows, macOS and Linux can run the chat program as long they have a JVM in their environment.

Peer-to-peer is used in this assignment as all the peers are only responsible for sending their message out and receiving messages that is directed towards them. The peers do not need to carry out the responsibility of other peers that are not related to them. In our peer-to-peer architecture, there is a supernode who is also a node, but has extra responsibility of handling new incoming nodes that are looking to join the conversation.



In our assignment, when the current supernode process crashes, the actor system of the application will be informed, and it will send out a notification to all the nodes connected. Then a new supernode will be assigned automatically alphabetically. So once the original supernode crashes, all the remaining node can continues chatting without the need of a supernode. When a new node wants to join, they can join using the new supernode address. The process of can be said as fail stop as the actor system is able to detect on the supernode crashes.

Initially in our implementation, the supernode will keep track of all the currently connected nodes and Room. A separate RoomActor is created in the supernode whenever a node requests to chat with another node. The RoomActor is responsible to keep track of the chat room information such as messages and the users' **ActorRef**. Every time it receives a message from the supernode, it will broadcast to all the node in the room. Hence, by doing this the node have to go through the supernode, where the supernode then tell the RoomActor to broadcast the message. We realized this approach is not reliable and contains a lot of communication overhead. The main problem with this approach is that when the supernode dies or crashes, because all the node go through the supernode to communicate with each other, the Nodes cannot communicate with each other anymore. This would mean that the whole chat system would fail as a whole as none of the nodes are able to send message to each other.

Hence to solve this problem, we have change our approach towards this implementation, the supernode and all the nodes will be responsible to store the list of ActorRef. In that case, the node does not need to communicate through the supernode and the supernode's burden is lessen. The supernode is only responsible for relaying the current nodes and incoming new nodes. The node now can send message directly to other node since they also hold a local copy and do not need send messages through the supernode. RoomActor is no longer needed to track the node to chat with another node. In conclusion, this implementation does not need to rely on the supernode to pass the message. So even if the supernode fails, the core functionality can still be carried on. The overhead of the system can also be reduced since they can communicate directly with each other.

### Strength

- Heterogeneity
  - Our program works in different operating system because we are using JVM.
- Fault Tolerance
  - Reassignment of supernode
- Reliability
  - Each node has their own local copy
  - Each node can become a supernode

### Weakness

- Data does not persist
  - There is no database hence the once the user quit, it cannot be traced back
- Security
  - There is no password required to join
  - No end-to-end encryption

<b>Group Member</b>	<b>Contribution (%)</b>
Choong Kai Wern	100
Lim Shi Hern	100
Mah Qi Hao	100
Mu Chun Khang	100
Ong Li Sheng	100