# HW4 - Groupy

Reethika Ambatipudi

5th October, 2022

## 1 Introduction

We need to implement a multicast group communication application. Using multicast, a source can send a single copy of data to a single multicast address, which is then distributed to an entire group of recipients.

## 2 Main Problems and Solutions

### 2.1 gms1 - Basic Application without election

In gms1, we initiated a node, made that as leader. Later, we initiate other nodes, which request leader node to join.When ever a new node is added, leader also updates the new view and broadcasts it to all the other nodes. Any requirement of change in the state should be informed to the leader and then leader broadcasts the change in state and all the nodes will be in sync.

Problems in this is, if the leader node crashes, there is no solution.

### 2.2 gms2 - Introduction of Monitor and election

In order to solve the problem we faced with gms1, now we introduce a monitor which checks the activity of the leader. If the activity of the leader goes down, election is done and the node which is created next to the old leader will become the new leader. In order to test this, we crash one node after a random time.

Though this solves the problem we faced in gms1, the problem here is if the leader node could successfully pass the last message to only some of the slaves, the slaves which received the last change of state message will be in sync but not the others. Now, this is a serious problem since there is no sync.

### 2.3 gms3 - Introduction of sequence number

To resolve the problem we faced with gms2, now the new leader has to resend the old message it received by the old leader. So, we keep a copy of the last message of the old leader LAST and give all messages sequence number N and is incremented every time a message is received.

```
{msg, N, Msg} ->
  Master ! Msg,
  slave(Id, Master, Leader, N+1, {msg, N, Msg}, Slaves, Group);
```

But the problem here is there could be duplicate messages in some of the slaves which have received message by the old leader as well. To solve this problem, we add a clause to check if the sequence number of a particular message is less than the sequence number of the last message received by that slave. If so, it takes the message, else it ignores.

```
{msg, I, _} when I < N ->
  slave(Id, Master, Leader, N, Last, Slaves, Group);
```