

Introduction

The system is developed to implement and demonstrate a group membership management service that provides the members of a group with a consistent view of its group's membership at any given time and tolerates full range of failures that commonly arise in distributed systems architecture, e.g., process failure, lost messages and network partitions. The system is based on peer-group architecture deployed in a Star Topology.

Specification

- Infrastructure Specification:
 - Communication Protocol: UDP
 - Group Structure: Peer-Group Structure
 - Topology: Star Topology
- Functional Specification:
 - All processes in a group communicate with all the other processes via UDP multicast protocol.
 - All the processes are in an agreement that everyone will keep their own record of the group view and will regularly broadcast their 'keep alive' (*Heartbeat) messages to all the other members of the group.
 - The processes read these 'keep alive' messages from the other processes and update their group views whenever required.
 - The processes also keep a track of the last time an individual process in the group sent a 'keep-alive' message. If it has been a while since the last time a certain other process (sender) had sent its "keep-alive" message then the processes in the group assume that the (sender) process is no longer alive and they update their respective group views by removing the process from their view.
 - Any external process wishing to join the group sends a join message with its ID on the multicast port and starts listening on

the same port for messages from other processes in the system. Thus, becoming a part of the group satisfying the requirement of Joining the Group.

- In the 'Heartbeat' message and the 'Join' message along with the ID the processes also send a counter of the message. Other processes in the group keep a track of this counter for every process and only act on messages from the processes who's counter is greater than the ones stored in their local view. The counter acts as a logical clock to identify the sequence of messages mitigating the inconsistency that may arise due to the unreliability of UDP protocol.
- The processes are also regularly printing the group view onto the console to constantly show the present state of the group.
- Limitations in the system:
 - The processes cannot reliably display the sequence of membership changes in the group view.
 - A process can be a part of multiple groups and maintain different group views for different views. However, the only limitation in this implementation is that a process can only join multiple groups together at the same time and leave all the groups at the same time. Processes can't choose to leave a single group if they are a part of multiple groups.
 - There is no concept of group creation in this implementation. Since it is a P2P motivated architecture, as soon as the process subscribes to a port and if it's the first process for that port it creates a group automatically, other processes then bind to the same port to be a part of the same group.
 - Another issue caused due to UDP message loss is partial link failure. It may so happen that during a scenario, a process in a group is not able to receive messages from other processes but is able to transmit its 'keep alive' messages to them, this leads to a partial link failure. Such a scenario happens when there is message loss during downlink but the process is successful in uplink

transmission. In such a case the other processes receive the 'keep alive' message from this process so in their group view this process is still a part of the group, however the process facing partial link failure assumes that the other processes have left the group since it does not receive any messages from them. This leads to an inconsistent view of the group among all the processes

Architecture

- Technical Architecture:

1. Process Group Structure:

Process Group Structure is based on Peer-group structure where all group communication is directed from processes in the group to all the members of the same group. There is no leader that is responsible for maintaining the group views. The group view is maintained by the mutual agreement among all the processes by keeping a track of the presence of all the other processes (nodes) present in the group. This adheres to the distributed system concept where there's no single point of functionality that maintains the entire group view.

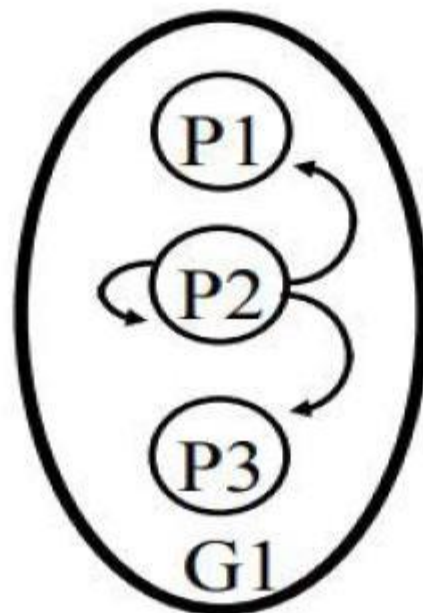


Figure 1: Peer Group Structure

2. System Topology:

The peer group is structured in a star topology, where each process communicates via multicast protocol. Thereby every message sent by a process, is received by every other process (Node) belonging to the same group as shown in “Figure 2” below.

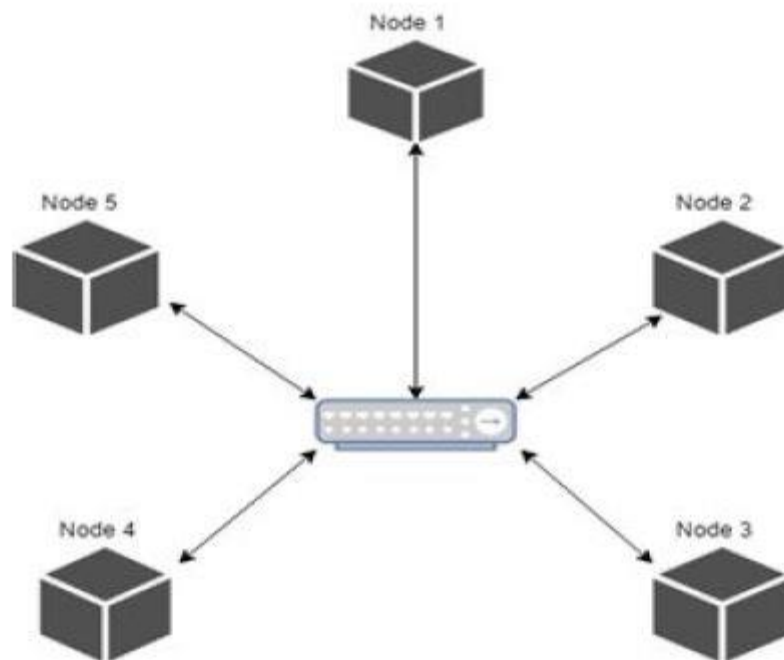


Figure 2: Network Architecture

3. Classes:

The classes used to formulate the architecture and functionalities are depicted as follows:

GroupManagementApplication:

This class is responsible for initiating the application over Spring Boot framework.

Peer:

This is the main class that accepts the input on the console and forks the Manager thread.

Manager Thread:

This class is the Manager class that depending on the inputs provided in the console invokes three main threads, i.e., TimerThread, ReadThread, SendHeartBeat, for each of the processes getting spawned with respective port numbers and initiates the group(s) creation process.

TimerThread:

This thread class is responsible for continuously checking for processes that stopped messaging. If no “heartbeats” are received from a process (member of a group) for a specific time period then that process is assumed to have left the group or is dead and accordingly the group view is updated by the removal of that process. Also this class is responsible for checking when a process should start sending heartbeats,i.e, if there is only one member in a group(the process itself) ,the process would wait for someone to join the group before initiating the heartbeat.

ReadThread:

This thread class is responsible for maintaining a consistent group state view at any given point of time. It continuously listens to the port (group) for any incoming multicast message, except for the messages sent by the process itself as source, and on receiving the message parsing, further processing of the message is done to know the nature of the multicast message, viz, is it a join request from a process or leave request or simply “keepAlive” heartbeat message.

SendHeartBeat:

This class is responsible for generating a heartbeat message and sending it across (multicast) when other processes also exist in the same group.

Implementation

Each process connects to a specific multicast port and then communicates with the processes that are also connected to the same port via IP multicast messages.

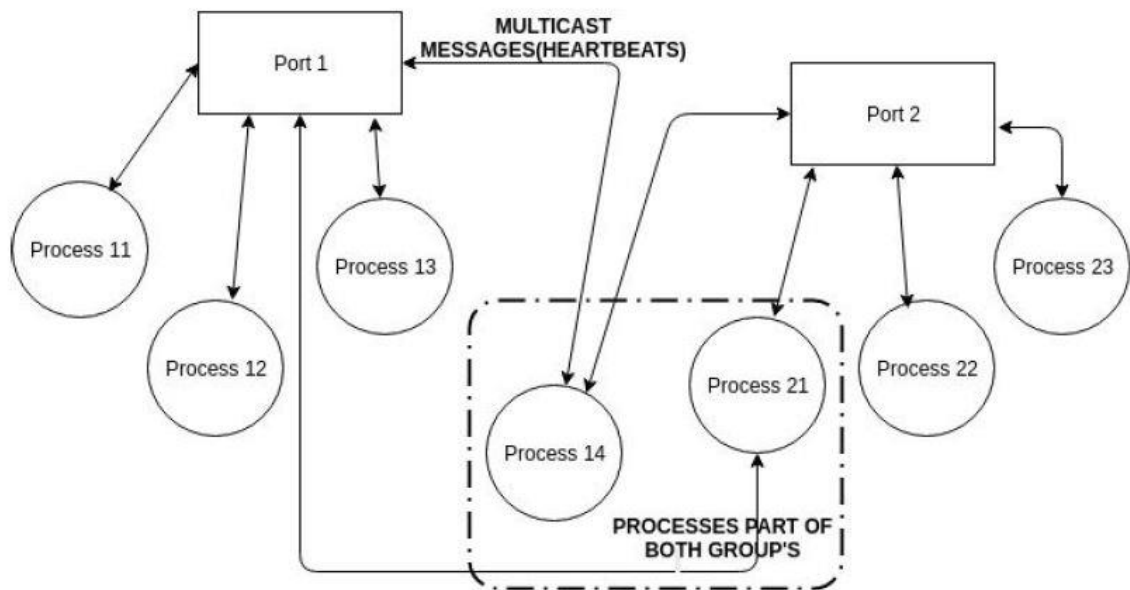


Figure 3: Network Architecture

The processes connecting to the same port form a group. A process can be connected to multiple groups by connecting to multiple ports, for example Process 14 and Process 21(as depicted in Figure 3 are members of both groups).The algorithms formulated for the implementation are as follows:

- Heartbeat algorithm
 - When part of a group, each process sends its heartbeat to all the other processes within the same group.
 - A process updates its local view of the group on receiving multicast message (heartbeat) depending on its logical clock value and the incoming heartbeat counter value. The algorithm for the same is depicted as :

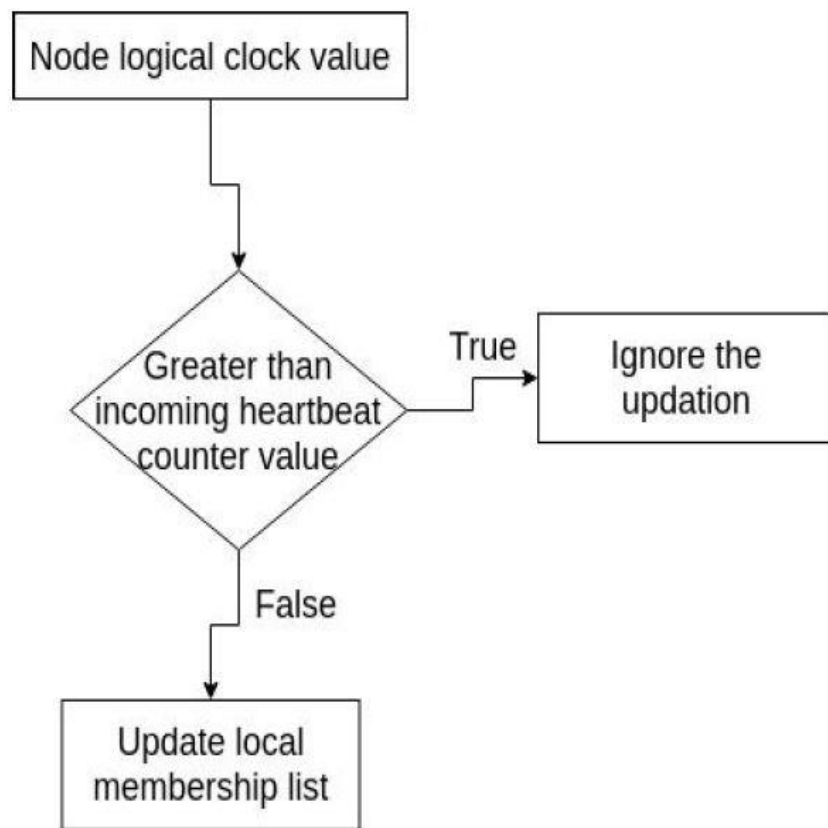


Figure 4: Heartbeat Logic

- Failure detection algorithm
 - Every process A keeps a track of the time when another process B in the group last sent a message.
 - Process A, after regular interval of time, checks if it has been more than a specified amount of time since process B last sent its heartbeat message.
 - If it has been more than a certain threshold since process A received a keep alive message from process B then process A assumes process B has died or left the group and removes process B from its group view.
 - This time period is always updated by process A whenever it receives a heartbeat message from process B.

- All the processes in the group follow the same procedure for all the other processes in the group.
- UDP's flaw mitigation algorithm
 - To counter the non-sequential delivery of packets in UDP, a simplistic version of logical clocks was implemented by gaining motivation from Lamport's logical clock and also vector clocks.
 - The system updates the heartbeats state only if the heartbeat counter value is greater than that of the logical clock value of the process (which is the recipient heartbeat counter).

Testing

Test Cases for functionalities:

1. Display of Group View:

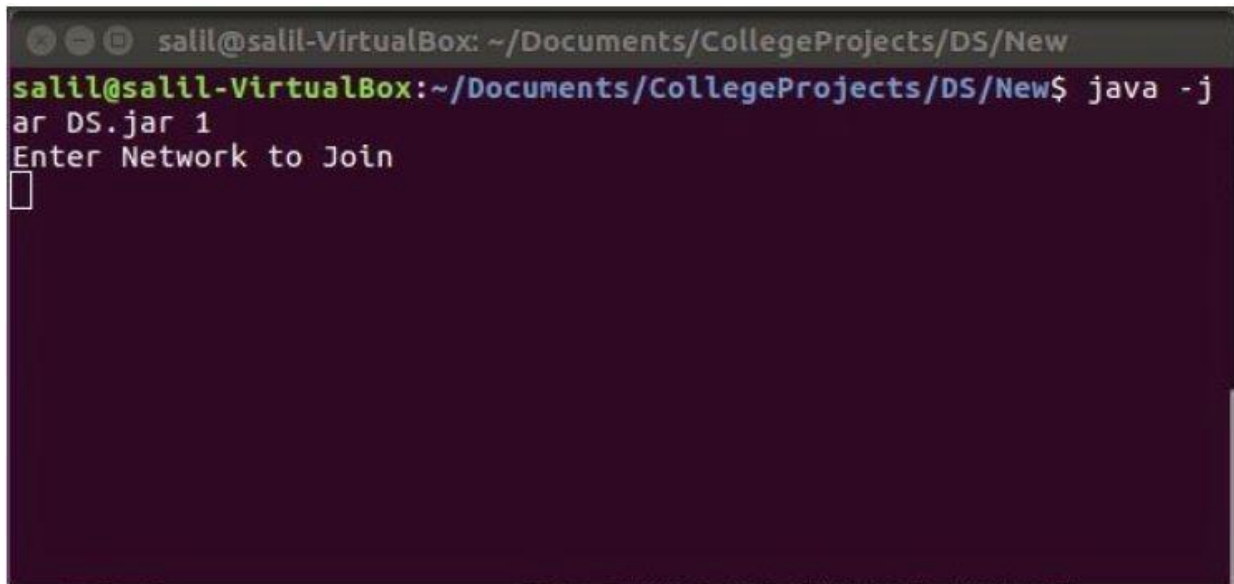
- All members regularly display the state of their own view of the group which can be interpreted as follows:
 State: **From Network: 8900, {1=423, 2=456, 3=153}**
 - 8900 - Identifies the network (Group) the state is of.
 - {1=423, 2=456, 3=153}- Identifies the state of the group.
 - 1=423, 2=456, 3=152- Indicates the state of the individual members of the group.
 - 1,2,3 are the IDs of the group members respectively.
 - 423, 456,152 are the logical clock record maintained for every member.

2. Member Process joining/creating the group:

- When a process is up, it decides which group (Network) it wants to join (Figure 5, 6).
- Once this process joins a group it starts getting messages from the current members of the group and the other members start

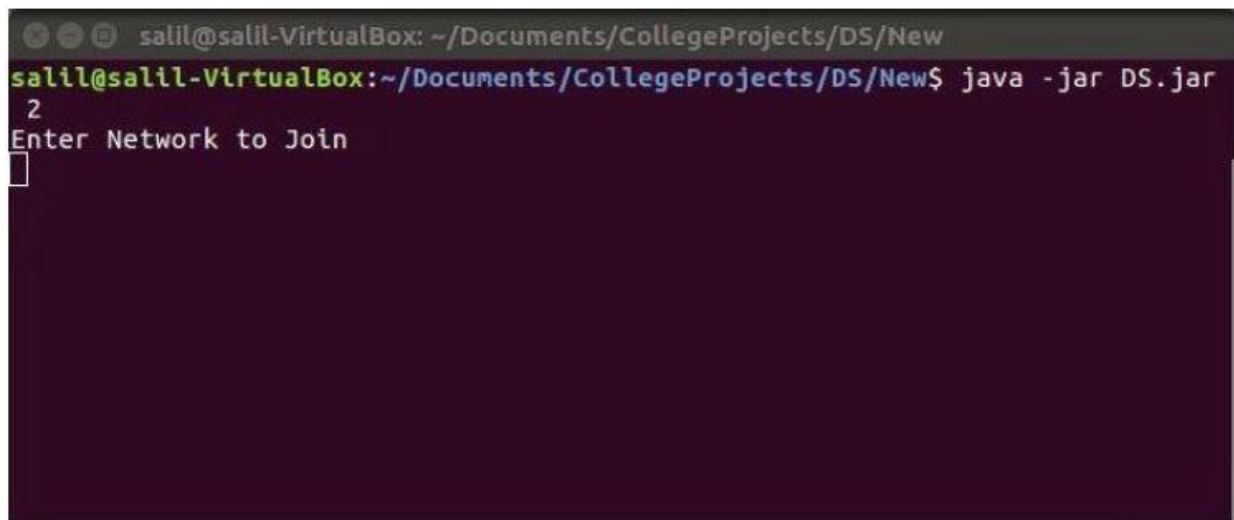
getting the heartbeat messages from this new process (Figure 7, 8).

- A process is able to join multiple groups as well (Figure 9).

A terminal window with a dark background and light-colored text. The title bar shows 'salil@salil-VirtualBox: ~/Documents/CollegeProjects/DS/New'. The prompt is 'salil@salil-VirtualBox:~/Documents/CollegeProjects/DS/New\$'. The user has entered 'java -jar DS.jar 1'. The program output is 'Enter Network to Join' followed by a small white square cursor.

```
salil@salil-VirtualBox: ~/Documents/CollegeProjects/DS/New
salil@salil-VirtualBox:~/Documents/CollegeProjects/DS/New$ java -jar DS.jar 1
Enter Network to Join
█
```

Figure 5: Process 1 ready to join Group (Network)

A terminal window with a dark background and light-colored text. The title bar shows 'salil@salil-VirtualBox: ~/Documents/CollegeProjects/DS/New'. The prompt is 'salil@salil-VirtualBox:~/Documents/CollegeProjects/DS/New\$'. The user has entered 'java -jar DS.jar 2'. The program output is 'Enter Network to Join' followed by a small white square cursor.

```
salil@salil-VirtualBox: ~/Documents/CollegeProjects/DS/New
salil@salil-VirtualBox:~/Documents/CollegeProjects/DS/New$ java -jar DS.jar 2
Enter Network to Join
█
```

Figure 6: Process 2 ready to join Group (Network)

```
salil@salil-VirtualBox: ~/Documents/CollegeProjects/DS/New
salil@salil-VirtualBox:~/Documents/CollegeProjects/DS/New$ java -jar DS.jar 1
Enter Network to Join
8900
Node-1 Started
New Node Added:2
From Network: 8900, {2=2, 1=2}
From Network: 8900, {2=3, 1=3}
From Network: 8900, {2=4, 1=4}
From Network: 8900, {2=5, 1=5}
From Network: 8900, {2=6, 1=6}
```

Figure 7: Process 1 joined to Group (Network) 8900 with Process 2.

```
salil@salil-VirtualBox: ~/Documents/CollegeProjects/DS/New
salil@salil-VirtualBox:~/Documents/CollegeProjects/DS/New$ java -jar DS.jar 2
Enter Network to Join
8900
Node-2 Started
From Network: 8900, {2=0, 1=2}
From Network: 8900, {2=2, 1=3}
From Network: 8900, {2=3, 1=4}
From Network: 8900, {2=4, 1=5}
From Network: 8900, {2=5, 1=6}
```

Figure 8: Process 2 joined to Group (Network) 8900 with Process 1.

```

salil@salil-VirtualBox: ~/Documents/CollegeProjects/DS/New
^Csalil@salil-VirtualBox:~/Documents/CollegeProjects/DS/New$ clear

salil@salil-VirtualBox:~/Documents/CollegeProjects/DS/New$ java -j
ar DS.jar 1
Enter Network to Join
8900
Node-1 Started
New Node Added:3
From Network: 8900, {3=2, 1=2}
From Network: 8900, {3=3, 1=3}
From Network: 8900, {3=4, 1=4}
From Network: 8900, {3=5, 1=5}
From Network: 8900, {3=6, 1=6}
From Network: 8900, {3=7, 1=7}
From Network: 8900, {3=8, 1=8}

salil@salil-VirtualBox:~/Documents/CollegeProjects/DS/New$ java -jar DS.jar
2
Enter Network to Join
8901
Node-2 Started
New Node Added:3
From Network: 8901, {3=2, 2=2}
From Network: 8901, {3=3, 2=3}
From Network: 8901, {3=4, 2=4}
From Network: 8901, {3=5, 2=5}
From Network: 8901, {3=6, 2=6}
From Network: 8901, {3=7, 2=7}
From Network: 8901, {3=8, 2=8}
From Network: 8901, {3=9, 2=9}

salil@salil-VirtualBox:~/Documents/CollegeProjects/DS/New$ java -jar DS.jar
3
Enter Network to Join
8900,8901
Node-3 Started
Node-3 Started
From Network: 8900, {3=0, 1=2}
From Network: 8901, {3=0, 2=2}
From Network: 8900, {3=2, 1=3}
From Network: 8901, {3=2, 2=3}
From Network: 8900, {3=3, 1=4}
From Network: 8901, {3=3, 2=4}
From Network: 8900, {3=4, 1=5}
From Network: 8901, {3=4, 2=5}
From Network: 8900, {3=5, 1=6}
From Network: 8901, {3=5, 2=6}
From Network: 8900, {3=6, 1=7}
From Network: 8901, {3=6, 2=7}

```

Figure 9: Process 3 (Bottom) in Multiple-Groups (Network) 8900 and 8901 with Process 1 (Top-Left) in Group 8900 and Process 2 (Top-Right) in Group 8901

3. Leaving group / Identifying Process failure in group

- Whether a Process leaves a group or Fails, the group reacts in the same manner in removing the leaving/failing member from the group.
- The members of the group wait for a certain period of time before assuming the process has failed or left the group before removing it from their group view. (Figure 10).
- The wait period is implemented to prevent a scenario where if there is any message loss due to UDP protocol, the members of the group are not too quick to declare a process as failed and remove the process from their group view.

```

salil@salil-VirtualBox: ~/Documents/CollegeProjects/DS/New
salil@salil-VirtualBox:~/Documents/CollegeProjects/DS/New$ java -jar DS.jar 1
Enter Network to Join
8900
Node-1 Started
New Node Added:2
New Node Added:3
From Network: 8900, {3=0, 2=2, 1=2}
From Network: 8900, {3=2, 2=2, 1=2}
From Network: 8900, {3=2, 2=3, 1=3}
From Network: 8900, {3=3, 2=3, 1=3}
^Csali@salil-VirtualBox:~/Documents/CollegeProjects/DS/New$

salil@salil-VirtualBox: ~/Documents/CollegeProjects/DS/New
From Network: 8900, {3=2, 2=2, 1=2}
From Network: 8900, {3=2, 2=2, 1=3}
From Network: 8900, {3=3, 2=3, 1=3}
From Network: 8900, {3=3, 2=3, 1=4}
From Network: 8900, {3=4, 2=4, 1=4}
From Network: 8900, {3=5, 2=5, 1=4}
From Network: 8900, {3=6, 2=6, 1=4}
From Network: 8900, {3=7, 2=7, 1=4}
From Network: 8900, {3=8, 2=8, 1=4}
From Network: 8900, {3=9, 2=9, 1=4}
From Network: 8900, {3=10, 2=10, 1=4}
Removing key: 1
From Network: 8900, {3=11, 2=11}
From Network: 8900, {3=12, 2=12}
From Network: 8900, {3=13, 2=13}

salil@salil-VirtualBox: ~/Documents/CollegeProjects/DS/New
From Network: 8900, {3=0, 1=2}
From Network: 8900, {3=0, 2=2, 1=2}
From Network: 8900, {3=2, 2=2, 1=3}
From Network: 8900, {3=2, 2=3, 1=3}
From Network: 8900, {3=3, 2=3, 1=4}
From Network: 8900, {3=3, 2=4, 1=4}
From Network: 8900, {3=4, 2=5, 1=4}
From Network: 8900, {3=5, 2=6, 1=4}
From Network: 8900, {3=6, 2=7, 1=4}
From Network: 8900, {3=7, 2=8, 1=4}
From Network: 8900, {3=8, 2=9, 1=4}
From Network: 8900, {3=9, 2=10, 1=4}
From Network: 8900, {3=10, 2=11, 1=4}
Removing key: 1
From Network: 8900, {3=11, 2=12}
From Network: 8900, {3=12, 2=13}
From Network: 8900, {3=13, 2=14}
From Network: 8900, {3=14, 2=15}
From Network: 8900, {3=15, 2=16}

```

Figure 10: Process 1 (Top-Left), 2 (Top-Right), 3 (Bottom) in Group (Network) 8900 with process 2 & 3 removing process 1 after its failure

Appendix

- UDP was used to understand all the network related issues and provide solution.
- Frameworks for implementation of Group Management System already exist for Java which handle all the failure scenarios but was not used as the intention of the project was to dwell into the intricacies of the network.
- In the quest to find optimal solution we came across some state of the art concepts like
- Adaptive Peer Sampling, addressing the issues of light-weight gossip based implementation, which can be used to resolve message loss experienced in UDP to an extent.