1. Design

1.1 Structure

There are five components of our Distributed Group Membership, MmpIntroducer, MmpReceiver, MmpSender, MmpServer and MmpJoiner. When we start a MmpServer, it joins the group through the introducer, initializes the membership from the introducer, and launches two threads: the MmpReceiver and the MmpSender. MmpReceiver and MmpSender run separately at each machine, to receive and send pings. There are two logs in each machine, one manage by MmpReceiver and one by MmpSender. MmpIntroducer is the introducer, who introduce all new join nodes and broadcast the join message to every member (by calling the MmpJoiner). MmpIntroducer is also an extension of the MmpServer, so that it launches three threads while start: the MmpJoiner, the MmpReceiver, and the MmpServer. MmpJoiner is an extension of a thread, which adds the newly joined machine into the introducer's local membership list and broadcast the joined information to all available machines.
Between machines there are five types of messages: “JOINED”, “LEFT”, “FAILED”, “PING”, and “ACK”.
MmpReceiver is defined in MmpReceiver.java with two helper functions.
i). execMessage(DatagramPacket): This method analyzes the received message and takes corresponding steps. For instance, if the message is a “PING”, it sends an “ACK” to the sender machine; if the message is "ACK", it marked the sender machine as normal; if the message is "JOINED", it add the joined machine into the membership list; if the message is "FAILED", it removes the machine from the membership.
ii). writeToLog(String): This method analyzes and writes the received message into the receiver log, such as times of status changes (join, leave or failure).
MmpSender is defined in MmpSender.java with four helper functions.
i) sendPing(): send UDP packets with ping information to the chosen machine.
ii) broadcastToAll(String): broadcast UDP packets with failure or leave messages to all machines in the local membership.
iii) writeToLog(String): similar to MmpReceiver, write log to the local sender. The sender log includes the information of failure detections. iv) getMonitorList()
iv) getMonitorList(): get the following three available machines in a virtual ring. Details will be discussed in 1.3.

1.2 Message Type

All messages are marshaled into byte arrays, which is platform independent.

1.3 Number of Monitors

We design our algorithm following the SWIM style. Imagine the machines in a ring. The ring is present in each local membership list. Each machine pings the following **three** available machines in their membership list to monitor their availabilities. While pinged, each machine sends an acknowledgment to the sender machine. While a machine is detected to be dead (no ACK received for 500 ms), the detection machine broadcast the death to all other machines. Each machine broadcast its leave before it voluntarily leaves the group.

1.4 Scalability

Since our design uses a ring, the functionality would not be affected if we scale up the system. Each failure would be detected in the same time as in a smaller scale, and since a failure is broadcasted after detection, every machine will receive the failure approximately the same time as it is in a smaller scale.

1.5 MP1 Usage

Since we We create two logs in each machine, a sender log and a receiver log. The logs are populated with any change of status in the membership list (leave/failure/join). We integrate MP1 into our MP2 solution, to retrieve necessary information for debugging purposes. Therefore, we don't need to stare at the screen all the time, trying to see if the program works fine.

2 .Measurements

2.1 Background Bandwidth Usage

64 bytes per second (ping 32 bytes, acknowledgment 32 bytes). We set the timeout period to be 1000 ms.

2.2 Average Bandwidth Usage

The bandwidth usage depends on the number of machines. Since we broadcast when join/leave/ failure happen. The bandwidth usage when a node joins the group is 28+k*28+k*35 bytes, with k as the number of original nodes in the group. The first 28 bytes are used in the introducer when accepting the join message. The 28*k is also used in the introducer for sending the newly joined node the membership list. The 35*k is to broadcast the join message to all other nodes. For example, if k=9, the bandwidth usage is 28+9*28+9*35=595 bytes.

The bandwidth usage when a node leaves the group is 35*k, where k be the number of other nodes in the group. If there are 10 nodes in the group and one decided to leave, the bandwidth is 315 bytes.

The bandwidth usage when a node fails is 32*3+3*(32*k), where k be the number of other nodes in the group. The first term 32*3 denotes the bandwidth usage of the direct failure detection (since three nodes will detect its failure immediately). In these three nodes, each node broadcast failure detection to all other k nodes, which has a 3*(32*k) bandwidth usage. When k=9, the bandwidth usage is 960 bytes.

2.3 False Positive Rate

We experiment 20 times and plot the result below. We pick one node to see if it detects other nodes' failures. For n=2, there are only two cases, 0% or 100%. For n=4, there are four cases: 0%, 33.33%, 66.67% and 100%. In general, as the packet loss grows, the false positive rate increase, and the rate for N=4 is larger than it for N=2. This is because at N=2, one nodes monitors another, so if there is x% percent of packet loss, then the false positive rate is also approximately x%. For N=4, the false positive rate becomes lower since there are more nodes facilitate failure detections.

For N=2, the average false positive rate is 5%, 10% and 25%. For N=4, the result is 1.67%, 5.00%, 8.33%.