# CS425, Distributed Systems: Fall 2019 Machine Programming 2 – Distributed Group Membership

## Design

For this MP, we designed a distributed system with a virtual ring.

**Introduce and join**

When the first node joins the ring, it will scan all introducers in a hard-coded introducer list which contains IP address and port for all possible machines in the system. It waits for one second and if it did not receive an ACK from any of the machine, it will Join the ring with a membership list which only contains itself. For the rest of nodes, they will scan all machines and join the first introducer that replies. On the introducer side, when it receives a join message, it will send its membership list as well as a new id to the new node, and broadcast the new id, ip, port and join time to all other nodes in the ring.

**Heartbeat and monitor**

In our ring, each node will send heartbeat to three successor every 1.5 seconds and monitors heartbeat from its ancestor every 4 seconds. If the ring contains less than three nodes, it will monitor all nodes except itself. Whenever a node receives message for add new member, it will check if this node is inserted into three of its ancestors, if so, it will start monitor it. At this time, the fourth ancestor will stop sending heartbeat to it, and it will realize this is not a failure. Same applies to send heartbeat, if the monitor becomes its fourth successor, it will stop sending heartbeat. And if new node inserted into its successor, it will start send heartbeat to it. When a monitor did not receive heartbeat for 4 second, it will mark that id as failure. Then broadcast failed node id to all nodes in the ring. Then delete it from its own membership list.

**Leave**

A node leaves when it receives a linux signal sent by user. It will go into a signal handler and notify all nodes in the ring before it exits the program.

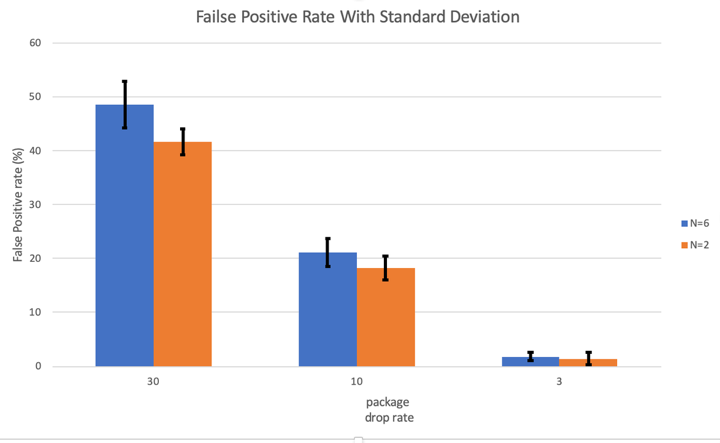
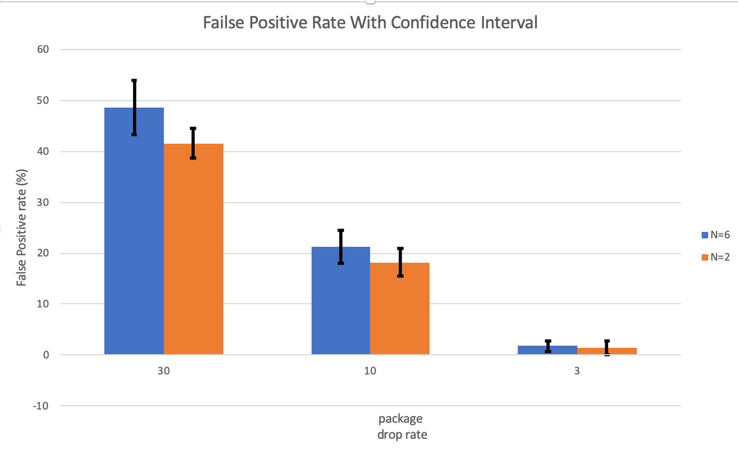
**Query Logger**

we used query logger to dump membership list of all machines. It helped us a lot when debugging. However, its usefulness is also depending on our ability to write logs in our program. We found many critical lines in our code which might be problematic and logging it helped us a lot when debugging

**Unit Test**

we wrote very comprehensive unit test code. Every time we made change to our code, we can directly run the test to make sure our code works as it used be. We also found many bugs by running unit test. Since unit test is separated for different tasks, we can focus on one bug and time which is very efficient way of solving problems.

Plot & Analysis

****

We set up the test by using 10 heartbeats for each test and check how many false positive happened. The decrease of package drop rate showed significant affect with the false positive rate. This is because false positive rate is related to package drop rate, when package dropped in accident for several times, the monitor did not receive message from that node for long enough time or 4 second in our case, it will treat that node as a failure node and broadcast this update to all other nodes. When we increase the number of nodes in the group to 6. All nodes send heartbeat to not just one but three other nodes, so either one of them did not receive heartbeat for long enough time will broadcast to everyone that the node failed. Thus N=6 have a slightly higher false positive rate. The standard deviation and confidence interval are very similar data both showing the distribution of our measurement we can see that it doesn’t vary a lot.