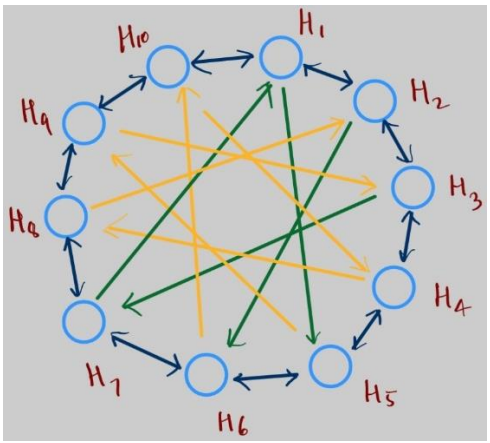
CS 425 – MP2 Report

Group 69 – mgoel7 and bachina3

Design and Algorithm

We have gone with a simple distributed architecture where every node is part of a ring and pings its predecessor and successor. In addition, each node also pings another node (4th successor) which is fixed. The ring topology that we have fixed is given below:



Every node has a full membership list and also knows whom to ping. We have 3 message types that a node can send, PING, ACK and INTRODUCE. With each message type, we piggyback the full membership list as well. This ensures that all nodes are updated about each other at all times. We are using the asyncio module of python to run everything in a single thread.

The node sends a PING every 2.5 seconds and waits for a PING_TIMEOUT (2 seconds) period until it receives an ACK. Before sending the ping, the node updates it own timestamp and status. When a node receives an ACK, it will merge the current membership list and received membership list by updating the timestamps. Now there are 2 special cases:

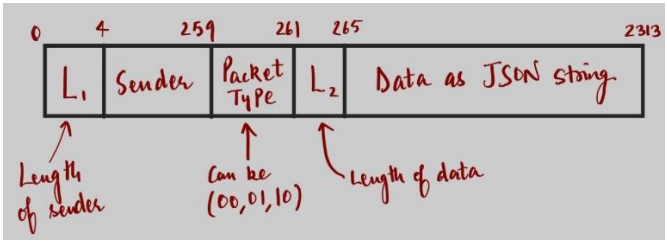
When node crashes or leaves

When a node (A) crashes/leaves, its immediate ping nodes fail to receive an ACK. In this case, they will mark the status of A as offline and propagate it forwards to other nodes in the next PING. After some time, every node has the updated membership list with A marked as offline. Then after a certain CLEANUP_TIME (10 seconds), every node will delete A from its membership list. When 3 such simultaneous failures/leaves happen, we initiate a topology change in the system to maintain completeness.

When node joins

When a new node A wants to join the system, it sends an INTRODUCE ping to the introducer. The introducer then gets to know that there is a new node in the system and sends the current membership list to A. Thus, after a few pings, every node knows that A is a new node in the system and A knows the other members in the group. Then, we initiate topology change at each node to maintain the ring topology at all times.

We are using a platform independent custom message format of length 2313 bytes. The packet format is given below:



The algorithm is 100% complete for M = 3, i.e., it can handle 3 simultaneous failures at a time. This is due to the fact that each node pings 3 nodes and as soon as 3 failures happen, the topology is changed and the ring structure is established again. The algorithm is also scalable to large N as each node only pings 3 other nodes. Therefore, even for large values of N, the load on each node will not increase.

Performance Testing

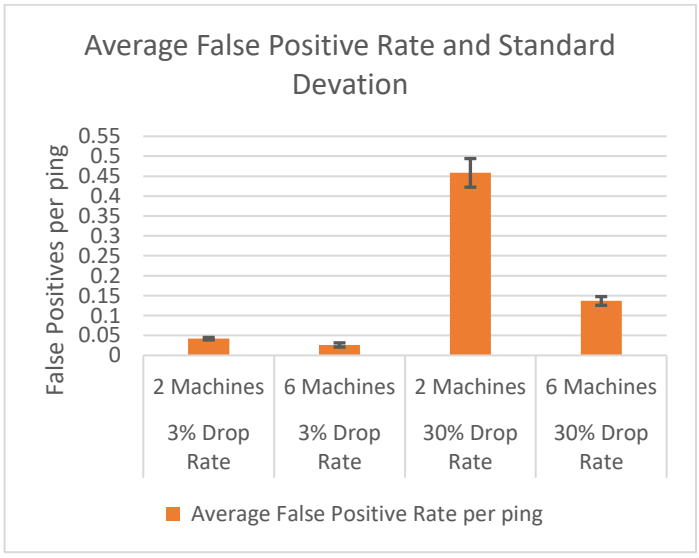
For debugging and performance testing, we successfully utilized MP1 log retriever for fetching the logs and comparing the timestamps for log entries of each machine. We marked our logs as [INFO], [ERROR] and [DEBUG] for easy identification. We tested the performance of our system on the following parameters.

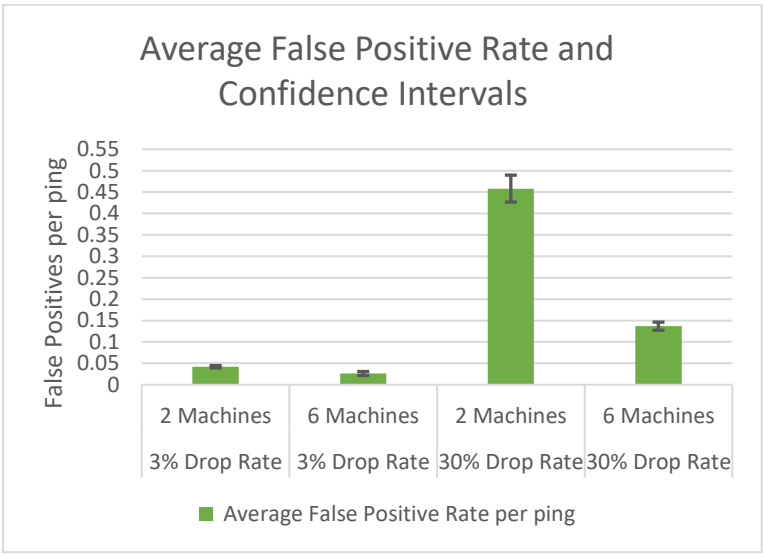
1) **Background Bandwidth Usage**
   To calculate Bps we calculate the (total number of bytes sent/total time elapsed) for each node and then sum the Bps for all the 6 machines in the system. The average background bandwidth comes out to be **33118 Bps.**

2) **Bandwidth Usage For Node Joins And Crashes/Leaves**
   In a similar fashion to part 1), we calculate the Bps whenever a new node joins or leaves.

   - Bps when node leaves (only 5 current nodes in the system) – **24442 Bps.** This is expected as now there are less number of pings and Acks being exchanged in the system.
   - Bps when node joins (6 nodes in the system) – **28992 Bps.** This is also expected as the newly joined node has exchanged less messages than the others, thus bringing the average down. However, after a certain time the Bps will eventually come back up to Bps in part 1).

3) **False Positive Rates**
   We calculated the false positive rates by calculating the number of times it marked an active node as inactive divided by the total number of pings it sent. We took 5 measurements for each N = 2,6 and message drop rate as 3% and 30%. From our measurements, we generated the following plots.





We observe that the confidence intervals and standard deviation is almost the same, meaning that most of our data lies within 1 standard deviation of the mean. We also observed that the false positive rate for 2 machines is higher than 6 machines because when there are only 2 machines, each machine detects each other thereby increasing the average. Also, since we suspect a failure on every message drop, our false positive rate closely matches 3% and 30% for each case.