CS 425 Distributed System MP2 Report

Group 43: Tzu-Bin Yan (tbyan2), Yash Saboo (ysaboo2)

I. Design

In this MP, our design of the server program is composed of four major parts, namely the UI thread, the main thread, the gossip thread and the heartbeat thread. The UI thread implements the "command-line commands" interface required in the demo instruction that allows one to list the membership list, show the server ID, join the peer network if not yet joined and peacefully leave if already joined. The main thread is tasked with starting up the UI thread as well as based on the passed command of the UI thread, initializing the server and starting the gossip and heartbeat threads or notifying the gossip and heartbeat threads to leave the peer network. The gossip thread spreads the membership list updates (join, leave and fail messages) around the peer network using push gossips to update the membership list at each node, as well as listen for the update messages to update its own membership list. The heartbeat thread periodically sends heartbeat messages to the three immediate successors (immediate in the sense of the next three nodes in the network with IDs in dictionary order larger than the heartbeating node's ID) directly to notify them that the node is still alive. In addition, the heartbeat thread listens for heartbeat registrations from other nodes in the network that register itself with the node and have the node monitor its heartbeat to see if it is down or not.

The reason that this design scales to large N is that based on this design, an additional node to the network does not impact the heartbeat workload of each node (the new node heartbeats to the three immediate successors only, and that the remaining node will switch their heartbeat targets to the new node if the new node is one of its three immediate successors, effectively leaving each node in the network in the state of heartbeating to three nodes and receiving heartbeats from three nodes, even when the network size is large). On the other hand, the additional node DO impact the gossip workload of each node. Nevertheless, due to the nature of the gossiping, the load experienced by a node with respect to gossiping per update message is of complexity O(log(N)), which is scalable. Note that for the case that there are many updates propagating in the network, our design is still scalable since the number of update messages sent out by each node per gossip round is limited to a maximum size rather than sending all update messages per gossip round. This effectively slows down the dissemination of update messages, but at the same time reduces the bandwidth requirements in case of excessive update messages.

For the marshaled message format, all gossip messages and heartbeat messages sent onto the network are either directly given in byte values (for IP in gossip/heartbeat message content, state in gossip message and message type in heartbeat message), or are fixed-length variables (for TTL in gossip message and timestamp in heartbeat message) encoded in network byte order (i.e. big endian), which makes it platform-independent.

II. MP1 Usefulness

The way that MP1 was used in this MP was by having them running as a separate process on each of the VMs and serving the log file of the MP2 server process, after which we queried the MP1 servers using the MP1 client program to obtain the content of all log files across VMs. MP1 was useful in this MP for debugging since with MP1, we can check the logs of all remote server locally without have to firing up a number of terminals and ssh to all the VMs to see the logs, which is very time consuming.

III. Measurements

Measurement for (i) and (ii) are done using the tcpdump command with the size of packet headers included in the bandwidth usage calculation.

(i) Background bandwidth usage is **3,242.73** Bps for the whole system and **540.46** Bps for a node.

(ii) Average bandwidth usage whenever a node joins, leaves or fails (including background bandwidth usage)

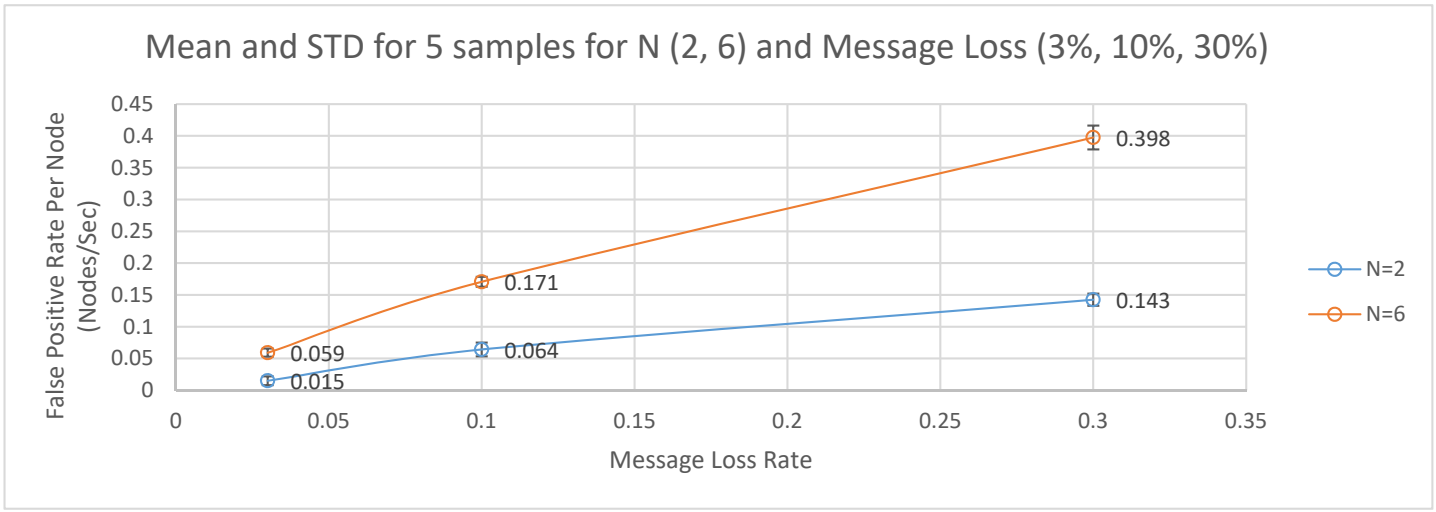
are as follows

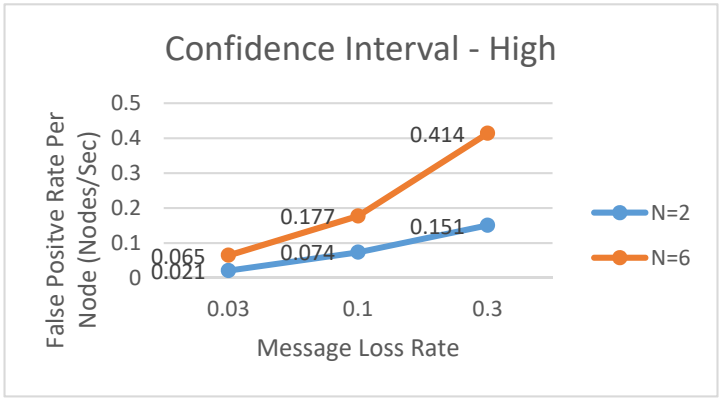
Join: **692.09** Bps for a node; Leave: **673.87** Bps for a node; Fail: **676.24** Bps for a node

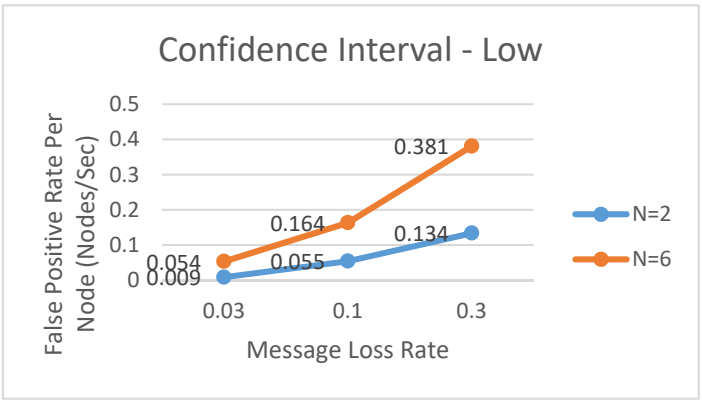
(iii) The graph for the false positive rate for membership list is given as follows:

a) In this measurement we changed the number of heartbeats per timeout period to 2 rather than the original 4 to decrease the amount of time needed to find a false positive, which isn't likely when using 4 heartbeats.

b) Confidence Interval: z*(Standard Deviation/sqrt(N)), where z = 1.96 (since we chose confidence level = 95%), n = # of samples = 5.







Analysis:

1. The increase in message loss rate resulted in the increase in number of false positive detected, regardless of N. This is reasonable since higher message loss rate increases the chance of droping heartbeat messages, thus increasing the chance of false detection.
2. The False positive rate is higher (approximately 3 times) for N=6 when compared to N=2. This is expected since for N=6, a machine is monitoring more nodes (3 nodes in our setting), which thus increases the expected number of false detections made by a machine.
3. Observe that for N=2, we have that the false positive rate is in fact much larger than the square of the message loss rate, which is the probability for a false failure detection to occur under two heartbeats. We believe that the reason for this is that the design of our program for this MP is a bit too complex with many threads doing different tasks being executed concurrently. Therefore, although a portion of the heartbeat messages arrived in time (especially those arrived at the server at moments before the timeout time), due to that the process may still be running other threads, they may not be served in time to prevent false detection of the monitored peer.