ECE 428 MP2 Report (Group – 01)

Ribhav Jain (ribhavj2)
Jiyu Hu (jiyuju2)

Classes:

Node – This class stores all the necessary attributes for a Node and has methods to help the Node communicate with other nodes.

Message – This custom class helps form messages with all necessary attributes (TTL, target, etc.) and then serialize them into Strings.

ListEntry – This custom class is used to store Node entries in the membership list and has methods to return Node IP, timestamps and IDs.

HeartbeatRcvr – This class is used to run threads on and receive heartbeats from the specified Nodes.

HeartbeatSdr – This class is used to run threads to send heartbeats to the specified Nodes.

Algorithm:

Our design used a ring structure for all the nodes in the system. Each membership list is sorted in an ascending time order, with the first entry being the earliest to join and the last entry being the latest to join. Since we know that at any time only 3 VMs can fail simultaneously, each node tracks 3 different nodes. This ensures time bounded completeness while being efficient. We use the membership list to tell us how the ring is structured, i.e. a node at index 3 joined right after a node at index 2. Similarly, we can create a full ring.

We have structured heart beats to be received from the next 3 neighbors (i.e. the 3 nodes that are on the right side on the node). This could mean that the latest node added is receiving heart beats from index 0, 1 and 2 since those are the nodes to its immediate right in the ring structure/ membership list. Similarly, we have structured heart beats to be sent to the previous 3 neighbors (i.e. the 3 nodes that are on the left side on the node). This could mean that the node added earliest is sending heart beats to the last 3 indexes of the membership list since those are the nodes to its immediate left in the ring structure. We restructure our ring and everyone's new respective neighbors once we detect a change in the ring. The heart beating time period is 0.5s.

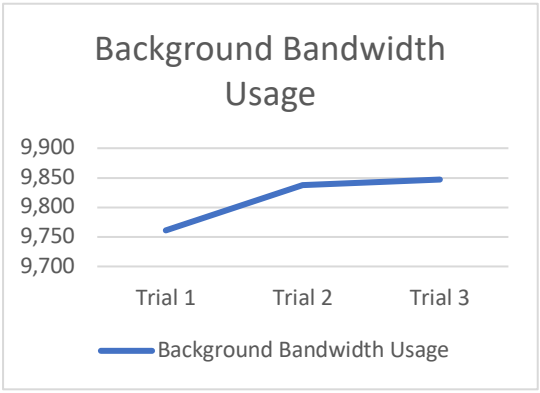
This algorithm scales to large N, since no matter how big N is, we only track 3 nodes at each node. Thus, our bandwidth usage per node does not increase with an increase in the number of nodes. We have also used TTL to ensure that messages die out after sending and are not repeatedly sent to machines. Another measure we have used to implement scalability is to maintain a cache of messages received, so if a new message is received about a new/dead node which was processed recently, we drop the request. We have also piggy backed messages, thus if a node is to be added or deleted, we send the message as the heartbeat, thus reducing bandwidth usage.

We have used a custom class Message to marshal our messages. We use the custom Message object to store the target machine, sender machine, TTL etc. We then serialize this data in a String format before sending it. Once, it is received at a VM, it is de-serialized and parsed again to ensure safe and efficient transfer of data using this common protocol.

MP1 was useful to debug MP2, since we could immediately grep all logs and see who had detected failures and who hadn't. We were able to grep a certain term and see whether that message had propagated to all machines or not. Thus, being able to grep all logs at once helped us to debug and verify the completeness of our system.

i) The background bandwidth usage as measured per machine for 6 machines is –

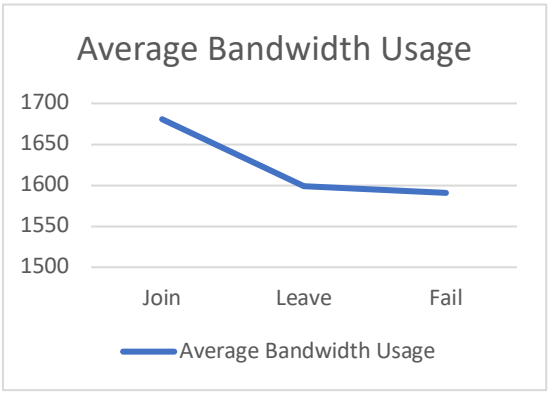
a. 1628.9 Bytes / second per machine ≈ 9,761 Bytes / second for system
b. 1639.76 Bytes / second per machine ≈ 9,838 Bytes / second for system
c. 1641.5 Bytes / second per machine ≈ 9,847 Bytes / second for system



We have a very consistent bandwidth usage across multiple trials ≈ 1,636 Bytes / second for 1 machine for heartbeats with period 0.5s. Thus, the background bandwidth usage for 6 machines is ≈ **9,820 Bytes**.

ii) Average bandwidth usage per machine for a 6-machine system when a node –

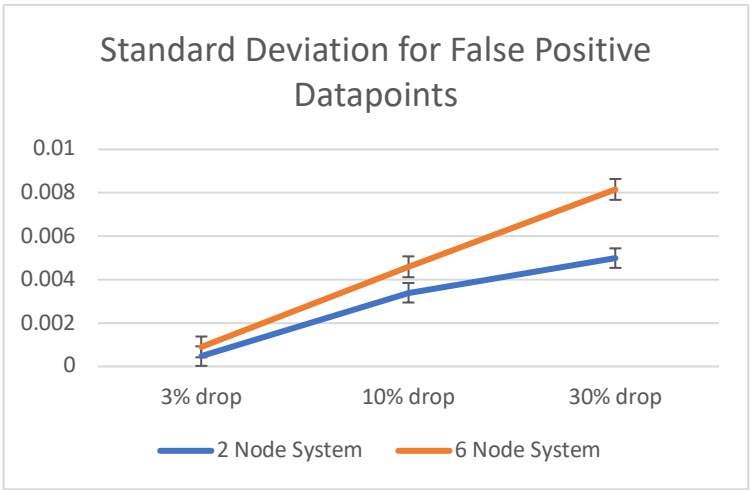
a. Joins – 1680.7 Bytes / second
b. Leaves –1599.3 Bytes / second
c. Fails – 1590.8 Bytes / second

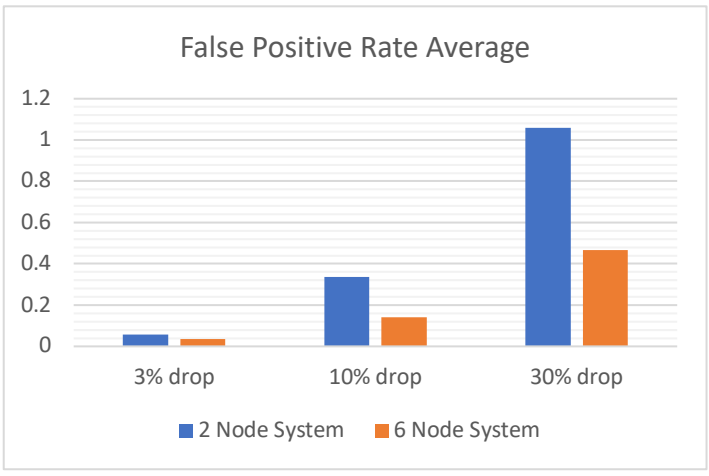


This clearly shows us how the bandwidth behaves during different behavior. We can see that when a node joins, the bandwidth remains nearly the same since the node ring is restructured and each node still sends and receives heartbeats from only 3 members. We also see that when a node leaves for fails, the average bandwidth does not change much since, the ring reconstructs.

iii) False positive rate of membership service (5 readings for 6 datapoints)

	2 Node System					6 Node System				
3% drop	0.02%	0.01%	0.13%	0.07%	0.05%	0.02%	0.03%	0.01%	0.01%	0.11%
10% drop	0.23%	0.47%	0.11%	0.01%	0.86%	0.33%	0.08%	0.10%	0.18%	0.02%
30% drop	1.75%	0.38%	1.22%	0.88%	1.07%	0.67%	0.45%	0.88%	0.22%	0.11%





We can see a trend that the false positive rate in the 6-node system is lower than the 2-node system. This aligns with our expectations since in the 6-node system we send heartbeats to 3 other nodes which makes our system much more stable and less prone to false positives.