CS 425 MP2: Distributed Group Membership | Sanath Nair (sanathn2), Praneetha Bhogi (pbhogi2)

Our system implements two membership and failure detection mechanisms, Gossip and Ping-Ack, both of which include an optional suspicion mechanism. The system is centered around MembershipLists containing information about all known nodes in the system, such as NodeId (an identification with the host, port, and timestamp), current status (alive, left, suspect, or dead), incarnation number, heartbeat counter, and failure detection mode. Through every heartbeat interval, this list is shared among k-random peers and is synchronized by comparing incoming timestamps or incarnation numbers with their local clocks to ensure newer information is updated and old information is deleted.

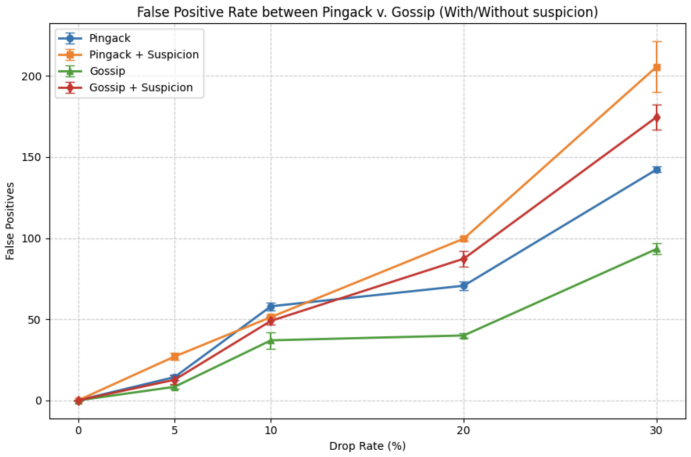
Each node functions independently and utilizes UDP sockets for sending and receiving messages. The Message objects serialize and deserialize the necessary information in the MembershipLists through custom binary formatting in message.cpp, which prioritizes size and speed. We also handle concurrency and synchronization via locking, ensuring safe reading and writing of the list. Therefore, despite constant exchanges, responses, and CLI commands for joining/leaving/switching, the membership state stays consistent.

We also implemented a “switch” command that allows nodes to dynamically change their strategy for failure detection at runtime. We supported four modes: Gossip, Gossip with suspicion, Ping-Ack, and Ping-Ack with Suspicion. Once the “switch” command is received at a certain node, it locally updates its state after communicating the update to the other nodes in the cluster. Once a node receives this information, it modifies the corresponding fields in its own MembershipList, leading to an instantaneous and consistent modification. This method allows for a seamless change without requiring a system restart. Switching from Ping-Ack to Gossip, or vice versa, also means a shift in how the heartbeat counter is updated. In Gossip mode, every node increments its heartbeat and sends the updated list to a subset of peers, which then update based on comparisons with their local values. In comparison, Ping-Ack directly pings random peers and waits for an acknowledgment within a time limit, eliminating the need for an updated heartbeat counter. Therefore, switching between modes also means opting in or out for heartbeat incrementation behavior.

The graphs below display the performance of our PingAck and Gossip (with and without suspicion) mechanisms in three scenarios: bandwidth usage, false positives counts depending on drop rates, and failure detection time. All conditions were kept constant (e.g. total time the program ran, number of VMs used, etc.) while calculating the values. Note that the data is slightly off as we use UDP to join nodes, and thus, increasing drop rates cause some nodes to start later than others.

We measured bandwidth over 60 second runs with varying number of VMs. We saw that PingAck variants had significantly less bandwidth usage, ranging between 70 (2 VMs) to 235 (10 VMs) Bytes/sec. On the other hand, Gossip variants ranged between 190 (2 VMs) to 1975

(10 VMs) Bytes/sec. Thus, PingAck is more efficient, regardless of the number of nodes in the system. In addition, we also saw that suspicion causes a slight, but not significant, increase in bandwidth usage for both mechanisms



We also tested the methods under a range of 0-30% message drop rates and recorded the number of false positives over the duration of 60 seconds. We saw that as the drop rate increased, PingAck with suspicion experienced the steepest rise, with an average of about 200 false positives after a minute with a drop rate of 30%. Gossip and Gossip with suspicion consistently performed better than PingAck and PingAck with suspicion, respectively, as they experienced significantly lower rates of false positives. Therefore, it seems like Gossip is more resilient to data loss due to the redundant information communicated.

Lastly, we looked at the failure detection times for a varying number of node failures. We saw that PingAck with suspicion detected failures slightly faster than Gossip with suspicion, and PingAck without suspicion detected failures significantly faster than Gossip without suspicion in low failure rate scenarios. The suspicion mechanism seems to have slightly increased latency, most likely due to the extra step of double-checking possible failures.

Based on our experiments, it seems that PingAck benefits the most from the suspicion mechanism, especially when looking at detection times in high-failure conditions. It detects failures faster than Gossip with suspicion. However, it is worth noting that it does perform worse when looking at the number of false positives recorded over a minute in high drop rate scenarios. Without suspicion for both methods, PingAck is faster but also experiences more false positives and data loss, making it less accurate. Comparatively, Gossip is slower but seems to be more fault-tolerant because of the redundancy.