**MP2 Report: Group 12 (Sharayu Janga - sjanga2, Aarushi Aggrwal - aggrwal3)**

**Design of the Program**

The program is designed to be able to support the 4 failure detectors: Gossip with/without suspicion, Pingack with/without suspicion. To achieve this, during initialization, a global state is set denoting the protocol mode (Gossip v/s Pingack) and suspicion mode (Suspicion v/s No Suspicion). Two goroutines run (StartProtocol and StartChecker), each invoking the handler corresponding to the current global state. The user may switch these modes at any time, without disrupting the membership list.

1. **Membership List:** All four modes access the same membership list struct. Each Member contains MachineId (IP, Port, Version), Heartbeat, Time, SuspicionState, and IncarnationNumber.
2. **Parameters:** The implementation uses Tsus = 2 sec, Tfail = 3 sec, Tclean = 6 sec, Tgossip = 200 ms, Tping = 500 ms, Tsuscheck = 500 ms, and Tfailcheck = 500 ms. These values were tuned so that at least one node detects a failure within 3 sec and all nodes reflect the failure within 6 sec.
3. **Suspicion Checker:** The suspicion checker is implemented as a state machine with 3 states: stateAlive, stateSuspicious and stateFailed. If a heartbeat/ping is not updated/responded to for Tsus time, a node will be marked as suspicious. If nothing refutes this suspicion for Tfail time, the node is marked as failed. Otherwise, it transitions back to alive. The failed state is an irrecoverable state. Once marked failed, the node is deleted after Tclean time. This modular component is invoked by both Gossip and PingAck.
4. **Fail Checker:** Counterpart of the above suspicion checker for the no suspicion mode. Here, if a heartbeat/ping is not updated/responded to for Tfail time, a node will be marked as failed.
5. **Listener:** A global listener handles 4 types of received messages: join requests, gossip, pings and acks. The latter three call a protocol specific merge membership list function since membership lists are piggybacked on pings and acks. Join requests invoke a function that contacts the introducer, adds the new node to the member list, and copies the list to the node.
6. **Gossip Protocol Design:** Everytime StartProtocol invokes the Gossip sender, each node will randomly choose a neighbor from its membership list (of unique IP addresses, excluding duplicate versions of a single node) to gossip to. The entire membership list is marshaled into the data's JSON encoding, and sent to the random neighbor. When a node receives gossip, it merges its list with the incoming list, using the following rules.
    - New members are added immediately by contacting the introducer to fetch the membership list
    - For existing members, failed states always take priority.
    - If none of the entries convey failure, the incarnation number takes priority.
    - If the incarnation number is equal, heartbeat takes priority.
    - If even the heartbeat is equal, suspicion takes priority.
    - If a node tries to rejoin, then it again contacts the introducer to access the membership list.

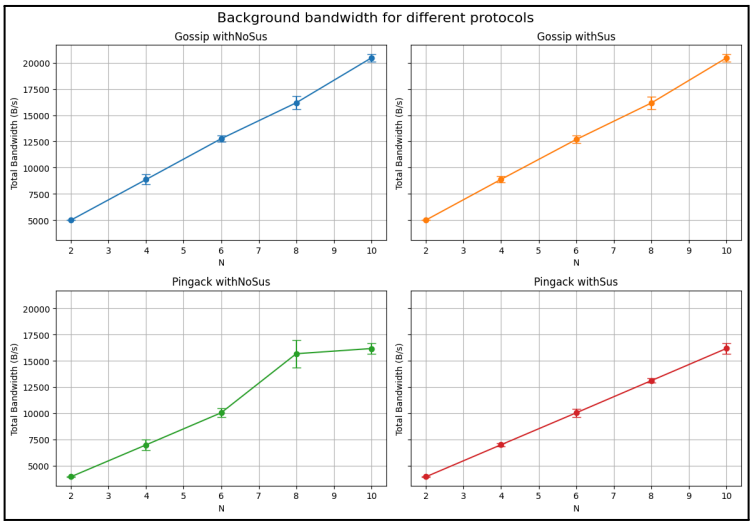
The only exception to these rules is updating self. If a node tries to mark the current node as suspicious, the incarnation number is updated and the state remains alive.

7. **PingAck Protocol Design:** Similar to above, each node will randomly choose a neighbor from its membership list to ping with the membership list piggybacked. The merging logic is very similar to the above for new members, failed states and incarnation numbers. In the case of the same incarnation number, if both members are alive, the heartbeat is updated. In all other cases, there is no update as we cannot conclude with confidence whether the incoming information is more recent or not. Again, an exception to these rules is updating self as described above.

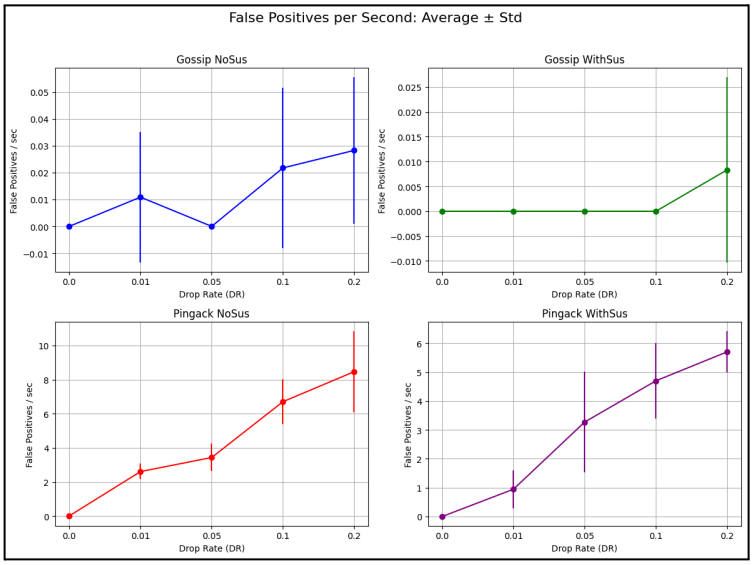
**MP2 Report: Group 12 (Sharayu Janga - sjanga2, Aarushi Aggrwal - aggrwal3)**

## Experiments and Results

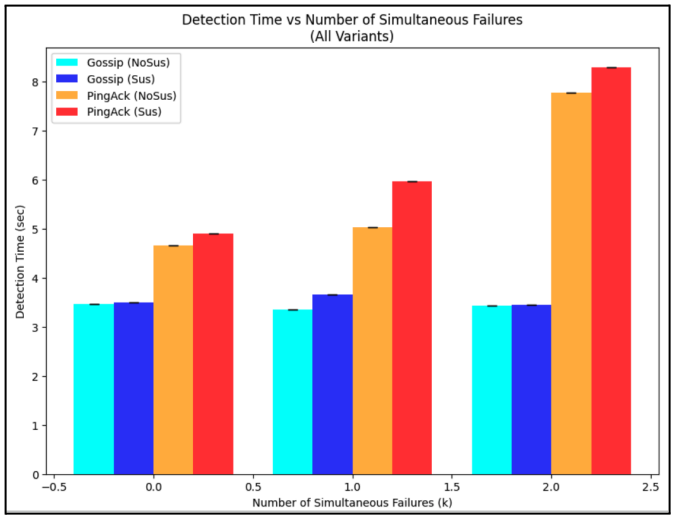
A design choice we made for Pingack is that it sends a ping and waits for an Ack for Tfail time. This wastes some time that could've been used for dissemination which led to PingAck underperforming than Gossip. Essentially, pings are only sent every Tfail time. This gives better bandwidth but worse detection time and false positives. Below are the specific results for the experiments.




1. Across the four different variants, the background bandwidth grows linearly with the number of VMs in the group. As the number of machines increases, the size of the membership list grows and sends out more data (membership_list) per gossip. Gossip and PingAck differ in the value but both have a steady growth. Pingack has smaller bandwidth as it sends less messages per sec.




2. The false positive rate per second across different drop rates for Gossip was really low (close to 0) due to our TGossip time being low. In the PingAck case, there were much more false positives per second as drop rate increased. However, false positives stay close to 0 for drop rate = 0.
3. Detection times for gossip stayed approximately the same across the different number of simultaneous failures. But for PingAck there was an increase. As the number of simultaneous failures increases, more failures must be discovered and confirmed, so the "last one found" drives up the average detection time. In contrast, Gossip spreads updates to a much higher number of peers in each round, so its detection time stays stable across different k values.




4. When comparing with and without suspicion for Gossip, our results were very similar and both variants performed well making it hard to compare. For PingAck, PingAck with suspicions had less false positives per second.
5. Gossip without suspicion performed better because PingAck was piggybacking information with each ping and ack while waiting to receive an ack at the same time.