# MP2 Report

## Overall Design

We implemented a multi-servers system where each server keeps track of all other servers/processes status in its membership list. Within each server, we implemented multiple threads to execute different tasks: the main thread for standard I/O, one for sending gossip, one for receiving gossip, one for updating the heartbeat of itself, one for periodically checking other processes' status, and one for TCP socket. Each server has a lock to prevent race conditions.

To start the system, one must start VM1 as it is the only introducer in our design. After doing so, the way other servers join the system is as follows: 1. initialize a server, including its ID , an empty membership list, and a TCP socket; 2. Get current mode from the introducer via TCP and set mode correspondingly; 3. Add itself to its own membership list and send to the introducer via UDP; 4. Initialize and start all other threads.

Within our membership list class, we keep track of each process's ID, status, heartbeat, local time. A process's ID is a tuple that is composed of (IP, PORT, Timestamp at creation). Status is also a tuple of (string, int) where the first element is either alive/suspected/failed and the second element is incarnation number, which is initialized to zero. Heartbeat is incremented periodically by the server. For local time, we use python's built-in time() module to keep track of it (note: we still treat local time as 'local').

## Gossip vs Gossip+S Design

We implemented two different failure detection methods: Gossip and Gossip+S. In our design, they differ in two fundamental aspects: how they update and manage processes' status and how they merge incoming membership lists.

For status management and update, Gossip has two status–alive and failed. A process is updated to failed status when the difference between current local time and last updated local time is greater than T_fail; and a failed process is removed after T_cleanup, which is used for preventing ghost processes. Gossip merges an incoming membership always by comparing heartbeat. If a process is alive in the server's membership list and receives a heartbeat of an incoming membership list, we update the heartbeat to the incoming one and local time to current local time. However, if a process is failed in the server's membership list, we would only merge if the incoming membership is sent from the process that is failed as failed and if it has a higher heartbeat.

On the other hand, Gossip+S has alive and suspected status. A process is marked as suspected after T_suspect and will be removed from the list after T_fail. We prevent ghost processes in the merging process, by only accepting 'alive' processes when a process is not in the server but exists in an incoming membership list. If the process is already in the server, then we check the following: if the process is the server itself and has a status of 'suspected', we increment the incarnation and update status back to alive and continue to the next member. Members with higher incarnation overwrite lower ones. If the incarnation is the same, suspected status overwrites alive status. And if all the above conditions are not met, then we check heartbeat like we did in Gossip.

When switching modes, we would remove all non-alive processes to prioritize completeness over accuracy. Specifically, after removal, if processes are still alive, they would automatically be added back through membership list merging. If not, we ensure completeness.

**Results**

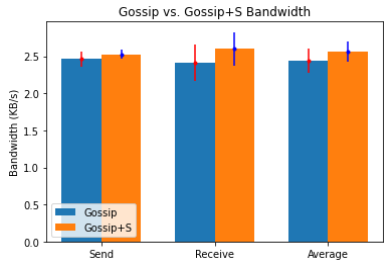


figure 1a

Gossip+S has a slightly higher bandwidth as it has a lower false positive rate. Thus, the membership lists tend to be more complete, hence larger.

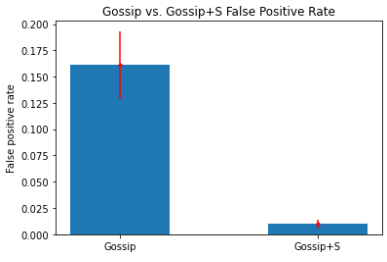


figure 1b

Gossip has a higher FP rate, as expected, whereas Gossip+S has a way lower FP rate.

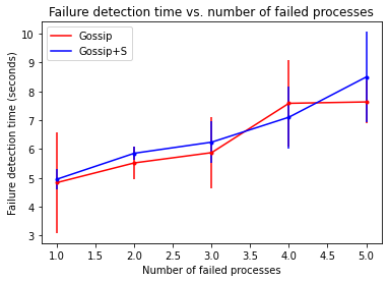


figure 1c

The increase could be a result of higher logging time or personal error, which can be indicated by the relatively large standard deviation.

Config for 1a, 1b, 1c: T_fail = 2.25, T_suspect = 2.25, T_cleanup = 2.25, fanout = 2, drop rate = 0

We set T_fail, T_cleanip and T_suspect each to 2.25 seconds, providing a 0.5 second transmission delay time, to enable that failures would be detected within the 5 seconds time-bounded completeness.

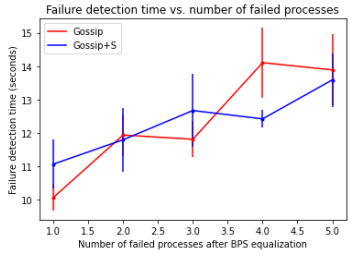


Figure 2a

The increase could be a result of higher logging time or personal error, which can be indicated by the large standard deviation and the irregular pattern.

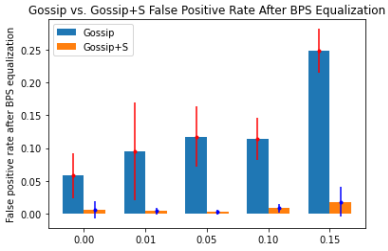


Figure 2a

The false positive rate increases as drop rate increases for both algorithms. Gossip+s still has a lower FP, which is the same as our expectation.

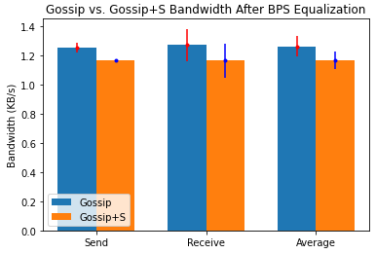


Figure 2c

We increase gossip+S's period to 1.1 second to equalize their BPS. Thus, gossip+s' frequency is now 0.91 messages per second.

Config for 1a, 1b, 1c: T_fail = 5, T_suspect = 5, T_cleanup = 5, fanout = 1, drop rate = see above

As we increase T_fail, T_suspect and T_cleanup, even with some low message drop rate, we observe a significant drop in false positives as we provide processes more time to 'redeem' themselves from failures. Another observation we made is the drastic increase in FP rate after drop rate reaches 15%. We conclude that drop rate and false positives might have an exponential relation as dropping messages would lead late updates, which can potentially induce chained false positives among servers.