CS425 MP2 Report

g57 - hanxu8 (Xu) & zerunz2 (Zhao)

1. Design:

We implemented a bi-directional ring-based SWIM-style failure detector. In order to meet the requirements (minimal number of monitors/no random pinging/not pinging anyone), **each machine in our processor pings its 2 predecessors and 1 successor every 0.3 second**. Since each member **has the equal load and the average network message load is light** (see section 3 measurements), our failure detector scales to large N. Our failure detector has **5 sub-components**:

1. **shell()**: an interactive shell that allows users to specify the desired action
2. **join()**: allows a machine to join the group
3. **send_ping()**: allows the monitor to ping to its 3 neighbors
4. **receiver()**: updates the local membership list according to the type of message received and reacts according to different types of messages
5. **monitor()**: monitors whether any machine fails by comparing current local time with the last updated time in a local dictionary

The message type could be JOIN/PING/PONG for join()/send_ping() and receive() methods. All messages are transmitted in UDP. **The marshaled message is a list with format**: [message_type, sender_hostname, sender_membership_list].
For each machine, we utilized Python's threading library to start 1 thread each for shell()/receiver()/monitor(), and 3 send_ping() threads to ping its 3 neighbors. We set our 10th machine to be the introducer, so other machines can visit the introducer and retrieve its full membership list. For each supervisor-supervisee pair, the receiver() program on the supervisee ACKs the ping message. If the supervisee leaves the group and fails to ACK the ping, the monitor() on the supervisor detects such failure by comparing the current local timestamp and a last updated timestamp in a local dictionary. On top of that, we use Python's Logging library to generate logs on each machine and integrate our MP1 (distributed grep) to debug our failure detector. The distributed grep allows us to count a certain type of message (join/ping/pong) is transmitted.

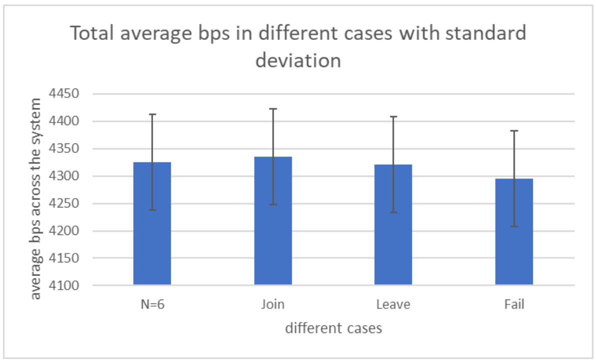
2. Completeness guarantees and violations

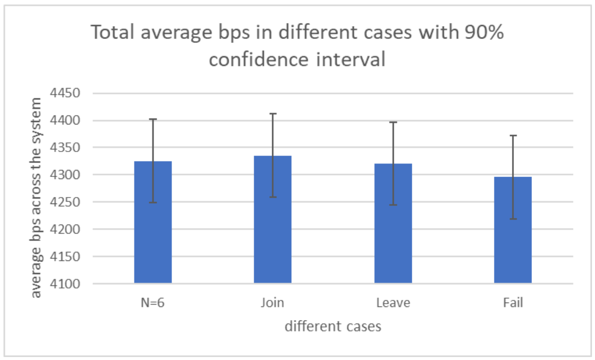
The table explains how our design meets the completeness requirements.

completeness up to 3 failures	There are 2 worst cases: a. when a machine's 3 monitors fail, the machine itself does not fail, so it is fine not to detect it; b. The machine fails and its 2

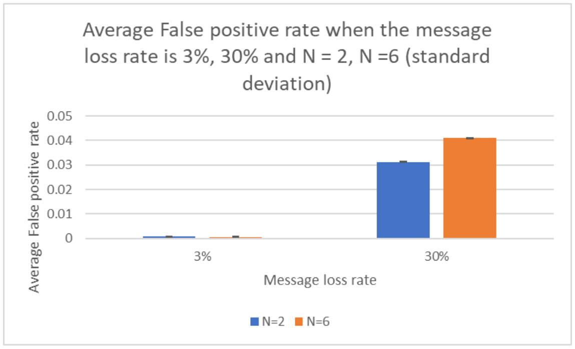
	monitors fail, but there is 1 monitor alive and the machine's failure will be detected. In both cases, completeness is satisfied.
5-second completeness	We set our failure timeout period to 2s and our send interval to 0.3s (worst total time to discover a process failure/leaving is 2.3s), which lies within 5 seconds.
completeness violation with at least one combination of 4 simultaneous failures	When a machine's 3 neighbors fail, and the machine itself fails. The machine's failure cannot be detected and thus violates the completeness.

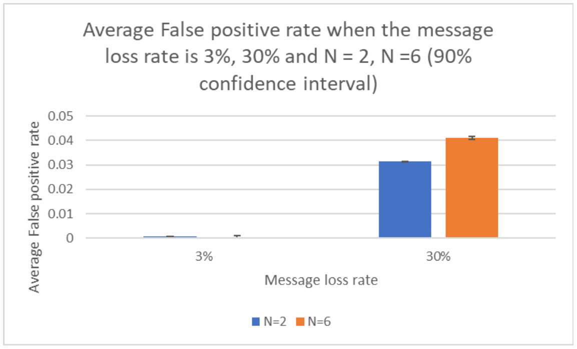
3. Measurements and discussion





We observe that the total average bps remains almost unchanged when N=6 or a node joins/leaves/fails, which is expected since our design has each machine monitor 3 neighbors. This measurement also explains that the network load is light and our failure detector scales to large N.





When the loss rate increases from 3% to 30%, the false positive rate for N=6 increases much more than the N=2 case, which is expected. This happens because of the much smaller number of successful PING-ACKs in the N=6 system.