CS425 MP2 – Distributed Group Membership

Group 04: Hsin-Yu Huang (hyhuang3), Yun-Liang Huang (ylh2)

Design and Algorithms

Overview

We implement distributed group membership in C++ based on ring topology. We use socket to enable UDP communication between all members. We use a hashmap to maintain the ring topology. Every member in the group finds 3 successors by looking up the local hashmap, and pings them periodically. The ping message is mainly used to detect failure and synchronize the ring topology.

Ring Topology / Membership list

We use a hashmap to maintain the ring topology of the group. The keys and values of the hashmap are both member IDs and defining directed edges, where keys are the source and values are the destination. Moreover, the keys of the hashmap are treated as the membership list of the group. Member IDs consist of timestamp, hostname and port number.

Every member of the group will have a local copy of the topology hashmap, and it will compare and update its hashmap after receiving others' topology hashmap.

Join / Leave Membership Group

In our design, an introducer can be any member in the group so that it knows the full topology of the membership group. To add a new member to the group, users can use our client program to command the introducer to join a host. The introducer will first update its local topology hashmap, and send the full hashmap to the new member along with a join-message. Also, since the introducer is one of the members in the group, it will send the member list, which contains the new member, to its successors in its next ping cycle.

To leave the group, a process will reset its status and drop any incoming ping until receiving a join-message from the introducer again. Other members will not be able to detect the left process, and think it may be crashed or left.

Ping-ACK Failure Detection

We use the ping-ack style failure detection. Once a process is executed, it will start receiving UDP messages. In a ping cycle, a process will send UDP messages to k successor members. After sending messages, the process will collect ACK responses from those members to check whether they are alive.

For failure detection, we set a **failure timeout T** (T=2 seconds). In each ping cycle, if a process cannot receive an ACK message from the process it pings, it will set or accumulate the **suspect failure time** of that process. **Once the suspected failure time of a process exceeds T**, the process is considered as failure. The process that detects the failure will delete the failed process in the ring topology and send the failure information to other processes. The suspect failure time of the receiving process will be reset to 0 once the ACK message from the receiving process is received by the sending process.

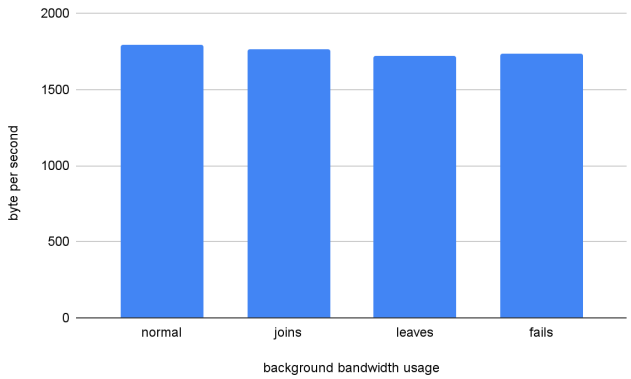
Completeness Analysis

a. **5 second completeness**: each process will use ping 3 successors to detect failure. Suppose processes are numbered from 1 to 10, and the process #2 is crashed, process #1, #9 and #10 will detect the failure if no Ping-ACK from #2 in two seconds. Process #1 will forward the updated membership list to processes #3, #4 and #5. Then, #5 will further forward the information to process #6, #7 and #8. Therefore, it takes 2 seconds to detect the failure and 3 ping cycles, which is 3 seconds, to stay complete.

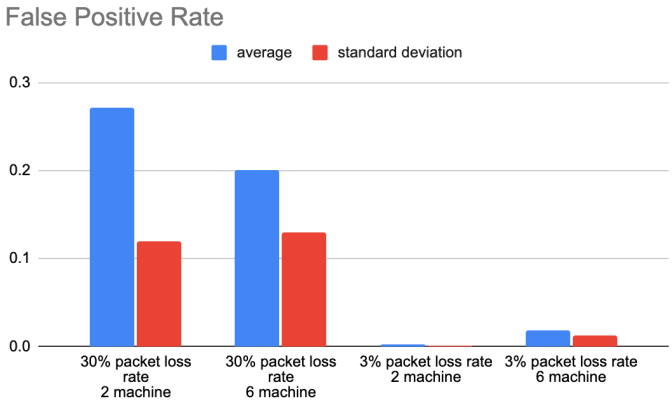
b. **Completeness up to 3 failures**: every member ping three successors, so each of them could detect the failure of the next three members. Also, every member has a full copy of the ring topology, so if one process detects a failure, it could delete the failed member from the membership list and adjust the ring topology.
c. **Completeness would be violated with at least one combination of 4 simultaneous failures**: if 4 continuous processes crashed simultaneously, the failure of the last one process X will not be detected immediately because all of its failure detectors are also failed. However, every member in the ring has the full copy of the topology and will adjust the topology. Thus, process X will become a successor of some processes and the failure will eventually be detected.

MP1 is useful for debugging MP2 since we could grep keywords from our log file easily. We use the querier to check the member list on all processes.

Measurements



Plot 1. Background Bandwidth Usage Measurement



Plot 2. False Positive Rate Measurement

Background Bandwidth Usage Measurement

In our design, the behavior of maintaining, joining, leaving the membership, and detecting failure are very similar. Members always send the full membership list (topology hashmap) to others. So we expect that there should be no significant difference between these behaviors. We also expect that failure and member leaving could have lower byte-per-second, because the number of members in the group has decreased, so the topology size is also smaller. As the Plot 1 shows, the measurement matches our expectation.

False Positive Rate

We expect the false positive rate will be very low for 3% packet loss. And we expect the false positive rate for 6 machines will be higher than 2 machines since there will be more UDP messages sent between machines when executing 6 machines.

The measurement result is illustrated in Plot 2. The result matches our expectations. We get a high false positive rate when setting 30% packet loss rate and low false positive rate when setting 3% packet loss rate. The rate difference between 2 machines and 6 machines is not obvious in the 30% packet loss rate, while the difference is quite obvious in the 3% packet loss rate. It is reasonable since there are fewer UDP messages in one cycle in 2 machines than one cycle in 6 machines, so there will be fewer failure detections.