CS425/ECE428 MP2 Report

Yen-Chieh Sung (ycsung2), Chih-Shin Wang (cswang6)

Design

For demo's convenience, we write client RPC code to call the server to join, leave, show member list, and show self id. On the server side, all the non-heartbeat messages (e.g. introduce new nodes to other nodes, failure detection handling) are in RPC with TCP. Heartbeat is sent by UDP.

- Join event
  The new server will call the already known introducer to ask for joining, then the introducer will call all the other nodes on its member list and add this node to its member list. The other nodes will then update their member list and call the new node to add them to its member list.
  Note: If the introducer ***i*** directly send the current member list back to the new node **a**, then a problem occurs in this case: when another member node **k** left before it knows **a**, and the leave message arrived ***i*** after ***i*** sends the member list (which still contains **k**) to **a**. Now, **k**'s leaving event will not be sent to **a**, and thus **a** will never know that node **k** is gone, but **a** still maintains a member list that contains **k**. In contrast, our protocol design guarantees that only alive nodes will be shown in **a**'s member list.
- Leave event
  The leaving node **b** will call all other nodes to remove **b** from their member list. **b** will also empty its member list.
- Failure detection
  The heartbeat monitor thread will periodically (once per second) access to the member list, update the timestamp of non-monitored nodes to current, and check if the monitored nodes are timeout. When timeout (a failure is detected), the heartbeat monitor thread will use RPC to call all other nodes (including the other two detectors) to remove that failed node from their member list. We set the timeout = 4 second, i.e. if it hasn't received heartbeat from its neighbor and updated the timestamp over 4 second, then mark it as failure and handle.
- Heartbeat design
  The heartbeat message, including host IP, RPC port, UDP port, and incarnation number, is marshaled into json format with Golang's package. Node i will send 3 heartbeats to node i-1, node i+1, and node i+2 (i means the index on the ring position). When node i and two of its monitor nodes fail together, there is still one node monitoring node i, which ensures up to 3 failures completeness. Every time before sending heartbeat, the sender will check the current member list and decide who to send. Thus, when join/leave/fail happen, the member list will be updated by RPCs, and the heartbeat sending thread will always get the up-to-date neighbors (targets). To make bandwidth low but still able to detect failure in 5 seconds, we set heartbeat interval to 4 seconds.
  Note: It is ok to set the heartbeat interval to about 4 seconds to reduce bandwidth, but in order to avoid false detection caused by UDP packet drop, we can use 2 seconds as the heartbeat interval, and still use 4 seconds as the timeout (so failure detection will occur only if 2 consecutive heartbeats are lost).

MP1's helpfulness

During our development, we print messages to local log file and use MP1 to grep the log in one terminal window, enabling us to easily inspect the state of all running server without switching tabs for different machines.

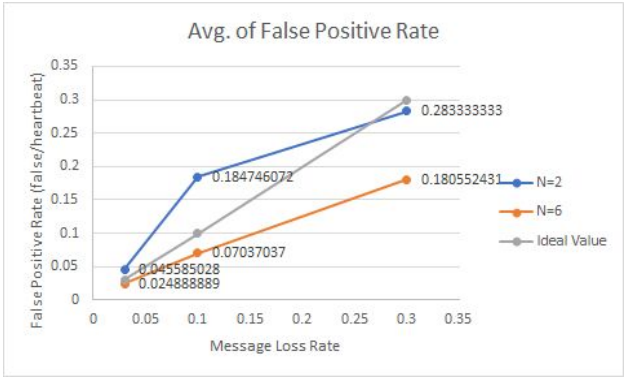
Scalability

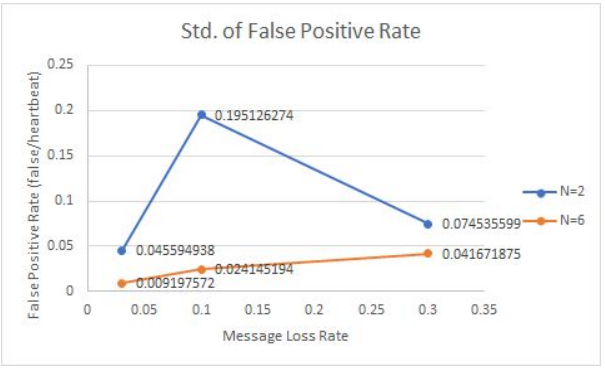
Our one to all notification is amortized to O(1) per node and our heartbeat is 3 messages per unit time. Thus, it is also O(1) per node, so the design scales to large N.
Total messages in MemberJoin: 2N - 1
Total messages in MemberLeave and FailureHandling: N - 1
So, the non-heartbeat message amortized cost per node is O(N / N) = O(1)

Measurement

(All the calculations of bandwidth below ignore headers of different internet layers)

1. Background bandwidth: We use json format for the heartbeat message, and each packet has size of “117 + incarnation digit number” bytes, approximately 120 bytes. With 1 heartbeat per 4 second, 6 machines, each node sending 3 packets per heartbeat period, the background bandwidth is 120 * 3 / 4 = 90 Bps per node, and the total bandwidth of the cluster is 90 * 6 = 540 Bps.
2. Average bandwidth when a node joins, leaves or fails:
    a. The size of a message (exclude Golang RPC header) is approximately 100 Bytes
    b. Assume the messages will be sent in 1 second
    c. Joining n node cluster: 1 message from new node to introducer + n-1 messages from introducer to others + n-1 messages from others to new node = 2n-1 messages = 200n-100 Bytes (n=6: 1100 Bytes)
    d. Leaving n node cluster: n-1 messages from leaver to others = n-1 messages = 100n-100 Bytes (n=6: 500 Bytes)
    e. Failure in n node cluster: n-1 messages from first fault detector to others = n-1 messages = 100n-100 Bytes (n=6: 500 Bytes)
    f. When n is larger, the # of messages increases, but the amortized (average) bandwidth per node remains the same. For example, there are 200n-100 Bytes being sent when a new node wants to join, but the average bandwidth per node is (200n - 100) / n = O(1).
3. false positive rate v.s. message loss rate in 3%, 10%, 30%:





Each data point consists of 5 measurement. The false positive rate is # of false positive events / number of heartbeats. If we set the period to be k seconds before heartbeat is timeout, there will be $\lfloor \frac{5}{k} \rfloor$ heartbeats. Only when all of them are lost will false positive happen. Thus we can get $false\ positive\ rate = (message\ loss\ rate)^{\lfloor \frac{5}{k} \rfloor}$. Since we set the heartbeat period to 4 seconds, the ideal false positive rate for us will be the message loss rate. The result slightly varies from the expectation. The cause of the data point larger than ideal value might be the lack of enough sample. For the data point smaller than ideal value, the cause might be underestimating the intrinsic drop rate of UDP.