**A2: Group Membership** **Lucas Palazzi, Jiahui Tang**
CPEN 533

# 1 System Design Overview

We designed our system such that the nodes keep a consistent membership lists, as well as check for failures, by sending and receiving heartbeat messages in an extended ring topology. In our ring topology, shown in Figure 1, each node sends heartbeat messages to two predecessors and two successors. This way, the system will be able to handle up to three simultaneous failures as per the requirements.

Here we want to describe our designs about how to deal with three key events in the system: a new node joins the group; a node leaves the group; a node fails. If a new node wants to join the group, it should send a JOIN message to the gateway node (a hard-coded node as the introducer to the group), then the gateway node will send the joining information to all the other nodes. When the other nodes receive the joining information, they will update their own membership list. When an existing VM node wants to leave the group, it will send a LEAVE message to all the VMs in its contact list. When the other nodes receive the leaving info, they will delete the leaving VM in their membership list. After a node joins the group, it will send a HEARTBEAT message to the VMs in its contact list every 1 second and also set a timer to compare the timestamps. When the other nodes receive a heartbeat message, they will send a ACK message back to the sender. When an ack responds to the sent heartbeat message is not received after 2 seconds, the timeout process will treat that node as failed and then delete the record of that node in the membership lists.

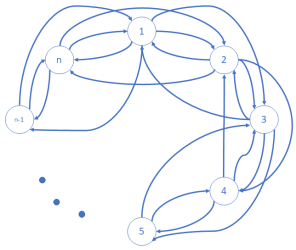


Figure 1: Ring topology used for sending heartbeat messages between nodes.

In our implementation, we used UDP socket (port number: 55313) to build communications between the VM nodes. We create four threads running at the same time on each VM: receiver process: listens, receives and responds the message sent by other nodes; heartbeat process: keeps sending heartbeat message to its neighbours; timeout process: checks if there are failures for the other nodes; commands process: listens and responds user commands like joining the group (join), leaving the group (leave) and print membership list (ml).

***Scalability:*** Our system can be easily scaled because we defined a mapping function to update the contact list for each VM node whenever the membership list is changed. Hence, when a new node wants to join the group, the membership list will be updated correspondingly. For the failure detection times, we can solve it by changing the number of neighbours of each VM nodes.

***Message Format:*** As we discussed before, we used UDP and defined 4 types of message: JOIN, LEAVE, HEARTBEAT and ACK. All the messages are transmitted as JSON payloads of UDP datagram. Those messages have four fields: TYPE, HOST, PORT, and CONTENT. For the LEAVE message, there is no CONTENT field. For JOIN and ACK messages, the contents include id of the VM; status of the VM; local timestamp of the VM node. For the HEARTBEAT messages: the contents are the senders current membership list.

***Log Querying Tool:*** We save all the logs into a local file called Server[hostname].log in each node, so we can use the log querying tool we built before to help us debugging and do the

measurements. For debugging, we can get all the error messages in the log files.

## 2 Measurements

For the bandwidth measurements, we used *iptop* on the introducer node to help use get all the results. For four machines, when the network is stable, the background bandwidth is around 10.3-10.9 Kbps.

***Background bandwidth:*** We used the same system as before with 4 nodes in the group, and when a new node joins, the node needs to send message to the gateway node, and gateway node sends the message to others, the average bandwidth usage after is around 17.1 - 20.7Kbps. When a node leaves, the node sends leave messages to others, the average bandwidth usage after is around 8.4-9.7 Kbps. When a node fails, there is no instantaneous usage change, the average bandwidth usage after is around 10.9-11.1Kbps.

***False Positive Rate:*** In our implementation, we are able to drop heartbeat message in different probabilities, but other messages cannot be dropped. For this measurements, instead, we use command: *tc qdisc add dev eth0 root netem loss 30%* to simulate the message loss. We tried 5 different rounds for each case. The confident interval is calculated based on 95% confidence level. The table below and Figure 2 show the results.

| | ***3 VMs*** | | | ***4 VMs*** | | |
|---|---|---|---|---|---|---|
| ***probabilities*** | *0.02* | *0.1* | *0.3* | *0.02* | *0.1* | *0.3* |
| ***Mean*** | 0.0586% | 0.1588% | 3.7734% | 0.0427% | 0.2008% | 4.1466% |
| ***SD*** | 0.0025% | 0.0176% | 0.0766% | 0.0020% | 0.0109% | 0.2021% |
| ***CI*** | 0.0031% | 0.0219% | 0.0952% | 0.0025% | 0.0134% | 0.2510% |

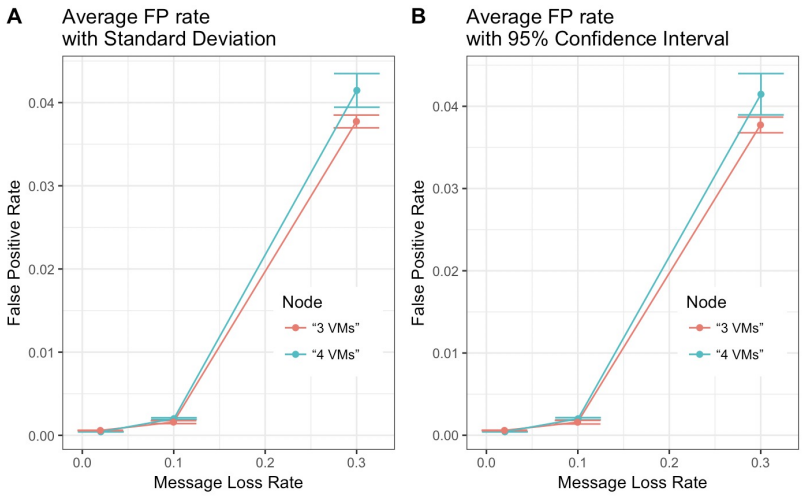


Figure 2: False Positive Rate Measurement Results.

We can easily find that a higher message loss rate yields a higher false positive rate, where the false positive rate is exponential in terms of message loss rate. We also expected a higher false positive rate with more machines, as the higher number of potential false positives increases with the number of other nodes to communicate to. According to our design, only a consecutive heartbeat message loss to a single node leads to a false positive (timeout), so for message loss rate 0.02, basically there is no false positive. As the message loss rate is set to 0.3, the false positive rate increased a lot.