Distributed Group Membership Report
CS 425 - MP2 - Group 70 (@sushmam3 and @jadonts2)

Design: We maintain a full membership list at each machine and use a ring as backbone. Apart from the introducer, each member has 4 important threads:

1. A TCP server (to wait for Leave/Crash/Join messages)
2. A UDP listener (to receive both Pings/ACKs and send ACKs when needed)
3. Main Protocol (to send Pings and wait for ACKs)
4. User (waits for commands from console input)

Our implementation is scalable, since there is no limitation on the number of nodes for failure detection.
Our messages are marshaled through the JVM and Serializable interface.

Joining: The introducer node (not a group member) is responsible for new process joins. Introducer runs a TCP server thread which listens to incoming join requests. It maintains a queue of joined machines, to check for any alive machines, popping machines no longer alive out of the queue. The introducer provides the details of a living process over TCP to the new machine. The new machine requests (TCP) the latest membership list from the living process, and adds itself. Then, the new machine disseminates its join to the rest of the members in the group, who update their membership lists. If the introducer fails, the already running group functions as normal, but without any new joins.

Leaving: When a machine is leaving the group, it removes itself from the membership list and disseminates a leave message to the rest of the members of the group, so that they can update their membership list.

Failure detection: For a protocol time of 1.5 seconds, each member pings its successor and waits for acknowledgement. If it does not receive acknowledgement in the timeout period, then it disseminates a crash message to the rest of the members to update their membership list. Since it has removed the crashed member entry, in the next iteration, it pings the successor of the crashed member. If there are 3 consecutive crashes, then in a period of 4.5 seconds, all are detected.

Thus, we meet the 5s completeness for 3 consecutive failures using just 1 monitor per node. It is then updated in all the machines quickly, since we are disseminating the crash message.

Single machine failure is detected within 1.5s if it is a good network, if there is network latency, it will be detected in the next time period. If we have more than 4 failures, it is detected after 5s.

Since join/leave/crash uses dissemination to update the membership list in the group, it takes less than a second, hence 3 consecutive failures are reflected within 6 seconds across all groups.

We can achieve completeness also by monitoring 3 successors for a period of 5s. Our code provides the flexibility to alter protocol time and number of successor/predecessor to be monitored.

Membership List: The membership list is a sorted tree-set, treated circularly. This facilitates fast insertion and deletion, while ensuring that every list has the same order at every process. This makes determining successors simple, no matter how often the list is updated.

Rejoin: Membership list stores the ip address, port and joining timestamp (used as incarnation number) of each of the members. Since we are treating crash and leave as the same, by removing the member from membership list, for rejoin it's the same process as joining a new machine.

Logging: Logs are located in `/srv/mp2_logs/` with a name of either `introducer.log` or `member.log` depending on what was run. Logs contain membership updates, as well as info on messages.

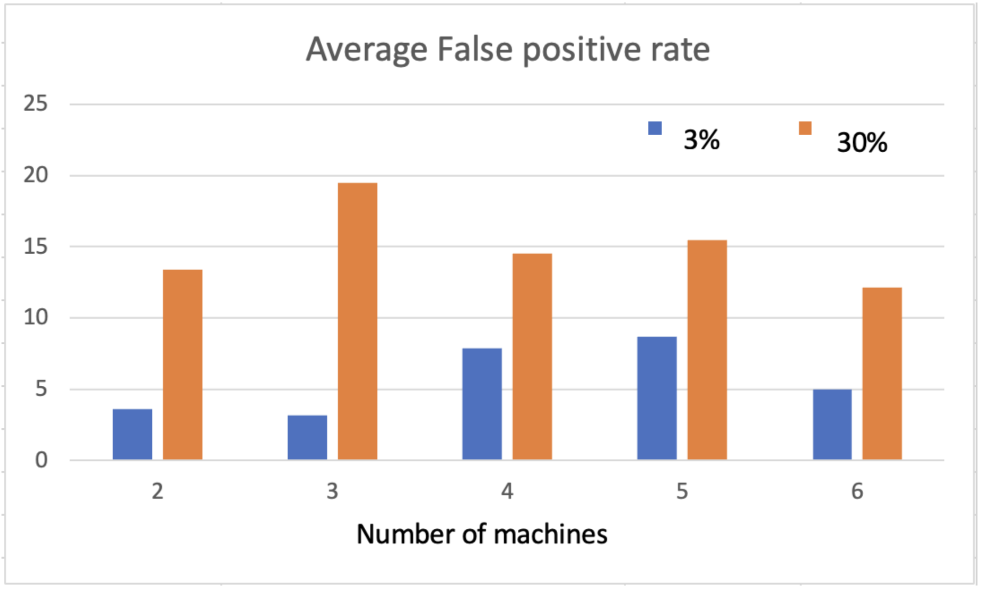
Performance Metrics:

Bandwidth usage: On average, using 6 machines, the total background usage (pings/acks) of our *failure* detection system was 3,925 Bps. *Joining* a 7th machine required 4,992 bytes. The 6th machine *leaving* required 2,590 bytes, and *crashing* the 6th machine resulted in 2,072 bytes of related messaging.

False Positives:

False positive rate was measured by counting the number of total pings and false positives in a period of 60 seconds. The metrics displayed below are normalized (%). False positive 13.41% means for every 100 ping/ack, we have 13.41 false positive. The message drop rate was simulated at the ACK sender. Confidence interval percentage is 95%.

Msg Drop Rate %	No of Machines	Average (%)	Standard Deviation	CI Upper Limit	CI Lower Limit
	2	13.41	4.78	9.59	17.2
	3	19.47	9.21	12.1	26.8
	4	14.55	4.19	11.2	17.9
	5	15.46	5.30	11.2	19.7
30	6	12.14	2.33	10.3	14
	2	3.57	1.12	2.67	4.47
	3	3.16	8.55	-3.68	10
	4	7.86	1.81	6.41	9.31
	5	8.68	4.43	5.14	12.2
3	6	5.01	5.42	0.67	9.35



As expected, with an increase in message drop % in the network, the false positive rate increases because nodes are not receiving acknowledgements on time. As the number of machines increases, there is a slight increase in the positive rate.