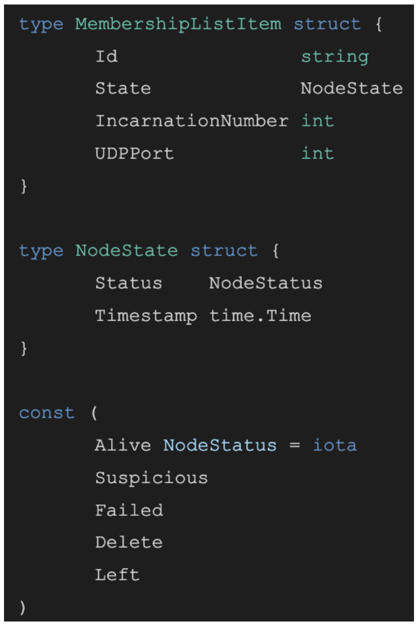
CS425 - MP2 - Distributed Group Membership

Lavanya Ramkumar (lr15)
Shahid Ikram (shahidi3)

Design:

We have implemented a SWIM style distributed group membership protocol where in each protocol period (0.5s), each machine in the topology (ring shaped) pings one of the neighbors (predecessor, successor, super-successor) in a cyclic way, making sure it monitors all the 3 neighbors over three protocol periods. This design ensures that 3 simultaneous failures are detected. This design scales well for a large number of nodes because each process just monitors 3 of its neighbors and sends a copy of the membership list to them instead of flooding the whole topology with the list which could lead to increased network congestion as in all-to-all heartbeat failure detection. We also piggy back the marshaled membership list with every “pong” that the node sends out ensuring gossip style membership list dissemination.



```
type MembershipListItem struct {
        Id                string
        State             NodeState
        IncarnationNumber int
        UDPPort           int
}

type NodeState struct {
        Status    NodeStatus
        Timestamp time.Time
}

const (
        Alive NodeStatus = iota
        Suspicious
        Failed
        Delete
        Left
)
```

Structure of the marshaled membership list → List of items of type MembershipListItem (shown in the attached code snippet).

The state of each of the entries in the membership list goes from “Active” -> “Suspicious” -> “Failed” -> “Delete”, after which the entry for that process is deleted. If the process doesn’t respond to the UDP ping, the process is marked “Suspicious” and if it remains suspicious for T_FAIL = 1s, the state is updated to “Failed”. Once the process is marked as failed, we wait for another T_DELETE = 1s to mark the process state as “Delete” after which the process will be removed from the list. This ensures that a failed process will be removed from the list in well under 3s.

The distributed log querier that we created in MP1 helped us debug through an issue where the threads were getting into a deadlock. We logged statements where lock was being acquired and released by threads to narrow down to the procedure call in which the acquired lock wasn’t being released to get past the issue. We also used it to compute the bandwidth of the pong sent per second by logging the number of pong messages sent over a time period.

Implementational Details

We run a stabilization protocol on top of a ping-pong based SWIM detector to stabilize the topology of the ring and update the states of the processes in the membership table. This protocol periodically (1s) scans the membership lists and does the following:

- If a process has been marked Suspicious for >T_FAIL, it updates the state to Failed.
- If a process has been marked Failed for >T_DELETE, it updates the state to Delete.
- If a process has been marked as Left (left the network) for > T_LEAVE (2s), it updates the state to Failed.
- If a process has been marked as Delete, it will be removed from the list

In parallel to this protocol, a UDP server and client is set up at each node that performs:

- Ping - Pong messaging to monitor neighbors, with the membership list piggybacked in the pong message.
- Merging of the Membership list received from the neighbor.

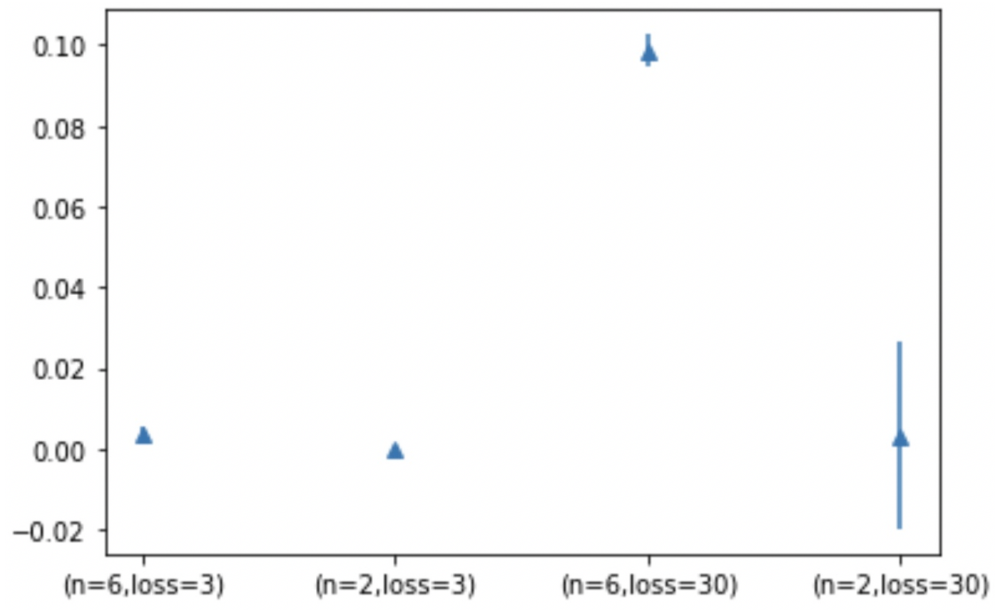
Analysis:

For 6 machines in a ring topology with no membership changes we observed a total bandwidth usage of 1834 bytes/second. This seems like a fair number because the size of the membership list at every process was found to be around 600 bytes and having 3 connections with the neighbors justifies the bandwidth usage.

Average bandwidth usage with 6 machines over 5 trials per operation:

- Joins	= 1569.2 bytes/second
- Leaves	= 1483.8 bytes/second
- Failures	= 1684.2 bytes/second

The average bandwidth would be around the same value for all the operations because in each case we disseminate the whole membership list.



Experimenting by dropping UDP packets on the server to simulate packet loss, we observed that with the increase in loss, we saw an increase in the false positives which is expected behavior. We also see from the above graph that when the number of nodes in the system is less, we have low false positives. This is also expected, because when there are a large number of nodes, the rate of dissemination of a false positive message is higher and hence greater chance of marking the process as failed.