1. Design
   1.1. Algorithm

   This group membership service is based on **Gossip**-Style Protocol. On each node, a **gossiper** is in charge of the maintenance of the local **member list**, where each entry includes the node **ID**, the newest **heartbeat** number heard, the local **timestamp** of that heartbeat, and the **status** from the local view. Periodically, the gossiper randomly chooses a member and **gossips** to it membership rumor in format as shown in 1.3. Specifically, the gossiper will refresh the member list first according to the configured time threshold for suspecting a joined node, failing a suspected node, and forgetting a left node. If there is still any joined member, the gossiper will heartbeat once and send it the rumor, which contains valuable information of the joined or left nodes. Meanwhile, the gossiper **listens** (on port 2334) to others and updates its member list using the heard rumors. Specifically, if some node is said to be left, the local entry (if there is any) will be updated accordingly; if some node is said to be joined, a new entry will be created if there isn't an existing one, otherwise the existing entry will be updated if the heard heartbeat is newer (larger). In all of the situations above, the timestamp is local time. Via those two long-running threads, the membership changes can be captured globally. When a node **joins**, it adds a well-known introducer into its member list in order to report its joining. Since there's only one entry, the gossiper will keep gossiping to the introducer until the introducer hears it and eventually it's in the group. When a node wants to **leave**, and it knows any joined node, it will mark itself as left and gossip to a joined node. When a node **fails**, its status in others' member lists will transit from joined to suspected, and finally to failed.

   1.2. Scalability

   This implementation has high scalability basically because of the epidemic-style propagation. Each node participates in rumor generating and spreading, with a similar load. There is no hotpot when the membership is stable. However, if there is a surging pattern of node joins, more introducers should be created to share the incoming UDP's.

   1.3. Message Format

   | id | heartbeat | status |
   |---|---|---|

   The rumor message is a list of information about joined and left nodes. Each element contains the node ID, the heartbeat counter, and the status of this node. The list is dumped to JSON format before sending.

2. MP1

   The distributed log querier is helpful when debugging distributed programs, because all the logs across multiple hosts can be queried from a single host. So we can check and compare the member lists' changes by one query on a single host.

3. Measurements
   3.1. No Membership Changes
      3.1.1. Expected Bandwidth Usage

      The size of a rumor is around 37 (metadata) + 66 (per entry) * 3 = **235 Bytes**. (The entry of destination node is excluded from the rumor because no one knows better than itself.) The gossip frequency is 5/s. So the bandwidth usage of each node is around **9.6 Kbps**, and totally it is around **38.4 Kbps**.

3.1.2. The Experiment Result

**36.0 Kbps** in total. The difference with the expected one may be due to the UDP loss.

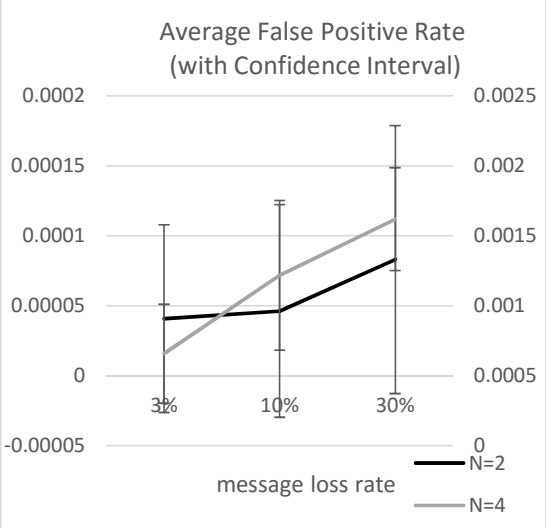
3.2. Membership Changes

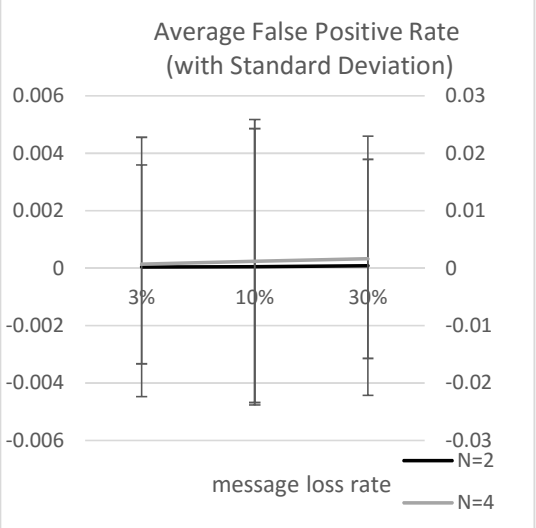
| | **Intermediate Usage** | **Stable Difference** |
|---|---|---|
| **Join** | O(1), one rumor with one entry | + O(N), two regular rumors per gossip interval |
| **Leave** | O(1), one entry | - O(N), two regular rumors per gossip interval |
| **Fail** | 0 | - O(N), two regular rumors per gossip interval |

The concrete usage varies by the number of nodes.

3.3. False Positive Rate

| **Readings** | | **Message loss rate** | | |
|---|---|---|---|---|
| | | 3% | 10% | 30% |
| **N** | 2 | Total: 6132*2<br>1/2: 1+0 | Total: 5314*2<br>1/2: 1+0 | Total: 3516*2<br>1/2: 1+0 |
| | 4 | Total: 1616*4<br>1/4: 1+2+0+0<br>2/4: 0+0+3+4<br>3/4: 0+0+0+0 | Total: 1438*4<br>1/4: 0+0+0+0<br>2/4: 1+3+8+2<br>3/4: 0+0+0+0 | Total: 1506*4<br>1/4: 1+0+0+0<br>2/4: 0+6+5+8<br>3/4: 0+0+0+0 |




Discussion: As the message loss rate increases, the false positive rate also increases. This trend makes sense, since some rumors (with new heartbeats) are lost because of the bad network, but the gossiper will consider the staleness of its entry is caused by node failure instead of network issue and mistakenly mark the joined node as failed. With higher message loss rate, more message will be dropped by the network, leading to higher possibility for gossiper not to receive rumor. This will increase the possibility of mistaken marking, aka, increase the false positive rate. Thus the false positive rate should grow as the loss rate increases.