# CS425: Distributed Systems – MP1 Report

**Gopalakrishna Holla V (hollava2)**
**Alok Tiagi (tiagi2)**

## Design and algorithm

### Membership List Design

The membership list is a table of membership detail entries that each corresponds to one machine in the network. Each entry has 6 fields: -

1. ID – The network ID of the machine. This is a concatenation of the IP of the machine with the timestamp of its inception (according to its own clock).
2. Heartbeat – The heartbeat counter of the machine. A machine can only increment its own heartbeat.
3. Local timestamp – Time at which the last update to heartbeat was made.
4. Failed – If this is set, it means the machine is suspected of failure.
5. Leaving – A machine sets its own leaving bit when it desires to leave the system.
6. Failed timestamp – Time at which this machine was marked as failed.

Lists are received from peers and merged based on the ID. If an ID is unseen, it is added to the table. This results in a 'join'. When an entry matches an already existing entry at the local table, the entry is updated if the heartbeat value received is higher than the one in the local entry. This is a normal gossip. When a machine wants to leave, it sets the leaving bit in its entry and propagates the entry in the normal gossip fashion.

The processing of the membership list at a machine is orthogonal to the receipt of gossips. At the end of every interval in the main context, the machine checks its list to see if any entry has not been updated for $T_{fail}$ seconds and marks it failed. If a machine has been marked as leaving or failed, it is deleted if it has been $T_{cleanup}$ seconds since that event was recorded.

### The main context

The daemon, when invoked, spawns a thread. The spawned thread listens on a port to receive gossiped lists. Once a gossip list is received, another thread is spawned to handle the updating of the local list. All accesses to the local membership list are mutually exclusive.

In the main context, at the end of every interval, a list of machines designated to be receivers is obtained from the local membership list. The local membership list is then processed for failures, leaves and cleanup, the heartbeat of the local machine is updated and, if required, retirement is requested (leaving bit is set). The processed list is then sent to the designated receivers (a fraction of the members in the list).

When a non-master daemon initially starts up, it simply puts its own entry in its local membership list and sends it to the master. The master adds the entry to his list and gossips it out. Eventually, the new member will receive the membership list either from the master or another member.

**Timeliness constraint:** With proper tuning of $T_{fail}$ (< 5s), every machine can be sure that an entry in its membership list that has failed to update its heartbeat will be marked as failed within $T_{fail}$ seconds. Since the processing of the list is orthogonal, it is certain that a failed machine will be marked as such in $T_{fail}$ seconds because it will fail to update its heartbeat. This may, however, result in false positives due to loss of messages or network latencies.

Similarly, a join will be gossiped to everyone from the master. Proper tuning of the interval of gossip and fraction of machines to gossip to, will result in the join being propagated within 5s across the system (the fixed fraction ensures that an increase in the number of machines will not cause increased delays to gossip propagation).

**Marshalling:** We use serialization to text (by means of boost serialization text archives) to ensure architecture and "endian" independent message passing. All it requires is a gcc compiler and matching boost versions.

**Use of MP1:** Used for tracking joins, leaves and failures. It was particularly helpful to tune the various parameters involved and do the right tradeoff between rate of false positives and bandwidth usage.

**Experiment/tool used to monitor bandwidth usage:** We used the 'nettop' command on OS X to monitor the port level traffic at one machine in a system of four machines. We monitored the data received every minute. During joins and leaves, for our design, there is no extra change in bandwidth usage other than the increase/decrease in usage due to change in the size of membership lists. We calculated this by incrementally adding machines and monitoring the usage at every increment.

Average bandwidth usage measured at 1 machine in a system of 4 machines without joins/leaves – **512 bytes/s (30 KB/min)** – this is the rate of bytes sent at one machine.

The average bandwidth usage for 3 and 2 machines is 239 bytes/sec and 171 bytes/sec respectively. The expected bandwidth usage change because of a join depends on whether adding a machine results in each machine gossiping its list to an extra peer. We have set the fraction of machines to gossip to at 0.5. So in the case of 2 and 3 machines, each machine gossips to only 1 peer. But in the case of 4 machines, each machine gossips to 2. This is the reason for the smaller increase in bandwidth between 2 and 3 machines and the huge jump when the number of machines is 4.

**Configuration used:**

- Gossip Frequency – 0.9 seconds
- Fraction of machines that are selected to receive gossip – 0.5
- $T_{fail}$ = 4 s
- $T_{cleanup}$ = 4s

# Extra Credit Report

Raw data

- Legend: Avg – Average , S.D – Standard deviation, M.E – Margin of error
- Confidence level – 95% Confidence interval – Avg +/- M.E
- **The entries measure false positives per second at ALL machines. Not just at one machine.**
- Each entry is false positives/second (originally measured per minute).

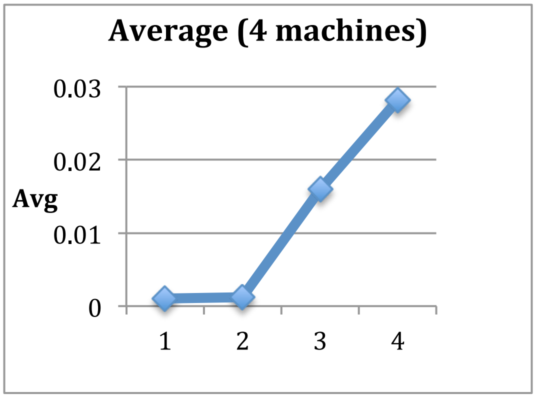
| | 1% | | | 5% | | | 15% | | | 50% | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Avg** | **S.D** | **M.E** | **Avg** | **S.D** | **M.E** | **Avg** | **S.D** | **M.E** | **Avg** | **S.D** | **M.E** |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.0236 | 0.0132 | 0.0116 |
| **3** | 0 | 0 | 0 | 0.083 | 0.028 | 0.024 | 0.125 | 0.006 | 0.005 | 0.246 | 0.019 | 0.017 |
| **4** | 0.01 | 0.02 | 0.02 | 0.013 | 0.019 | 0.017 | 0.146 | 0.032 | 0.028 | 0.253 | 0.111 | 0.098 |

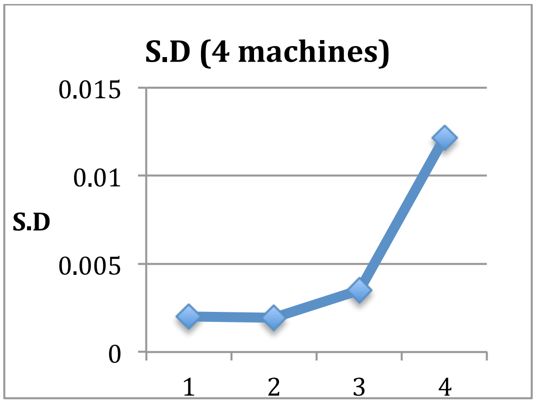
To calculate false positive rate, we need to calculate false positives/(false positives + true negatives). This is equal to false positives/No. of gossip messages in the system. Our rate of gossiping is once every 0.9 seconds. Fraction of machines being gossiped to is 0.5 (1 peer in 2 machine system, 1 peer in 3 machine system, 2 peers in 4 machine system)

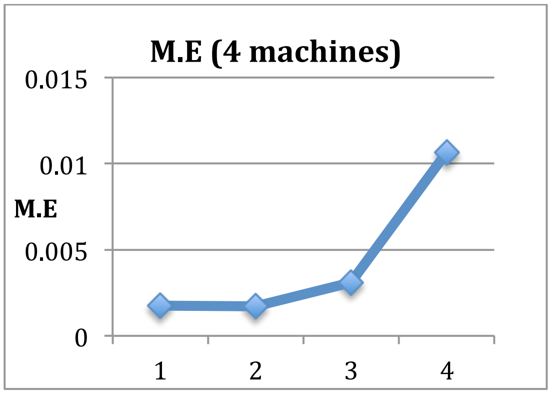
- No. of gossip messages in a 2 machine system/min = (60/0.9) * 2 = 133.3
- No. of gossip messages in a 3 machine system /min = (60/0.9) * 3 = 200
- No. of gossip messages in a 4 machine system /min = (2*(60/0.9)) * 4 = 533.3
- Each entry in this table is the false positive rate (false positives/No. of gossip messages in the system).

| | 1% | | | 5% | | | 15% | | | 50% | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Avg** | **S.D** | **M.E** | **Avg** | **S.D** | **M.E** | **Avg** | **S.D** | **M.E** | **Avg** | **S.D** | **M.E** |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.01 | 0.006 | 0.005 |
| **3** | 0 | 0 | 0 | 0.025 | 0.008 | 0.007 | 0.038 | 0.002 | 0.002 | 0.074 | 0.004 | 0.004 |
| **4** | 0.001 | 0.002 | 0.002 | 0.001 | 0.002 | 0.001 | 0.016 | 0.003 | 0.003 | 0.028 | 0.012 | 0.01 |

Comparison of false positive rates for different rates of packet loss (4 machines)
Legend for X axis: 1- 1%, 2 – 5%, 3 – 15%, 4 – 50%

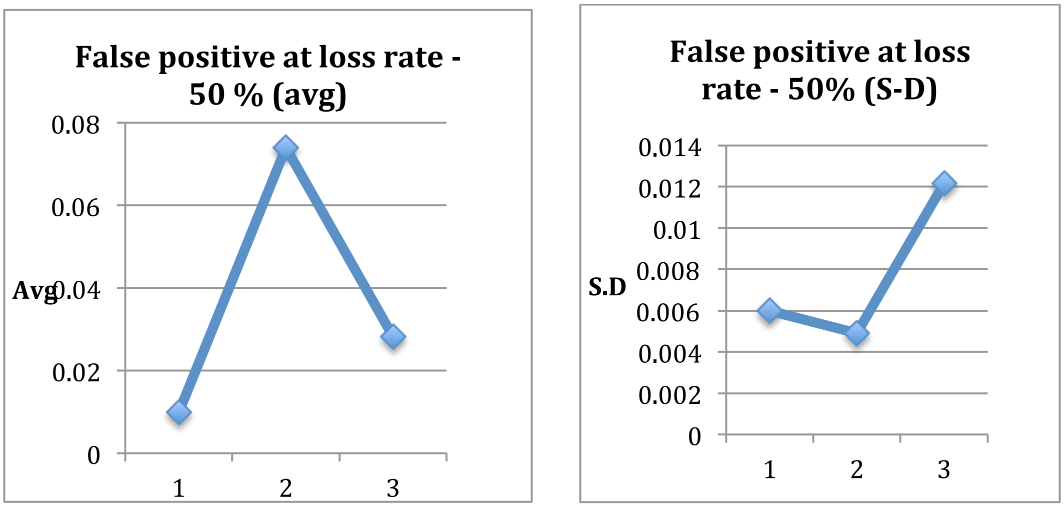






It is clear from the graph that the average rate of false positives increases with increased packet loss. This is to be expected since heartbeats will be updated less regularly. The sudden shift after 5% probably represents the threshold where more heartbeats started to go past $T_{fail}$ marginally. After that threshold, the graph is linear. The standard deviation on the other hand seems to follow a parabolic curve (i.e. jumps higher for higher values of packet loss). This is because of the unpredictability of packet loss and the fact that sometimes, if the random selection of gossip peers follows a fortunate pattern, packet losses can be made up for because of the nature of the protocol itself (each machine not only propagates its own information but information it has received from the others as well). The margin of error also shows similar behavior owing to the fact that number of false positives across readings fluctuates a lot as the packet loss rate increases.

<u>Comparison of false positive rates across different number of machines – (% packet loss)</u>
Legend for X axis: 1- 2 machines, 2 – 3 machines, 3 – 4 machines



The comparison of averages across number of machines is particularly interesting. There seems to be a staggering decrease in false positive rate when increasing to 4 machines from 3 as compared to an increase when scaling to 3 machines from 2. This is because of our tuning parameters. We send the gossip messages to half the peers. When the number of machines is 2, each machine sends to 1 peer. Even when the number of machines is 3, we still send messages to only 1 peer. This results in packet loss having a much stronger impact on false positives in the case of 3 machines. As soon as we increase the number of machines up to 4, each machine starts gossiping to 2 other machines and this greatly reduces the impact of packet loss.

On the other hand, standard deviation follows the expected pattern. As discussed in the last scenario, owning to the randomness of packet loss and the possible fortunate selection of peers by each machine, false positives for a higher number of machines can vary wildly. The dip in standard deviation when the number of machines is 3 is probably due to the fact that the false positive rate is consistently high because of the strong impact of packet loss.

While the graphs confirm the logical conclusion that the rate of false positives increases with increased packet loss, they also show the huge role that configuration plays during the scaling up of the system.