1. Design:

Algorithm: In this distributed group membership service we treat all the machines as in a ring structure and each communicates with some of its neighbours in the ring. Since we are told there will be no more than three simultaneous failure, we let each machine communicates with at least four of its neighbours, specifically two predecessors and two successors on the ring structure.

When the service is working, there are four threads running on each machine. A **receiver process**, listening to a specific port to receive UDP messages of four types: (i) *ping*, which is a heartbeat like message, (ii) *ack*, which is an acknowledge message upon receiving the PING, (iii) *join*, joining request of a node, (iv). *leave*, leaving request of a node and update the node's member list according to the received messages. A **sender process**, continuously sends UDP PING messages to neighbouring nodes for every 1 second. A **checker process**, which keeps checking the member list table and looking for any failure nodes int it. Finally we have a **monitor process** listening to user commands of (i) join: join current machine to the group service, (ii) leave: safely leave the node from the group, (iii) ml: print the current member list table, (iv) id: print current server id.

We treat the machine 01 as the introducer. For every other nodes to join the group, it must send a joining request to the introducer. When the introducer received the joining request it will multicast the joining information to all the other nodes in the group. Once a node joins the group, its sender will start to ping its neighbours every 1 second and record the local timestamp in a timer. The timer is used by checker to check if one of the node in the member list is failed, which is detected when an ACK respond to the sent PING message is not received by receiver after 2 seconds. If that happens, the checker will mark that node as failed and sender will send the updated member list to its neighbours through pinging.

By using our protocol, for a ten-node network, one message from a node can be sent to all the nodes in the network within 3 seconds (three pings). And it takes at most 3 seconds (2 seconds for timeout and 1 more second wait in the worst cast to send the message) for a node to detect the failure of a neighbouring node and send such message to other neighbours. Therefore, the machine failure will be detected within 6 seconds and a join or leave can be detected within 4 seconds.

Scalability: This design can be easily scaled up to large N. By adding the numbers of neighbours a node connect to can also fix the maximum failure detection times.

Message Format: Each message has a metadata contains a message type (ping, ack, join, or leave), a sender host name and the port. For *ack*, *join*, and *leave* the main content of the message includes (i) the id of the sender, (ii) the status of the sender and (iii) a sender's local timestamp. For the *ping* message, the main content is the sender's current member list table.

2. Use of MP1:

We use logging to print debug logs into file and use MP1 to grep useful information from those log files to debug as well as to perform some of the measurements mention below.

3. Measurements:

i. Background Bandwidth Usages for 4 Machines:

The measured total background bandwidth usages for a 4-machine network is about **13.4**($\pm$ **1.3**) **Kbps**. Theoretically a machine have to ping the other three machines and also receive pings from them. It also need to send ack messages to respond the ping as well as receiving ack messages. So the total bandwidth usage for a single machine is 6*ping(approx. 457 bytes) + 6*ack(approx. 145 bytes) = 3.612 Kbps and the total bandwidth usage for 4 machines is **14.4Kbps**, which consistent with the measurement.

ii. Average Bandwidth Usages:

The following measurements are based on the 4-machine network with the background bandwidth usage discussed above.

Join: As a fifth node joins the network, instantaneous usage will increased by the join messages to introducer and from introducer to all the nodes. The average bandwidth usage after that is: **23.2**($\pm$**3.0**)**Kbps**

Leave: As a node leaves, instantaneous usage will be increased by the leave messages between node and introducer. The average bandwidth usage after that is: **7.24**($\pm$**0.38**)**Kbps**

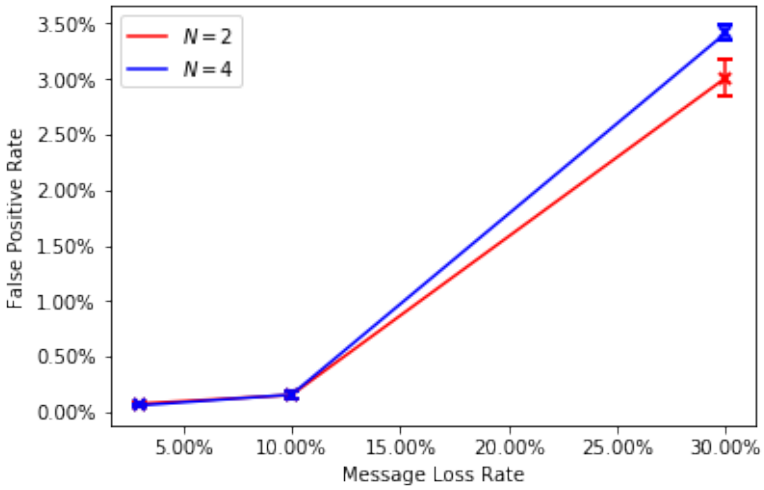
Fail: No instantaneous usage change. average bandwidth usage after that is: **9.16**($\pm$**0.68**)**Kbps**

iii. False Positive Rates:

We simulate the message lost by randomly dropping ping messages and measure the false positive rate by running the system and collect the number of times a machine is marked as failed faulty.

The confident interval is calculated based on a 90% confidence level following the equation: $CI = z * \sigma/\sqrt{n}$, where n is the sample size ($n = 5$ in our measurements) and z is the z-value ($z = 1.645$ for 90% confidence level).

	N=2			N=4		
Message Loss Rate	3%	10%	30%	3%	10%	30%
Average	0.0748%	0.1527%	3.0098%	0.0594%	0.1559%	3.4136%
Standard Deviation	0.0113%	0.0361%	0.1659%	0.0081%	0.0329%	0.0714%
Confident Interval	0.0093%	0.0297%	0.1365%	0.0067%	0.0271%	0.0588%



Discussion: According to our design, a node will only be marked as failure after no receiving acks for 2s. In our simulation this corresponds to a consecutive ping message drop to a single node. Therefore, as the message loss rate increase the false positive rate increase dramatically, which is consistent with the measurements. For small message loss rate (3% and 10%) the false positive rates for both $N = 4$ and $N = 2$ are similar. However for large message loss rate the false positive rate of $N = 4$ becomes larger than $N = 2$. This is due to the nature of our design. In order to quickly spread the information when a machine is detected as failed, a machine will send such failure detection to all its neighbours. Although such false detection will be overwritten immediately after receiving pings from target machine, it will still appears in the logs and being counted as false positive in our measurement.