# MP2: Distributed Group Membership

Group 28 - Abhishek Verma (averma11), Dominic Le (ddle2)

The MP follows the design of SWIM failure detector to keep membership lists up to date. Each machine has a full membership list. Each group contains different nodes connected in a ring fashion where each member watches the next two members in the ring. This was a design implementation we came up with since it was the minimum number of machines to take care of while maintaining completeness. The protocol was implemented in Go Language to utilize its many features built-in specially for server programming. The protocol involves running a membership script in the background (using a tmux or any other way) on each system, and includes a user prompt where any user at any point look at 1) the membership list, 2) his own IP, 3) Join the group by pinging the introducer, 4) leave the group voluntarily.

**Completeness**

The implemented protocol is complete under the given assumptions that the minimum number of members in the group is 5 and that there will only be 2 simultaneous failures at a time (as stated in the MP2 documentation). If we consider the worst situation where two machines die simultaneously, each member of the group always have the opportunity to reorganize their membership lists and and their syn/ack structure before the next failure. Meaning, each machine is always monitored by at least one other machine given the above assumptions.

**Design**

Each machine maintains a sorted, full membership list which is used to monitor the next two machines in the system ring by using SYN->ACK->SYN messages. The protocol uses an introducer to accept and handle pings from newly joining members. Every process stores the IP address of the introducer as a constant so that at startup it knows what VM to ping when joining the group. Every process initiates 2 servers at startup: one for accepting incoming messages and one for accepting an updated membership sent by the introducer when a new member joins. Membership lists are sent over port 10001 and messages are propagated over port 10000. UDP connections are used to send messages and data to and from machines within the group. GoLangs Gob encoding was used to handle formatting of sent information. Machines in the group are stored in the membership list as member structs containing the machines IP address and the timestamp for when it was added to the system. The membership list is always sorted on the basis of the IP address of the system and each system takes care of the (n+1)%N and (n+2)%N members on its list, where N is the size of the membership list. We use a message structure which contains three fields: (1) Host which sent the message, (2) Message info/status, and (3) Timestamp for when the time the message was sent. The possible messages are explained below.

*Joining :* A node can join by sending a message to the introducer with a status of "joining". The introducer then updates its own membership list, and propagates this updated list to everybody on its membership list. This way, each machine updates its membership list and can appropriately change the members it is monitoring.
*Leaving:* A node can voluntarily leave the system by a user input on the command line. Before leaving the system, it sends a message with a status "Adios" to the two nodes who were watching over this node. The two nodes update their membership lists on receiving the message and propagate a message around the ring with the same "leaving" status. A machine m would propagate the message further to its (n+1)th and (n+2)th machines until it finds out that it has already updated its membership list at which point the message propagation would die and each machine would have updated their membership list.
*Crashing:* The protocol takes care of the nodes crashing by sending a "SYN" message to its (n+1) and (n+2)th successors on its membership list. Each machine maintains 2 timers: one for its n+1 successor and one for its n+2 successor. Timers are set to 2.5 seconds, meaning a successor has 2.5 seconds to respond with an ACK before it is marked as a failure. "SYN" messages are sent to a machine's 2 successors every second, expecting "ACK"s from each successor it pings. Every time an ACK is received, the corresponding timer is reset to 2.5 seconds. If an ACK is not received for a successor and the timer reaches 0, the monitoring machine would then send the message with a status "Failed" that a particular machine m failed to its n+1 and n+2th successors. These would in turn propagate the message. Every time a propagated message is received, timestamps for the message and its membership list are compared to determine if updates should be made and if the message should continue to be propagated.
*Introducer Rejoining:* the protocol maintains file with the current membership list in the introducer. This is update whenever the introducer detects a change in its list (leave/crash/join) and the file is overwritten. This is to help the introducer come back up if it crashes. The introducer when starting checks if such a file exists, if yes, it makes its membership list from the file, then it pings everyone on the list to see if any member of the ring died while it was down. The introducer then sends out the membership list to all other members so that everyone can update their n+1 and n+2th members while keeping the lists uniform.

This protocol is scalable because each machine only takes care of its n+1 and n+2th successors, reducing network bandwidth required. The protocol can be made even more scalable by using gossip style messaging to spread leave/fail messages and by using random messaging (as described in SWIM) to keep track of its n+1 and n+2th successors while keeping in mind packet losses. The protocol also maintains extensive logs using GoLAngs logger on each machine. The distributed GREP built in the previous weeks was helpful to debug the protocol as we could look for joins/leaves and any errors in a system from the introducer itself.

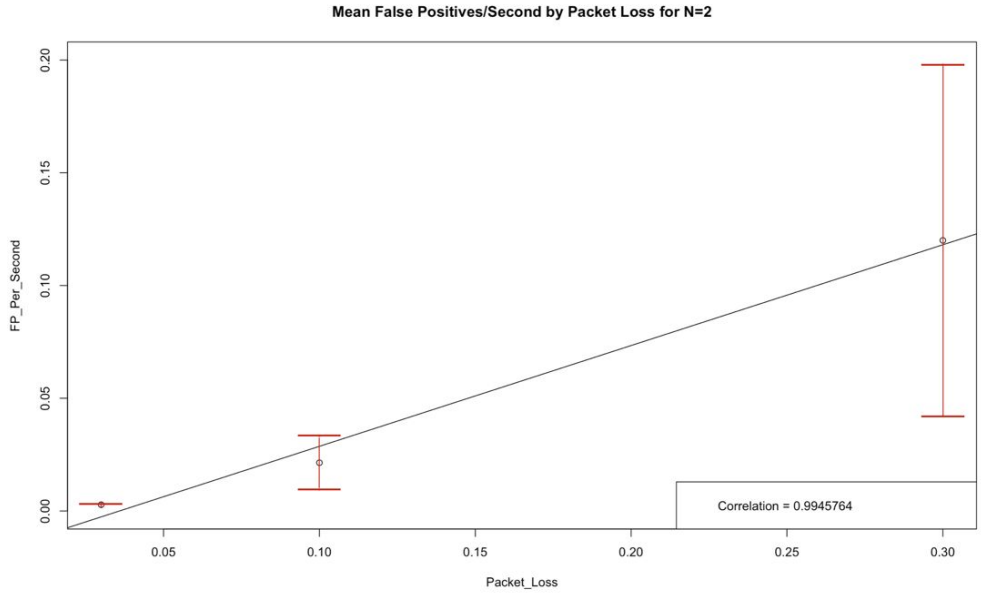
1) Background bandwidth usage for 4 machines : Sends 143 B/s (avg) and receives 143B/s

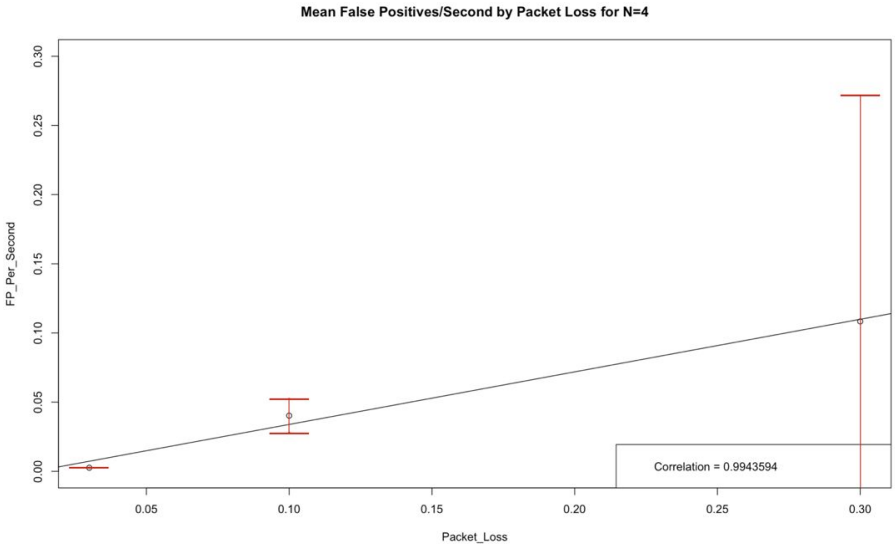
2) Expected bandwidth usage when a node joins : 403 B/s , leaves 216B/s or fails 217 B/s
3) False positive rate when

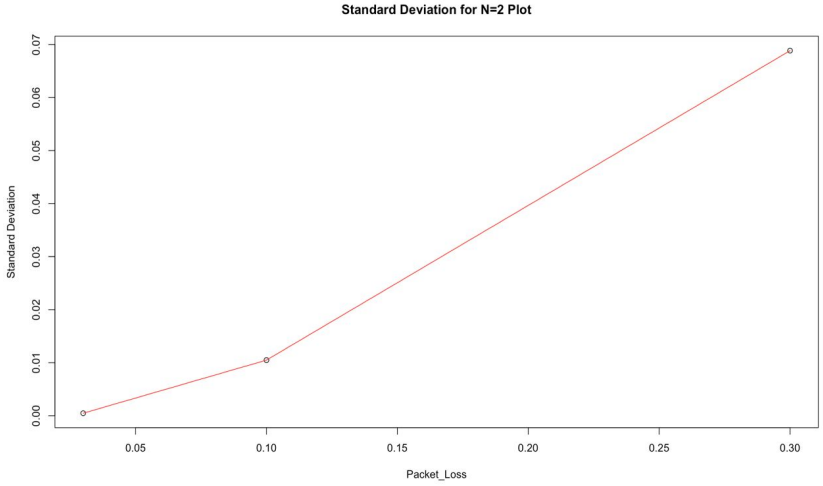
| Loss Rate/Num Machines | N=2 | N=4 |
|---|---|---|
| 3% | .00279 | .0026 |
| 10% | .0214 | .04023 |
| 30% | .12 | .10833 |

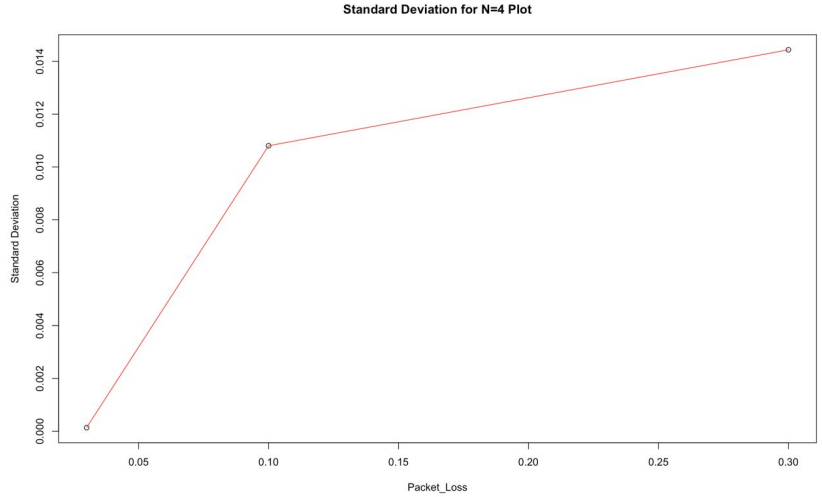
Plots for the mean false positive rate and the corresponding standard deviation are provided below for groups of size N = 2 and N = 4. Confidence intervals of 95% are also provided on the mean plot. To obtain the the data, 6 readings were taken for each N = 2,4 at 3%, 10%, and 30% packet loss. For each test, the corresponding number of members were connected and the time it took for a member to be marked falsely as failed was noted. For N = 4, VM's were subsequently reconnected to obtain more data points. For N = 2, the group had to be restarted between tests as our implementation did not allow for rejoining in such a scenario. Packet loss is simulated by using a random number generator to generator a pseudo random number between 0-100. Before sending a message, a new random number is generated. If it fallswithin the specified range (e.g. if the number was < 3 for 3% packet loss) then the message is dropped.

In the below plots for mean, correlation between packet loss and false positives per second is also shown. In both cases, it is clear that there is a strong, linear relation between packet loss and rate at which false positives are detected.




Mean False Positives/Second by Packet Loss for N=4
FP_Per_Second
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Packet_Loss
0.05
0.10
0.15
0.20
0.25
0.30
Correlation = 0.9943594


Standard Deviation for N=2 Plot
Standard Deviation
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
Packet_Loss
0.05
0.10
0.15
0.20
0.25
0.30


Standard Deviation for N=4 Plot
Standard Deviation
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
Packet_Loss
0.05
0.10
0.15
0.20
0.25
0.30