What is this Gossip Protocol about?

This **Gossip Protocol** a membership protocol. Since it is infeasible to run thousand cluster nodes (peers) over a real network, we have stimulated by implementing an emulated network layer (EmulNet). The membership protocol implementation will sit above EmulNet in a peer to-peer (P2P) layer, but below an App layer. Think of this like a 3 layer protocol stack with App, P2P, and EmulNet as the three layers (from top to bottom).

More details are below: -
Your protocol must satisfy:
i) Completeness all the time: every non-faulty process must detect every node join, failure, and leave,
ii) Accuracy of failure detection when there are no message losses and message delays are small. When there are message losses, completeness must be satisfied and accuracy must be high. It must achieve all of these even under simultaneous multiple failures.

The Three Layers

The three layer implementation framework we are providing will allow you to run multiple copies of peers within one process running a single-threaded simulation engine. Here is how the three layers work.

Emulated Network: EmulNet

EmulNet provides the following functions that Gossip membership protocol should use:
1. void *ENinit(struct address *myaddr, short port, char *joinaddr);
2. int ENp2psend(struct address *myaddr, struct address *addr, char *data, int size);
3. int ENrecv(struct address *myaddr, int (* enqueue)(void *, char *, int), struct timeval *t, int times, void *env);
4. int ENcleanup();

ENinit is called once by each node (peer) to initialize its own address (myaddr). ENp2psend and ENrecv are called by a peer respectively to send and receive waiting messages. ENrecv enqueues a received message using a function specified through a pointer enqueue(). The third and fourth parameters (t and times) are unused for now. You can assume that ENsend and ENrecv are reliable. ENcleanup is called at the end of the simulator run to clean up the EmulNet implementation. These functions are provided so that they can later be easily mapped onto implementations that use TCP sockets.

To facilitate this in the future, calls to the above functions are made respectively through function pointers MPinit, MPp2psend, MPrecv, MPcleanup.

Application: App

This layer drives the simulation. Files app.c,h contain code for this. Look at the main() function. This runs in synchronous periods (globaltime variable). During each period, some peers may be started up, and some caused to crash-stop. Most importantly, for each peer that is alive, the function nodeloop() is called. nodeloop() is implemented in the P2P layer (mp1 node.fc,hg) and basically receives all messages that were sent for this peer in the last period, as well as checks whether the application has any new waiting requests. Please do not modify these files app.c,h given to you.

P2P Layer

The functionality for this layer is pretty limited at this time. Files mp1 node.c,h contain code for this. This is the layer responsible for implementing the membership protocol. As such this is where your code should be implemented. You can very well imagine the P2P layer can be extended to provide functionalities like file insert, lookup, remove etc.

You can turn debugging on or off by commenting out the #DEFINE DEBUGLOG in stdincludes.h.

Two message types are currently defined for the P2P layer (mp1 node.c implementation) - JOINREQ and JOINREP. Currently, JOINREQ messages are received by the introducer. The introducer is the first peer to join the system(for Linux, this is typically 1.0.0.0:0, due to the big-endianness).

Logging
log.c,h has a LOG() function that prints out stuff into a file named dbg.log. Also it implements two functions logNodeAdd and logNodeRemove. Whenever a process adds or removes a member from its membership list, make sure you use logNodeAdd and logNodeRemove to log these respectively. These functions take two address parameters - pass the address of the recording process as the first parameter and the address of the process getting added/removed as second parameter.

What are these Other Files?
params.c,h contains the setparams() function that initializes several parameters at the simulator start, including the number of peers in the system(EN gpsz), and the global time variable globaltime, etc. queue.c,h has an implementation of a queue. The remaining files nodeaddr.h, requests.h, MPtemplate.h list some necessary definitions and declarations.

How do I Test my Code?

Testing
To compile the code, run make.
To execute the program, from the program directory run: ./app testcases/test name.conf
The conf files contain information about the parameters used by your application:
MAX NNB: val
SINGLE FAILURE: val
DROP MSG: val
MSG DROP PROB: val

where MAX NNB represents the max number of neighbors, SINGLE FAILURE is a one bit 1/0 variable that sets single/multi failure scenarios, MSG DROP PROB represents the message drop probability (between 0 and 1) and MSG DROP is a one bit 1/0 variable that decides if messages will be dropped or not.

There is a grader script Grader.sh. It tests your implementation of membership protocol in 3 scenarios and grades each of them on 3 separate metrics. The scenarios are –
1. Single node failure
2. Multiple node failure
3. Single node failure under a lossy network.

The grader tests the following things –
i) Whether all nodes joined the peer group correctly,
ii) Whether all nodes detected the failed node (completeness) and iii) whether the correct failed node was detected (accuracy). Each of these is represented as configuration files inside the testcase folder.