Clarifications and F.A.Q. on Program 2

This document clarifies Programming assignment 2. If you have questions, you might check here first.

What happens when failures occur? Can't this result in a network partition?

Yes, temporarily. This is repaired because servers send out Joins to all *neighbors* for every channel they are subscribed to every minute. Note that the Join is sent to neighbors, not just the other servers in the routing tables. This will rebuild the tree, assuming it's possible.

Of course, with a topology like: 1--2--3, if 2 really does fail, then the network is partitioned and there's nothing you can do about it. So, don't worry about that case! However, if you have:

```
  S1
 /  \
S2   S3
 \  /
  S4
```

and the tree for a channel is S2, S1, S3, but S1 fails, then when the periodic Join message is sent, the tree should span S2, S4, S3.

How long do I keep track of message ids?

You can keep track of them forever, though in a real system you wouldn't do this. We will not send so many messages that it's a problem. You can also assume that if you haven't received a duplicate message after a couple of minutes, then you can forget the message id. The actual upper bound is the maximum delay on the longest loop in the graph of the topology.

I don't understand how trees are built! Help!

First, when a client connects to a server and joins a channel, the server sends a Join request to all neighbors. These neighbors are determined by the neighbor list you specified on the command line. That server then builds a routing table for that particular channel. In our case, the routing table is simply a list of other servers that messages on this channel must be sent to. For example, assume we have 3 servers that have just started (so no one has joined any channels) and our topology is:

```
S1---S2---S3
```

Now, let's say a client connects to server 1 (S1) and joins the Common channel. In this case, S1 sends a S2S Join Common request to S2. Internally, it might have a routing table that looks like this: Common: S2. S2 receives the Join Common request and forwards the request to S3. S2's routing table would look like this: Common: S1, S2. S3 receives the Join Common request and has no one to forward it to, so the request dies there.

We have formed the initial "tree", though initially the routing tables may not form an actual tree, in case there were loops. For example:

```
S1--S2--S3
|   |    |
S4--S5--S6
```

In this case, the initial Join message will create entries in the routing table such that the "tree" has loops and exactly mimics the topology of the graph. It's not until a client actually sends a Say message that loops are broken and a true tree results.

To understand how this works, let's assume the above topology. A client at S1 sends a say message. S1's routing table is: Common: S2, S4. It sends the message to those people. However, S5's routing table is: Common: S4, S2, S6. Let's assume the S2S Say message arrives at S5 before the S2S Say message from S4 arrives. When S5 receives S4's S2S Say, it will see that it's a duplicate and send a Leave message to S4. It then updates its own routing table to be: Common: S2, S6. S4 receives the Leave message so it updates its routing table to be S1. Thus, at this point, the "tree" looks something like:

```
S1--S2--S3
|   |    |
S4  S5--S6
```

The other loop will produce a duplicate message, so another Leave message will be generated and you'll end up with a real Tree for the Common channel.

You said you would provide a script to start up servers. Where is it?

The file start_servers.sh is a shell script that you can use to start a bunch of servers at once. Just uncomment the topology you want to create and run the script. If you want to make your own topology, draw it on paper, label the nodes, and then figure out neighbor list for each node so that you can specify it in the script.

When do we prune servers out of the tree exactly?

Only when they are leaves. To determine if a server is the leaf in a channel tree, you 1) make sure that no clients are subscribed to that channel and 2) check to see that only one other server is connected to this channel. If these two tests pass, you can remove the server (ie, send a leave message to the single server in the tree you're connected to).

Can an odd number of arguments be passed to the server?

You should check that the number of arguments passed to the server is even, as it should be. Each pair of arguments is an IP address and port number for an additional server.