# **Project 3 ReadMe**

## Team members:

Isabel Laurenceau 7393-5064

Anshika Saxena 9530-5566

Input: To run project three use ‘mix run project3.exs [number of nodes] [number of requests]’

Output: Maximum number of hops traversed for all requests for all nodes.

## What is working:

The project3.exs reads the input arguments and starts the GLOBALSUP. We use a high level supervisor so that we can terminate once we hear back how many hops all requests took to resolve. GLOBALSUP initializes the the MAINPROJ module with the input arguments. The MAINPROJ starts the dynamic supervisor TAPESTRY module, starts all the nodes from TAPNODE module and adds them to the Tapestry mesh. Once they are all inserted to the mesh the MAINPROJ tells each node [number of requests] random objects from the Tapestry mesh. As requests are fulfilled they send the number of hops it took back to MAINPROJ which adds it to its known number of hops. Once it receives the number of hops it is expecting (number of nodes \* number of requests) it gets the max, outputs it to the terminal and terminates the program.

#### JOIN

The TAPNODE initializes all nodes with a randomly created number which is hashed using the SHA1 algorithm to produce it’s id. It is also initialized with an empty neighborMap, empty objectList and empty objectLinksList. The id and three lists will be used later for inserting into the tapestry and routing.

To add each node to the tapestry *addToTapestry*is called. Here we contact a contact Gateway Node using *contactGatewayNode* which returns the id of a node already in the Tapestry mesh. We (node N) than use this gateway node to route to where we should be in the mesh and fill in our neighborMap. *hNodeToRoute* first sends a hello message to nodes it encounters so that they made add us (node N) to their neighborMap.

When a node (H) receives a ‘Hello’ message from node N it uses *placeInNeighborMap(state, neighbor\_id)* to add it to it’s neighborMap. *placeInNeighborMap* first finds the longest matching prefix between node H and node N’s ids. It than finds i, which is the element in the index of the next id digit after the prefix of N. It uses j and i to place N in its neighbor map. If there is already an element in that location it uses *updateYourNeighborMap(j, my\_neighborMap, new\_neighbor)* to select the primary and backup links. If not it adds N to its neighbor map and tells B to add it to its neighbor map since its bidirectional.

For N to populate its neighbor map it uses routing. If there is an element where it would be in B’s routing table it routes to that element, copies that level that’s its routed too and tries to find a possible closer neighbor. It continues this until no neighbors are available. An interesting observation made was that as the network is small it’s more likely you will not have anything in common with the gateway node and that the gateway node will not have anything in common with its neighbors. This leads to very large first levels and an almost fully connected network. This is not the case with larger networks as it is more likely that you “match” prefixes with other elements and can better place yourself.

#### ROUTING

To Route an object MAINPROJ tells a node which target id to route towards. *RouteToObject* finds the repfix match length to check the neighbor map to see if that level exists. If so it gets that level and checks if there is a matching node. If so that node is a neighbor and you can send a direct message. If not the *findNetHop* is used to find the next neighbor closest to the target id. Once *findNextHop* finds that neighbor it contacts *nextHop* to move to that neighbor and repeat the process checking there.

## What is the largest network you managed to deal with: