# **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 dynamic supervisor DyServer which initializes the Node module arguments. We then add all the node arguments to the Tapestry mesh. Once they are all inserted to the mesh the we create and spread the objects throughout. Each node sends [number of requests] to random objects from the Tapestry mesh.

#### JOIN

The NODES initializes all nodes with a randomly created number which is hashed using the SHA1 algorithm to produce its id. It is also initialized with an empty neighborMap, empty objectList and empty objectLinksList and Max\_hop\_node as 0. The id and three lists will be used later for inserting into the tapestry and routing. Max\_hop\_node stores the maximum hop it took to complete the requests.

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. It also tells N to add themselves (B) to their map. If there is not already an element, 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.

#### OBJECTS

Before we can begin requesting objects, we first need to spread them through the tapestry mesh. To do this we assign each object a random id, created from the same hash as before. We than find its closest root node and add that object there. As described in the literature we also assign the object to two random duplicate backup nodes. We can now start sending requests.

#### ROUTING

The script tells a node which random object id to request. A new module ROUTING101 calls *routing\_route* which ultimately returns the number of hops it took to route to the object requested. To achieve this, we first check to see if that object is in our object list. If it is, we have finished and can return. If not, we return with the next node we need to route to get closer to our desired object and keep repeating it until our request for the object is fulfilled.

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

2000 nodes with a maxHop of 1