# DISTRIBUTED OPERATING SYSTEM
# PROJECT-3

**Group Members:**
Name: Shashank Bantakal UFID: 9592-6559
Name: Santhosh Suresh UFID: 5985-1919

## Implementation Details:

This project implements the Tapestry algorithm including its features such as dynamic node insertion, distributed routing table, prefix based routing and fault tolerance.

### TAPESTRY ALGORITHM

- Each node is assigned a unique identifier.
  - The identifiers are assigned to nodes at random.
  - In our implementation of tapestry, the identifier length is dynamic, i.e. the least length of hexadecimal digits required to represent the number of nodes required.
  - Root nodes are the obtained by prefix matching. Nodes with the highest possible matching prefix are considered to be root nodes.
  - **Dynamic node insertion** has been implemented to add nodes to the network as & when they are created.
- Every node maintains a **Routing Table (RT)**.
  - No of levels in the routing tables is equivalent to the no of digits in the node identifier.
  - No of columns in each level is 16, i.e. base of the identifier space.
  - The level represents the size of the shared prefix with the local node; i.e., a node on level n of the routing table shares a prefix of length n with the local node.
  - The slot is defined by the first digit after the shared prefix, that is, id[n] where id is a list of the digits of the node's ID.
- **Prefix Routing** is used to route message to a destination node.
  - In the selected node's routing table is inspected and the next node in the route to the root is chosen.
  - At each successive node in the route, the number of digits that match the object's hash value increases until the last digit has been matched & the destination node has been reached.

**TESTING**

In this implementation of the Tapestry network model, the number of digits in the node identifier's are being selected dynamically depending on the number of nodes to be created.

We experimented by varying the total number of nodes to get 3 digits, 4 digit and even 5 digit node identifiers. In each case, it was observed that the maximum number of hops required to reach the destination node is equal to the number of digits required to represent the node identifiers .

**BONUS - TAPESTRY NODE FAILURE**

To begin with as the network is being built, a certain percent of the node (as decided by user input) would be marked as being "**Failed**"

This implementation of the failure handling module for Tapestry maintains a Backup Routing Table (BRT). The structure of the BRT is similar to that of the Routing Table (RT). Each BRT slot contains a list of possible nodes (maximum of three nodes in each slot) that could be used for message routing in case the node in the original routing table is failed.

As the message traverses through the network hoping through node, if a successive node in the route is failed, then a nearest active node from the corresponding BRT slot is selected, this node is identified as the new successive node.

It was observed that even with certain percent of the nodes marked as failed, the maximum number of hops required to reach the destination node is equal to the number of digits required to represent the node identifiers .