ECE 658 - Design Report

Avinash Pallapu & Oscar Rodriguez

Topology :

We will be using unstructured topology, where all the nodes will be connected to each other. A node initially sends a REGISTER_REQUEST to the bootstrap server to register itself, upon which it will get the acknowledgment and the details of the other two closest or randomly chosen nodes in the network.

The node will connect to these nodes using the JOIN messages. Like this, each node will be joined to the existing network, thus forming our desired overlay network.

Communicate among nodes:

We will use UPD to send commands from one node to the other. The node will have a port in which they will always be listening for request. When a request comes in it will parse and execute the request in a separate thread, and will go back to listening right away. Nodes will only be able to communicate with the neighbor nodes they know about and with the bootstrap server.

Routing table:

We will be using a dictionary to store the ip addresses along with port no as the key and the files it contains as the values. In order for Nodes to perform search query the node wanting the file will ask it's two neighbors if it has the files, the neighbors will check if they have the files and if they don't they will ask their neighbors, the search query will have an ID, so if a node gets a search query with the same ID it will not forward it again, thus avoiding heavy broadcast traffic and infinite loops when looking for a specific file.

performance parameters:

The performance of our query will be O(n) because it will search through all the nodes. We will capture the parameters by printing out the amount of hopes it took to locate the file. We will also capture the performance by seeing the time it took to find the different files, and how this changes by having 20, 40, or 80 nodes.

Pseudo Code:

For searching the files based on the query:

```
for f in files_at_node:
   if f in query:
      send response to the requesting node
   else:
      askNieghbors(query)
```

The above search method will only be processed one time for each search, since searches will have searchID, the node will check the search ID and it will make sure it was not processed it already, if it has not then it will perform the search above. Each node will have a searchID list of 50 searchID that it has recently processed, that way it has some memory of which searches it has done recently.

Each request will have a key word at the beginning letting the server node know what type of request it wants, for example the keyword could be search, or retrieve. When a node receives a request it will use a switch statement to parse and then perfrom the actions, For example:

```
parseRequest(String request):
        actions = request.split(" ");
        keyword = actions[0];
        file        = actions[1];
        requester= actions[2];

        switch(keyword):
                case "search": searchForFile( file, requestor)
        break;

        switch(keyword):
                case "retrieve": retreiveFile(file, requestor)
        break;
```

We plan to implement the code in a very object oriented way, which will enable us to use this code if future assignment, as you can see by the sudocode above we will have everything in different methods, which will facilitate debugging, and allow us to avoid long spaghetti sting code.