# PROJECT REPORT

## PASTRY PROTOCOL

**Team Members:**
Arunima Agarwal (UFID: 3397-1331)
Karan Acharekar (UFID: 3868-9483)

**Description:**
The purpose of this project is to use Elixir and the actor model to implement pastry protocol and a simple object access service to prove its usefulness.

*numNodes* is the number of peers to create for the peer-to-peer system.

*numRequests* is the number of requests each peer will make per second.

**Pastry Protocol:**
In the pastry protocol each node added in the network is given a unique identifier (node Id) which is 128-bit long and is evenly distributed in a circular nodeId space that is ranging from 0 to $2^{128} – 1$. For routing purposes the nodeIds and keys are converted in base $2^b$ where b = 4 (in our case). Apart from the identifier file name is taken which is generated randomly and hashed using SHA-256, with the length equal to the length of the node id.

Now the protocol works as follows:

1) Initially the node joins one-by-one and update their state which consists of routing table, leaf set and neighborhood set.
2) Now once every node has initialized its state, the node will start the requests to search the location of file in the peer network or find the nearest node that matches the file key.
3) This file name will be hashed and act as a key to find the location of this file.
4) Now each node will make few numRequests which is equivalent to number of hashed files and thus find the location or nearest location of these files.
5) The hash value or the key of the file is matched with the local node Id, to check how many prefixes will match. The rows of the routing table define how many bits' match of the key with that of the node id in each column i.e. row 1 will give all the nodes whose 1 bit match with that of the key and so on.
6) Now once some prefix of the key is matched with some node id or its neighbor, it will send message to that node which will further match the prefix in its neighbor node list and send the message further till it either reaches the node id same as the key or reaches a node which is at minimum distance from that of key.
7) Each call to the neighbor with which prefix match will trigger send message function which means it is a step closer to its destination hence we will increment the hop count on each send message call. Thus, at the end when it reaches the node which is either

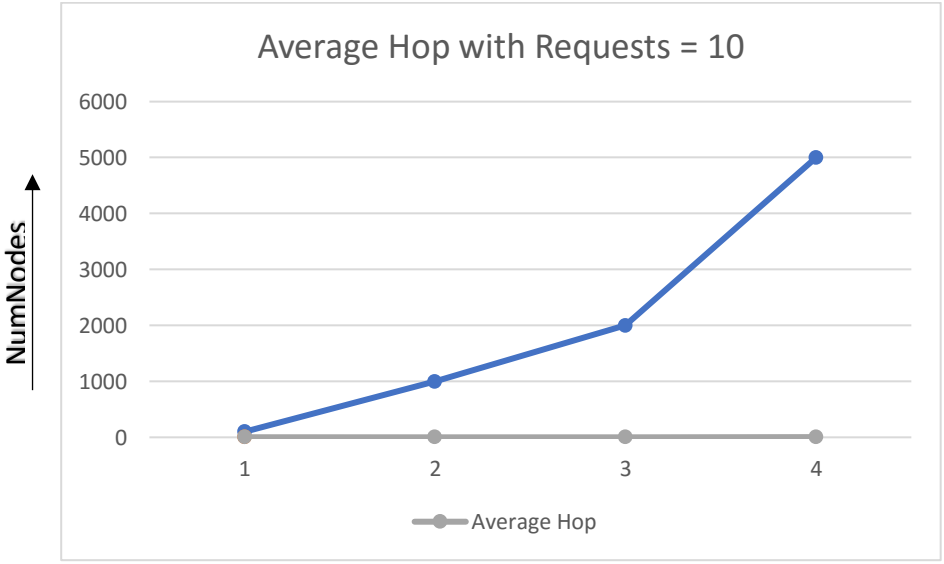
same as the key or near to the key it will return the number of hops to deliver the message to the destination.

In our implementation, we have taken a few files and hashed it to find the key. Each node will be responsible for making the requests equal to the file names generated. Also, each request is made at a gap of few milliseconds. As soon as all the nodes complete their requests to find the location of the key. The program will stop.

**The table representing number of nodes, each making 10 requests and the average hops as below:**

| NumNodes | NumRequests | Average Hop |
|---|---|---|
| 100 | 10 | 1.91 |
| 1000 | 10 | 2.79 |
| 2000 | 10 | 2.81 |
| 5000 | 10 | 3.69 |

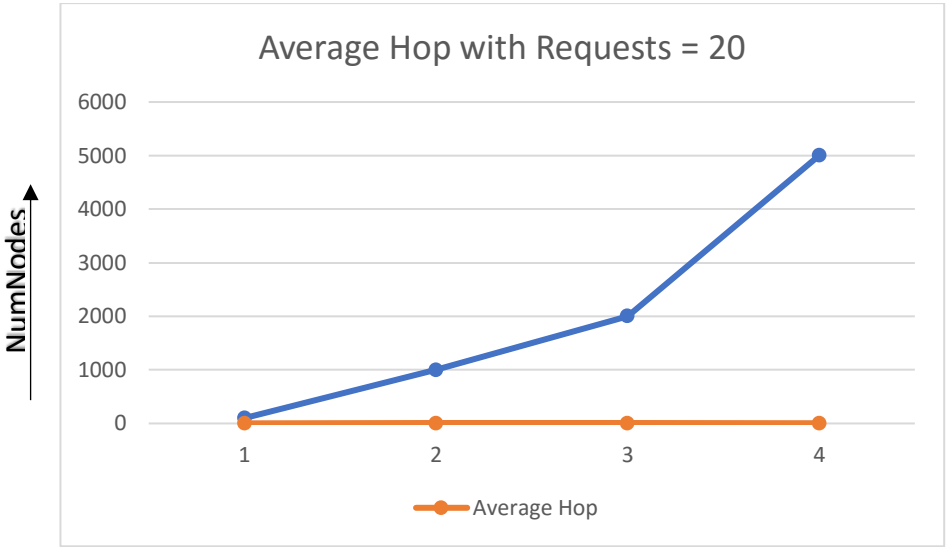
**The Graph plotted with the above values is as follows:**



**The table representing number of nodes, each making 20 requests and the average hops as below:**

| NumNodes | NumRequests | Average Hop |
|---|---|---|
| 100 | 20 | 1.63 |
| 1000 | 20 | 2.81 |
| 2000 | 20 | 2.83 |
| 5000 | 20 | 3.67 |

**The Graph plotted with the above values is as follows:**




**Observations**:

1) The average number of hops between any pair of nodes is $log_2^b N$. So, from the above table we can see that the $log_2^b N$ value for each given node where N is the number node. We will take the ceil value of $log_2^b N$. Thus, the average hops are in range to the value computed as below:

| NumNodes | $log_2^b$ N |
|---|---|
| 100 | 2 |
| 1000 | 3 |
| 2000 | 3 |
| 5000 | 4 |

2) From the above tables, we can see that even if we increase the numRequests the average hop will be in the same range as computed above.