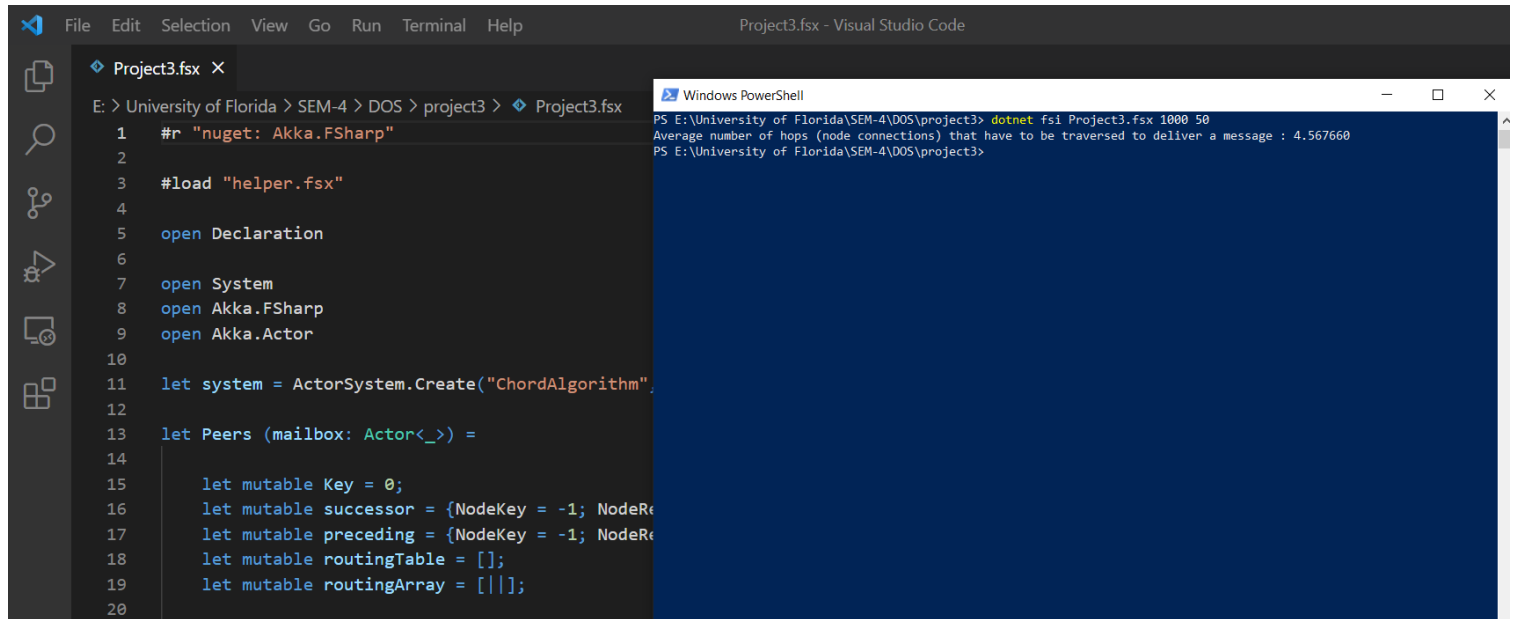


# COP5615 - Distributed Operating System Principles

## Project 3 – Chord Algorithm (P2P)

Submitted By: Parth Gupta, UFID: 91997064

### Sample Output



The screenshot shows the Visual Studio Code editor with the file 'Project3.fsx' open. The code in the editor includes comments for loading Akka.FSharp and helper.fsx, opening namespaces for System, Akka.FSharp, and Akka.Actor, and creating an ActorSystem named 'ChordAlgorithm'. It also defines a 'Peers' mailbox actor with mutable fields for Key, successor, preceding, routingTable, and routingArray. A Windows PowerShell terminal window is open to the right, showing the command 'dotnet fs1 Project3.fsx 1000 50' and its output: 'Average number of hops (node connections) that have to be traversed to deliver a message : 4.567660'.

### Some results of my algorithm:

Number of Nodes / Peers	Number of Requests	Average Number of Hops
100	10	2.936000
500	10	4.062600
1000	10	4.565500
500	50	4.064400
1000	50	4.565960
1000	100	4.590240
1500	100	4.948467
2500	100	5.295244
2500	150	5.369304
3000	150	5.437124
4500	200	5.760827
5000	200	5.795034
<b>5500</b>	<b>200</b>	<b>5.920152</b>

## What is Working ?

- 1) Implemented the network join and routing as described in the Chord paper and encoded the application that associates a key with a string.
- 2) After implementation of Chord Algorithm, I am printing the average hop count that has to be traversed to deliver a message.

## The largest network that I managed to deal with:

- 1) For this project, the largest network that I was able to manage was for **numbers of nodes = 5500 and numbers of requests = 200**.

## Project Description and Implementation

- 1) The input provided (as command line) will be of the form: numberOfNodes and numberOfRequests, where numberOfNodes is the number of peers to be created in the peer-to-peer system and numberOfRequests is the number of requests each peer has to make. When all peers performed that many requests, the program can exit. Each peer should send a request/second.
- 2) I have implemented the program that will print the average number of hops (node connections) that have to be traversed to deliver a message.
- 3) The entire project has been divided into two files namely: helper.fsx and Project3.fsx.
- 4) helper.fsx: this file contains all the structures and declaration of functions that are necessary for the main program to run where the actors have been defined. This file include structures like node /peer info, finger table / routing table info, node setup info, etc. Apart from that it also contains two function definitions which include generating a hash and generating a random string. Also, the logic for taking command line arguments has been written in this file. The hash
- 5) Project3.fsx: firstly, the helper file has been loaded in this file. After that the program continues. This file consists of mainly two important things namely: The boos actor who starts the process by taking in a string (any random) as an input. Secondly there is an actorList in which will have all the nodes / peers created. Basically, spawning number of actors equal to number of nodes provided by the user.

- 6) Each time the actor is created the following information is updated and stored in the routing table: update the key in the routing table for each new input, set up the node info, update the previous node every time we encounter the new node, join the nodes, and finally update the routing table.
- 7) Whenever a new node joins, each node's successor points to its immediate successor correctly, each key is stored in successor. Therefore, to ensure this predecessor field is maintained for each node. Whenever we encounter new node, all the nodes need to update their predecessors and then the routing / finger table is updated. This is how the node join works.
- 8) Each peer makes the number of request entered by the user and as soon as the total requests are equal to  $(\text{number of nodes}) * (\text{number of requests})$  the program prints the average number of hops to deliver the message and the program exists.