COP5615 – Project3

Neel Rami, UFID: 7712-3151

Ma Haodi, UFID:7719-2198

Email: nrami@ufl.edu , ma.haodi@ufl.edu

Oct 1, 2018

1. Introduction

The goal of this project is to implement in Elixir using the actor model the Chord protocol and a simple object access service to prove its usefulness.

In this project we implement the network join and routing as described in the Chord paper (Section 4) and encode the simple application that searches a key (same as the ids used in Chord). We implement it using a similar API to the one described in the paper. The result and findings will be shown in this report.

Assumptions made:

- We have built a simple application of doing a search for a key. This key is randomly generated, and we are not associating this key with a string just for simplicity.
- We are not storing keys in a node, but we assume that all keys from predecessor of a certain node to that node are stored in a node. For example, if there is a node with ID 85 and its predecessor is 51, then all keys from 52 to 85 are assumed to be in node with ID 85.
- In rare cases during network Join, the process of stabilization doesn't get completed due to some reasons which can lead the program to go into a infinite loop. So in those cases, please run the program again.

2. Implementation details

a) File explanation:

1) **GenServerMethod.ex**:

   This file contains methods which are related to GenServer and it contains all methods which are essential for Chord Protocol.

2) **proj3.ex:**

   This file serves as entry point of the project.

3) **utilityFunctions.ex:**

   This file contains some utility functions which are used frequently.

b) Approach used:

1) Calculation of Node IDs:

- We have used the concept of consistent hashing to calculate the identifier for every node.
- We have used the :crypto.hash(:sha “nodeID”)function to calculate SHA1.
- We calculate the hash of Node Index(i.e. 1,2,3…) to calculate the hash. Then we encode it to BASE 16 and truncate it to m bits. And then we convert the truncated result to integer and this integer serves as the node identifier for a node.

2) Calculation of Table size(m):

- We dynamically calculate the table size based on the number of nodes.
- We calculate $\log_2$(numNodes) to calculate the least number of bits required to represent a node and then we convert the logarithmic result to the nearest multiple of 4.

3) Network Join and Routing:

- We have implemented the functionality of Network Join by creating a Chord Ring of n-1 nodes and then join the remaining node.
- The functionalities of Network Join and Routing are implemented based on the explanation given in the research paper.

c) Instructions for running the code:

1) For Ubuntu based systems:
    1. Go the project directory
    2. Type the command in the terminal: mix escript.build (Optional)
    3. Type the command in the terminal: ./proj3 200 1
    4. Here the first command line argument is the number of nodes
    5. Here the second command line argument is the number of requests
    6. General command: ./projec3 <node-num> <numRequests>

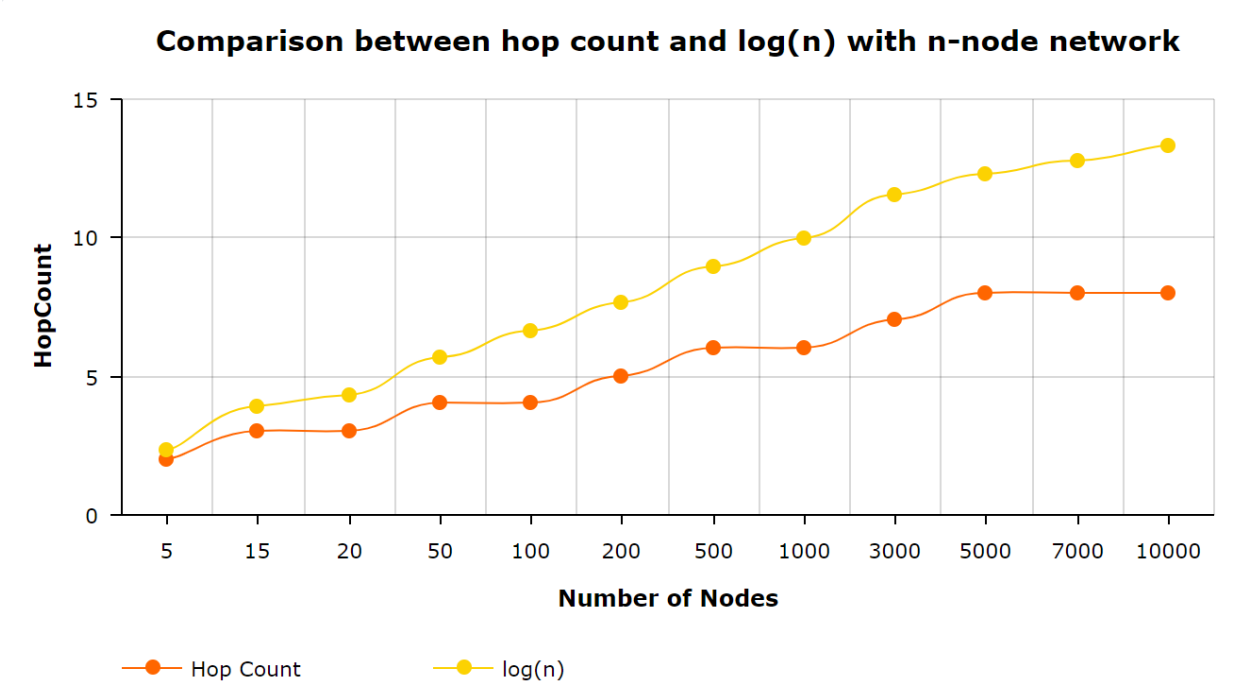
2) For Windows:
    1. Go the project directory
    2. Type the command in the cmd: mix escript.build (Optional)
    3. Type the command in the cmd: escript .\proj3 200 1
    4. Here the first command line argument is the number of nodes
    5. Here the second command line argument is the number of requests
    6. General command: escript .\projec3 <node-num> <numRequests>

3. Result

Number of nodes	Ceil(Hop Count)
5	2
15	3
20	3
50	4
100	4
200	5
500	6
1000	6
3000	7
5000	8
10000	8*

Table 1: Number of request = 1

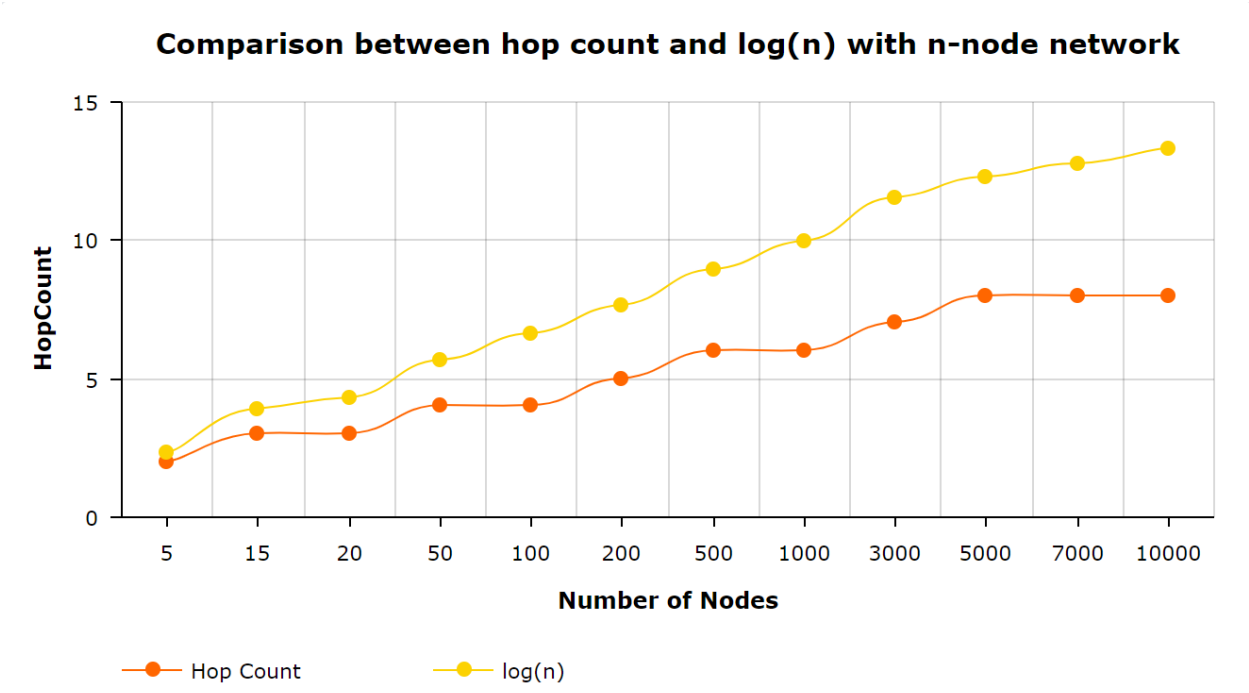
*: Takes a lot of time



Number of nodes	Ceil(Hop Count)
5	2
15	3
20	3
50	4
100	5
200	5
500	6
1000	6
3000	7
5000	7
10000	8*

Table 2: Number of request = 2

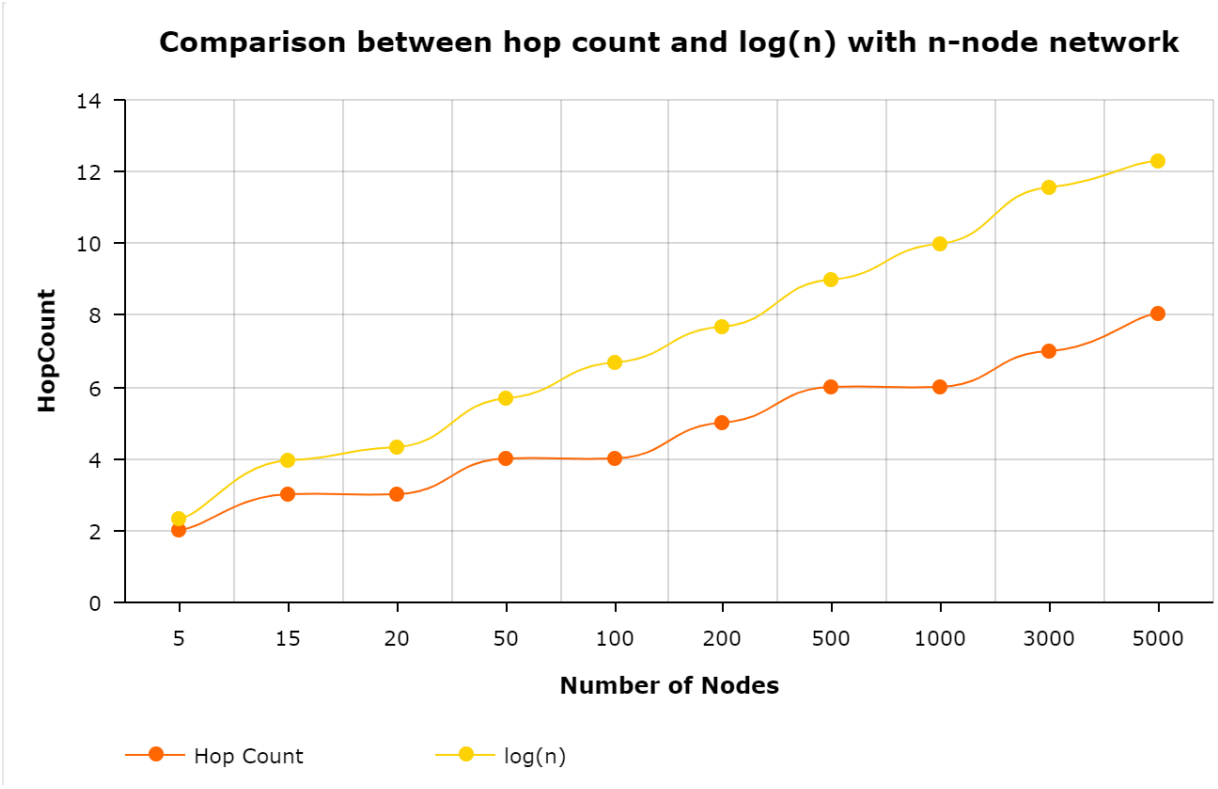
*: Takes a lot of time



Number of nodes	Ceil(Hop Count)
5	2
15	3
20	3
50	4
100	5
200	5
500	6
1000	6
3000	7
5000	7*

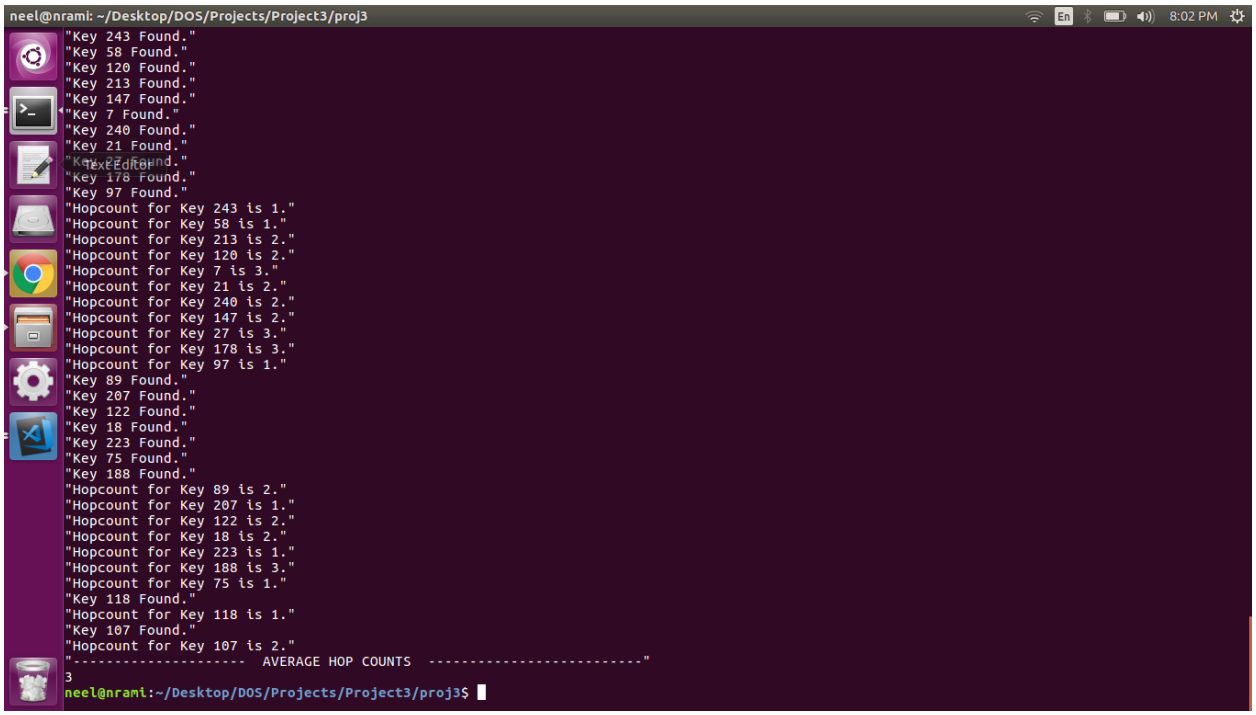
Table 3: Number of request = 5

*: Takes a lot of time

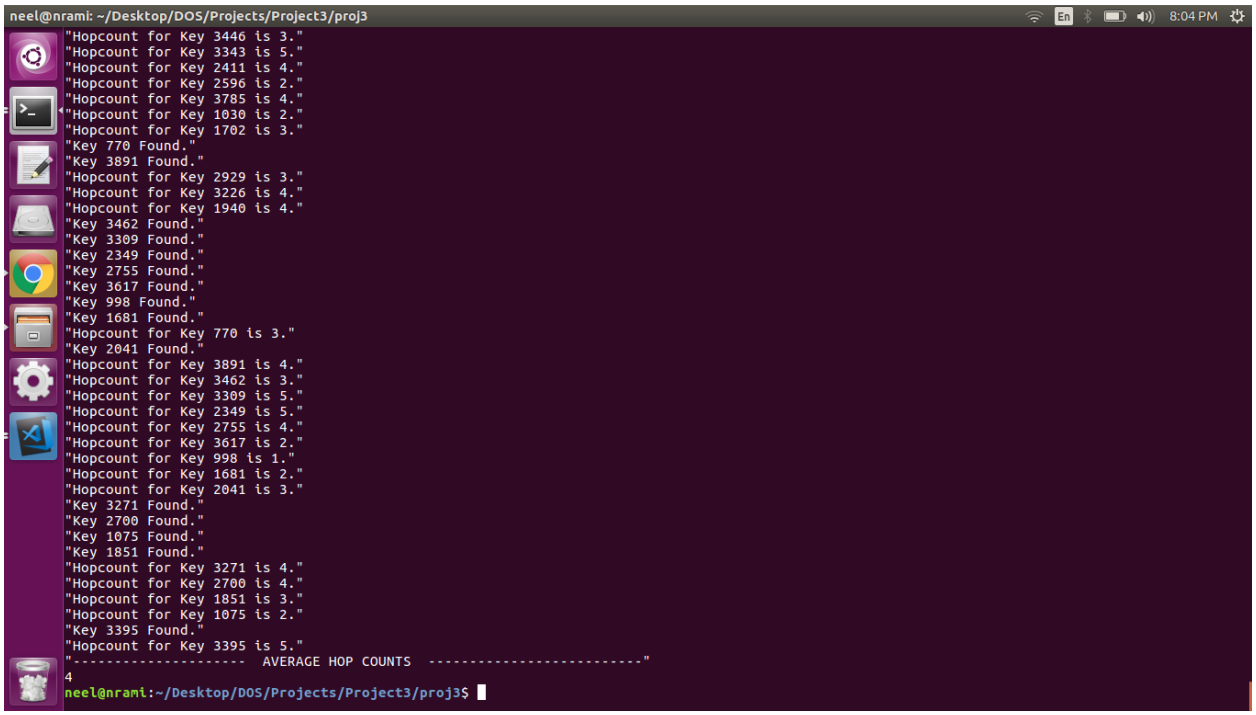


4. Output

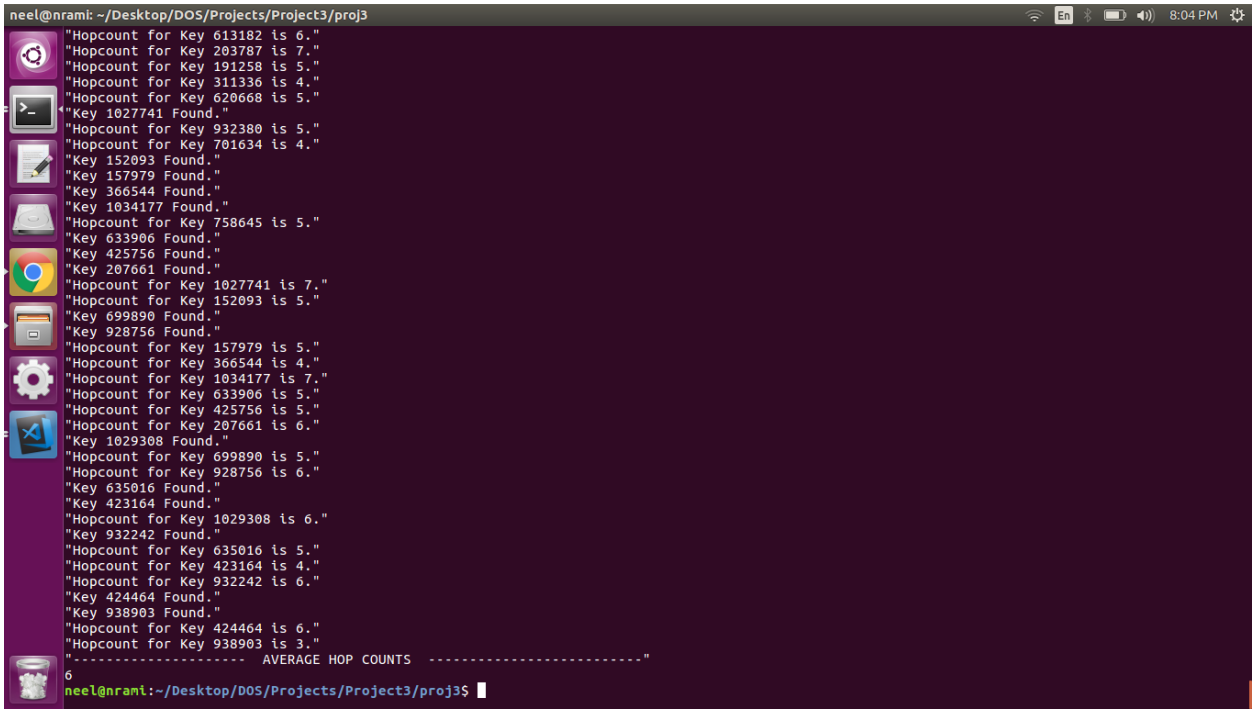
1. Output for numNodes=20 and numRequests=1



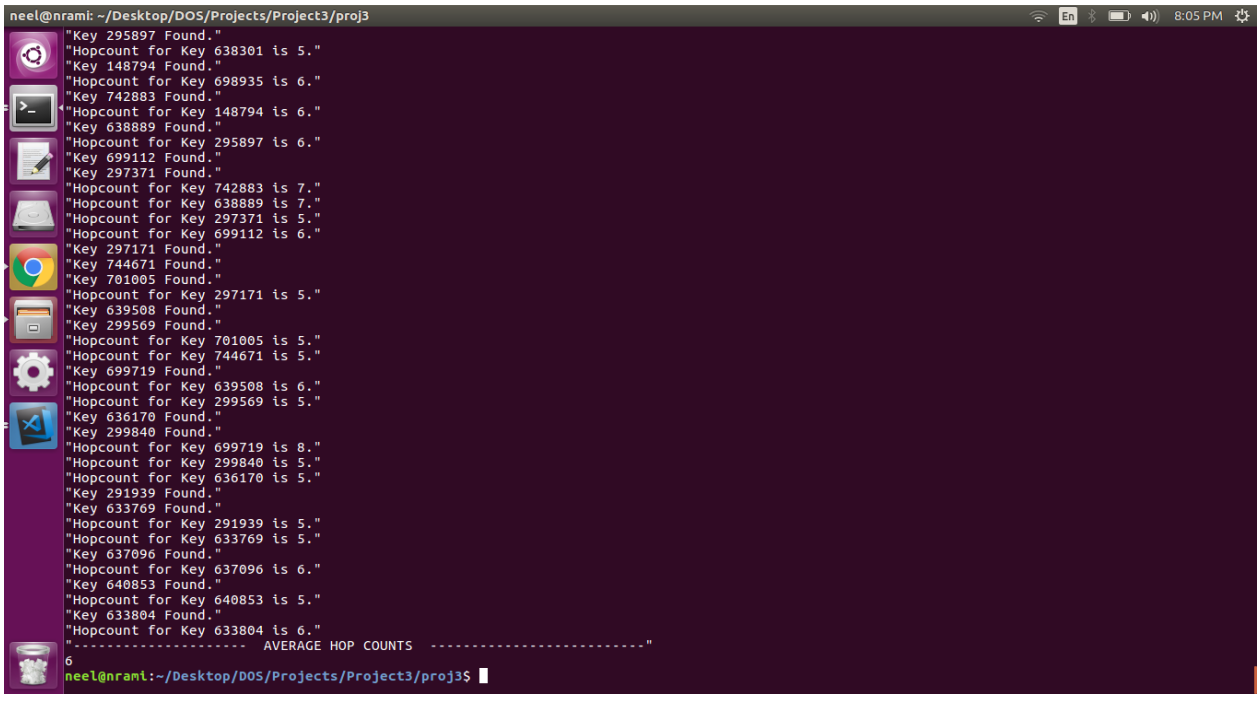
2. Output for numNodes=50 and numRequests=1



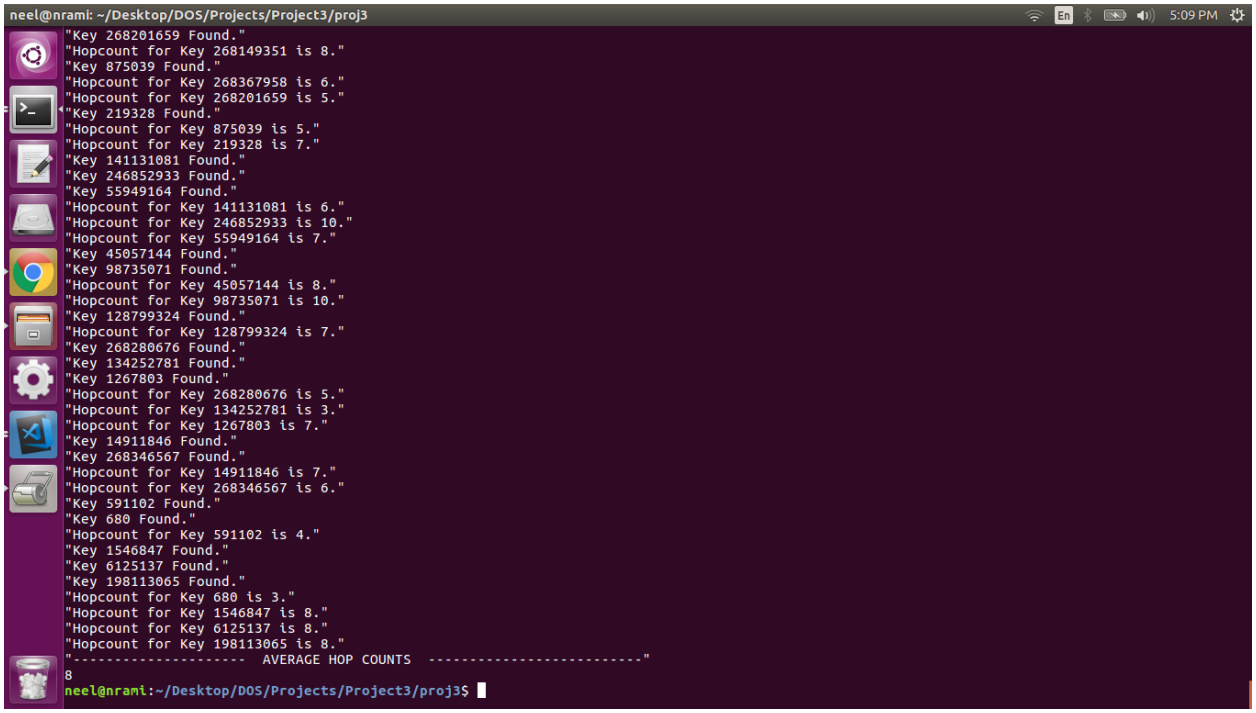
3. Output for numNodes=500 and numRequests=1



4. Output for numNodes=500 and numRequests=5



5. Output for numNodes=10000 and numRequests=1



5. Largest Network

The largest network on which our program ran successfully was 10000 nodes.