**RESULT ANALYSIS DOCUMENT LAB V**Distributed Hash Table Based Content Searching in a Structured Peer-to-Peer Network

Eshwar Reddy Pasula (830531654), Gayatri Pendharkar (830670002)

**INTRODUCTION:**

The main motive of this lab assignment is to devise a TCP based structured-overlay-based solution that allows a set of nodes to share contents amongst each other. A Distributed Hash Table (DHT) based content/resource discovery system for a structured P2P network is to be developed. We have used the Chord as the underlying DHT scheme. Chord is a protocol developedfor a peer-to-peer distributed hash table. A distributed hash table stores key-value pairs by assigning keys to different nodes; a node will store the values for all the keys for which it is responsible. Chord is based on consistent hashing, which assigns hash keys to nodes in a way that doesn't need to change much as nodes join and leave the system. The Chord protocol supports just one operation: given a key, it will determine the node responsible for storing the key's value. This lab can be distinguished in two different phases:  
  
PHASE 1: To generate the Chord overlay topology and the contents of each node. The network will consist of 20, 40, and 80 nodes sharing 160 entries among them, with each node contributing n entries.

PHASE 2: Search for a given entry name. Hash the given string and generate the search key. Find the peer responsible for the key. Retrieve the entry name, IP, and port number of the peer that contains the entry.

**DESIGN AND IMPLEMENTATION OVERVIEW:**

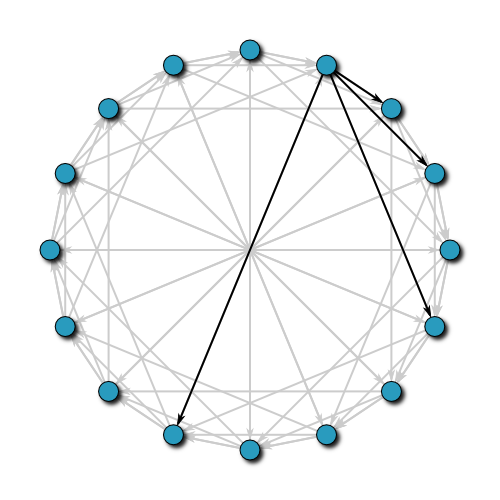


Fig. Chord Protocol for 16 nodes in network

**PHASE 1:**  
The figure above shows the implementation of the chord protocol with m=4. i.e. 2^m=16 possible positions for node placement. In this implementation, each node has a predecessor and successor.

The successor to a node (or key) is the next node in the identifier circle in a clockwise direction and predecessor is the one located just prior to the current node. The main purpose of this protocol was to design a hashing scheme to let nodes enter and leave the network with minimal disruption.

To maintain the consistent hashing mapping when a node n joins the network, certain key value pairs previously assigned to n's successor become assigned to n.

When node n leaves the network, all of its assigned key value pairs are reassigned to n's successor. No other changes in the assignment of key value pairs to nodes need to occur.

**PHASE 2:**

The chord protocol lookup proceeds in multiple hops around the identifier ring. Each hop eliminates at least half of the remaining distance to the desired successor.

This means that the hops early in a query's path travel long distances in identifier space, and later hops travel small distances.

Lookups take a number of hops proportional to the log of the number of nodes in the system. For each hop, the node doing the lookup sends a find successor message, and waits for the result; the result tells it where to send the next find successor message.

The Chord protocol needs to perform three operations:

1. Initialize the predecessor and fingers

2. Update the fingers and predecessors of existing nodes to reflect the change in the network topology caused by the addition of *n*

3. Copy all keys for which node *n* has become their successor to *n*. A new node *r* can initialize its finger table by querying an existing node for the respective successors of the lower endpoints of the *k* intervals in *r*'s table. Nodes whose routing information is invalidated by *r*'s addition are determined using *r*'s predecessor table.

The queries will be generated depending on the popularity which follows the zipf distribution.

**Implementation Steps:**

Each node sends a register message to the Bootstrap in the given format along with its IP address, the port number and key of its IP address and select random ‘n’ entries as its resources.

Each node finds its predecessor and successor and fills its Finger table. Also, the finger table sends a UPFIN message to the nodes with node ID = current\_nodeID + 2\*\*i-1 % 2\*\*m

An ADD message is sent in the network to add a new key in the network at a particular node location.

A GETKY message is sent to request keys that the new peer is responsible for, from the successor.

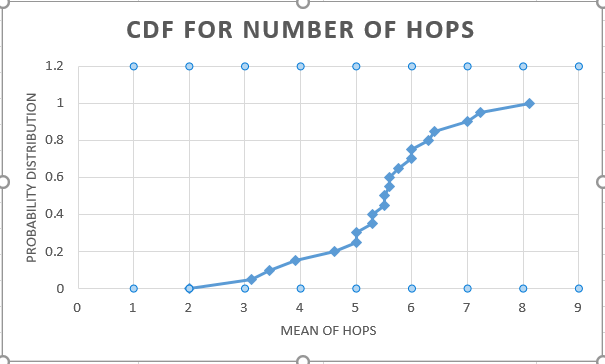
A GIVEKY message is used to send keys to the successor when a peer leaves the network

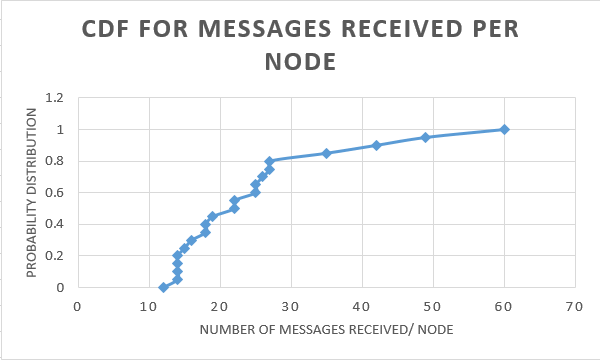
A SER IP message is used to locate a key in the network.

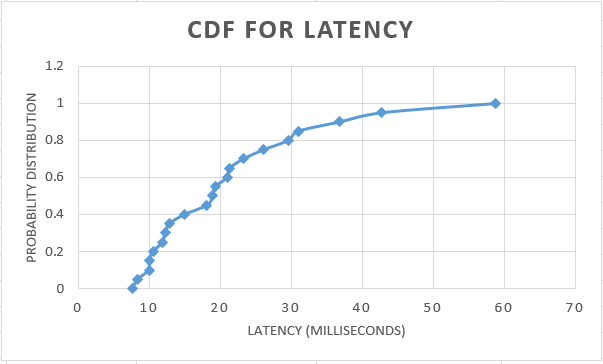
**Results and Analysis:**

1. s =0.8 and a 20 node network

|  |  |  |  |
| --- | --- | --- | --- |
|  | LAB3 (unstructured) | LAB4 (unstructured caching) | LAB5 (structured Chord) |
| Mean number of hops | 4 | 2 | 6 |
| Mean Latency | 433 ms | 612 ms | 28 ms |
| Messages received per node | 760 | 484 | 26 |







1. s=0.6 and 40 node network

|  |  |  |  |
| --- | --- | --- | --- |
|  | LAB3 (unstructured) | LAB4 (unstructured caching) | LAB5 (structured Chord) |
| Min Latency | 4.22E-3 | 5.34E-4 | 8.223E-2 |
| Max Latency | 3.1744 | 2.6120 | 7.012 |
| Std Deviation | 0.2131 | 0.2334 | 0.6875 |
| Min Hops | 1 | 1 | 1 |
| Max Hops | 3.981 | 5.23 | 26.46 |
| Std Deviation of Hops | 0.7811 | 0.8891 | 8.102 |
| Number of messages processed | 771.22 | 383.42 | 145 |
| Per query cost |  |  |  |
| Per node cost |  |  |  |

1. s=0.8 and 80 node network

|  |  |  |  |
| --- | --- | --- | --- |
|  | LAB3 (unstructured) | LAB4 (unstructured caching) | LAB5 (structured Chord) |
| Min Latency | 0.000126 | 1.29E-4 | 5.112E-3 |
| Max Latency | 2.698585 | 1.6506 | 5.991 |
| Std Deviation | 0.1575 | 0.124766 | 0.4031 |
| Min Hops | 1.1375 | 1 | 1 |
| Max Hops | 4.4875 | 7.3 | 39.231 |
| Std Deviation of Hops | 0.8375 | 1.4045 | 10.132 |
| Number of messages processed | 799.8375 | 464.1125 | 132 |
| Per query cost |  |  |  |
| Per node cost |  |  |  |