**Distributed Lookup using CHORD**

Siddharth Singh Mahendra Hegde Sudhendu Kumar

Requirements:

 The purpose of this problem is to understand the design of a basic Distributed hash table. [Chord](http://portal.acm.org/citation.cfm?id=383071) is a distributed lookup protocol that provides key-value based lookup. In the steady state, each node in Chord maintains information of only about O(log N) other nodes and resolves all lookups via O(log N) messages to other nodes.

This implementation will however deviate from Chord in many respects. The guidelines for the basic DHT is as follows:

* Each node is represented by program running on localhost in a port number. E.g., 3 Nodes can be running on localhost:50000, localhost:61111 and localhost:79999
* Each node's hash value is the hash value of the string combination of nodename and port number. E.g. the node of localhost: 50000 will have hash value of the string localhost50000. Hashing is done using MD5 hashing algorithm. Keys are also hashed using the same algorithm.
* Similar to consistent hashing, key to node assignment will be based on the hash values of keys and nodes. Key k will be assigned to the first node whose hash value is equal to or follows the hash value of key in the hash value space.
* The operations that the nodes should support from the client program's perspective are: PUT, GET, QUERY, END. You may choose to introduce other commands that would be used between node to node operations.
  + The PUT command is used to store the key,value pair. The format of PUT command is PUT:key:value. A PUT message to a node may need to be forwarded.
  + The GET command is used to retrieve the key,value pair by using the key. The format of GET command is GET:key. A GET message to a node may need to be forwarded.
  + The QUERY command is used to display all the key, value pairs stored by a node. A QUERY message to a node will not be forwarded. The format of QUERY command is the text QUERY.
  + The END command is used to end all the nodes. An END message to a node will require to be forwarded. The format of the END command is the text END.
* Each node will be run using the command
* dhtmain <n> where , n is the total number of nodes.

E.g., a 3 node DHT service would be run as:

dhtmain 3

dhtmain 3

dhtmain 3

Each node will run on a separate console, which will be used to display outputs of the client command.

A client will be run as:

client <portnumber> <command>

where command can be any of the commands mentioned above. For example, a client issuing PUT command to node localhost:50000 would be issued as:

client 50000 "PUT:key1:value1"

The client program needs to be extended to receive result values and output them (as seen in the example below).

* **Each node will only keep information of ⌊log N⌋ other hosts in a finger table**. During startup, hosts will initialize the node information. All other port numbers should be a random number in the range 1024..65535. You should use a local file called "nodelist", created by the first node, where each node appends its hostname:portno in a separate line. Once the file reaches n lines (where n is the maximal number of nodes), the last node initializes its fingers. It then sends a START message to its next neighbor in the ring. This message then propagates in the ring and results in finger initialization (by reading the entire file but only initializing its **⌊log N⌋ finger entries!**) on the other nodes until it reaches the initiator, where is it silently absorbed. Only after this point should queries with the above commands be issued (using the ports listed in the "nodelist" file). The "nodelist" file should be removed when the END message is received by one of the nodes (your choice which one).

The "dhtmain" should be enhanced to support following features:

* + **Join**: start another process as "dhtmain" without any parameter, which joins existing nodes
  + **Fail**: kill an existing "dhtmain" process (you need to use timeouts on requests to detect this case)
  + both join/fail require a redistribution phase
  + The **curr\_host** global variable should be implemented to figure out who the current host is for finger table initialization. This can be done via a read/write to the "nodelist" file(if no entries, curr\_host is 0, if 1 entry, curr\_host is 1, etc).

Sid: Why do we need to know of the current host when initializing the finger table? Can the hosts simultaneously update their finger tables?

* A sample output in the node console for a set of commands is given below.   
  Client commands:
* $ ./client 50000 "PUT:key1:value1"
* $ ./client 50000 "PUT:key2:value2"
* $ ./client 50000 "PUT:key3:value3"
* $ ./client 50000 "GET:key1"
* found key1:value1
* $ ./client 50000 "GET:key2"
* found key2:value2
* $ ./client 50000 "PUT:key53:value53"
* $ ./client 50000 "QUERY"
* $ ./client 50000 "END"

Node output:

$ ./dhtmain 3

DHT node (31eee38e22b6cf197c414b8c24aa576f): Listening on port number 50000 . . .

(key:value) =(key1 [c2add694bf942dc77b376592d9c862cd] :value1) inserted

PUT for key2 [78f825aaa0103319aaa1a30bf4fe3ada] was forwarded to localhost:50002

PUT for key3 [3631578538a2d6ba5879b31a9a42f290] was forwarded to localhost:50002

Found key value pair (key1:value1)

GET for key2 [78f825aaa0103319aaa1a30bf4fe3ada] was forwarded to localhost:50002

(key:value) =(key53[128a85ec696ff60d0cf5cc1b6e1a4805]:value53) inserted

Key Value

------------------------------

key1 value1

key53 value53

------------------------------

The following files contain the basic client server functionality that you can use to build the DHT nodes.

* [dhtmain.c](file:///C:\Users\Siddharth\Study\NCSU_Course\Semester%201\CSC501\Winter_Project\TokenRing\q3\dhtmain.c)
* [client.c](file:///C:\Users\Siddharth\Study\NCSU_Course\Semester%201\CSC501\Winter_Project\TokenRing\q3\client.c)
* [Makefile](file:///C:\Users\Siddharth\Study\NCSU_Course\Semester%201\CSC501\Winter_Project\TokenRing\q3\Makefile)

**Design Overview:**

**Logical Ring:**

First node->Second Node-> ....->Last Node->First Node

**DHT Node:**

Initialization:

> First DHT node that comes up creates file “nodelist”. It adds its hostname:portno entry to the file.

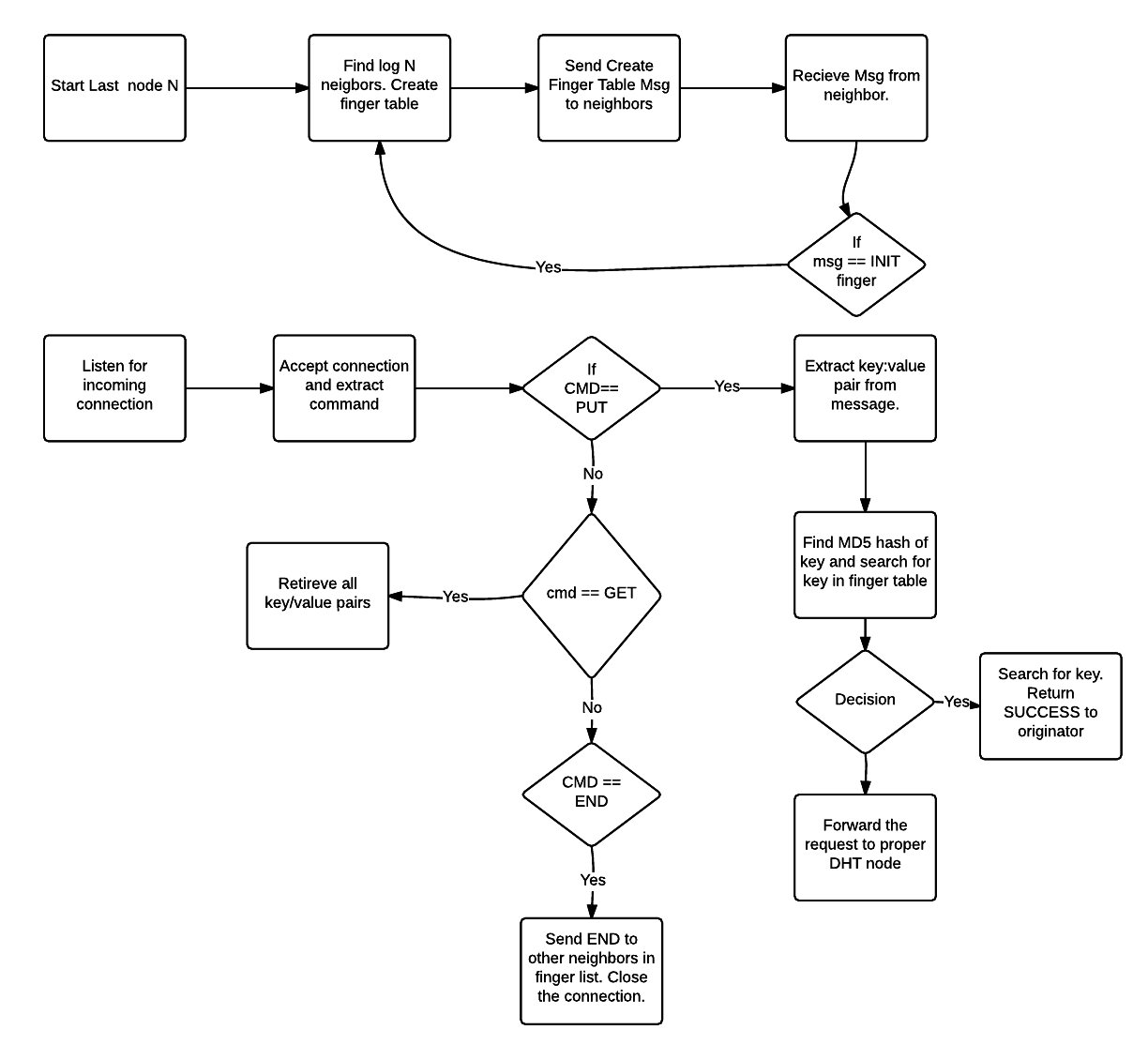
> DHT nodes add its entry for hostname:portno to the nodelist.

>Last DHT node to come up 1) writes to nodelist 2) sends START message to its next neighbours 3) Initializes its finger table.

>On receiving Start 1) DHT node will initialize its finger table. 2) Forward START message to next neighbour.

>When the initiator of START receives start message it will absorb the message.

Flow Chart:



**Client Node:**