Project 2
Simple Key-Value Storage

Team
Ashwin RS
Maithereyi Buddhiraju
Shashank Holavanalli
Shishir Garg
Vineet Aguiar

A. Components designed:

Our group has in general has made the following design components:

1. **Content Provider** – It provides all the functionality such as Chord Creation and Node Join, Insertion, Deletion and Querying the local storage. The Content Provider abstracts the entire functionality from the activity layer above by making available the interfacing methods like insert and query.
2. **Chord Join Function** – This component provides the logic to introduce a new node to the chord and insert it in the correct position. This also takes care of boundary conditions for zero crossings and single and two-node chords.
3. **Lookup Function** – This component provides logic to find the position where the node can be inserted into the chord. Each new message position is verified individually and on receiving the node position and insertion or query is made.

B. Algorithm Description

a. Chord Join Component

5554 is the first emulator to be initialized. All joining nodes first send a join request to the 5554. 5554 checks whether the node comes between itself and successor, each node follows the pseudo code mentioned above. Once the position to insert the node is found, a message is sent to the joiner to update its successor and predecessor. And another message is sent to the node's successor to update their predecessor to the joiner's node value.

b. Lookup for Insert and Query

The lookup function is called by the Content Provider before every Insert and Query operation. The input to the lookup function is the key and the port number of the initiator node. With every new message, the key is identified and the lookup function is called. The lookup function first checks if the key is stored with the node that has called it.

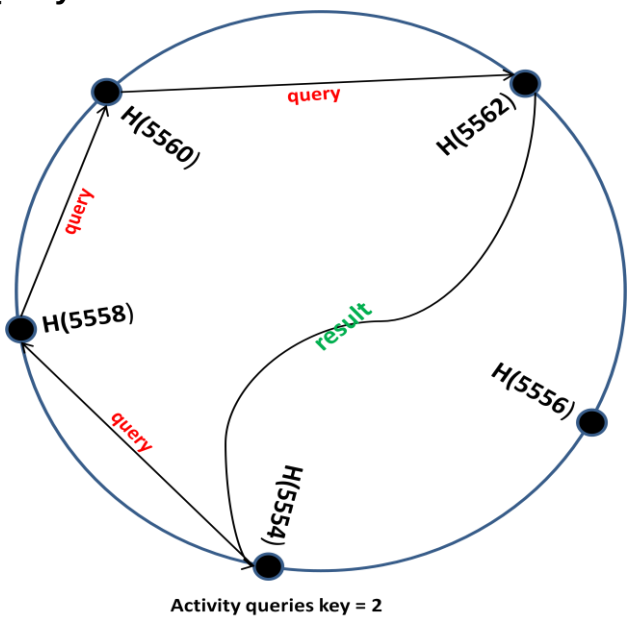
If yes, the lookup function makes the function calls to actually insert or retrieve data from the data storage. In a case where the lookup determines cannot belong to the current node, it passes it on to the next node by means of a socket connection.

C. Pseudo Algorithm:

Chord Join Component	DHT Insert

Every node runs the algorithm mentioned below until the place where to insert the node is found. Here curr = current, succ = successor, joiner, pred = predecessor all are Port numbers.
 	
if hash(curr) < hash(succ) **then**
If hash(joiner) > hash(curr) **&&** hash(joiner) < hash(succ) **then**
// position to insert node is found
joiner node's succ = curr node's succ
joiner node's pred = curr node
curr node's successor's pred = joiner
current node's successor = joiner
else
go to next node and check same condition
else
if hash(joiner) > hash(curr) **||** hash(joiner) < hash(succ) **then**
// position to insert node is found
joiner node's succ = curr node's succ
joiner node's pred = curr node
curr node's succ's pred = joiner
curr node's succ = joiner
	Returns true if node has to insert key and false if key has to be stored in another node
	
	if there is only one node in the network **then**
	return true
	
	if myHash > predecessorHash **then**
	if keyHash > predecessorHash && keyHash <= myHash **then**
	return true
	else
	forward to successor
	return false
	else
	if keyHash > predecessorHash | | keyHash <= myHash **then**
	return true
	else
	forward to successor
	return false

DHT Query



Activity queries key = 2