CSC 573: INTERNET PROTOCOLS

PROJECT #1 REPORT: CHORD-LT

BY

SALIL KANITKAR	**sskanitk**
MAYUR AWAGHADE	**mawagha**
SAGAR SANE	**sssane**
SAGAR NATEKAR	**snateka**

System Requirements:

Programming Environment: C
Compiler: gcc 4.1.2 (the same version is present on EOS machines)
OS: Unix Based
Tested On: VCL Image Name: Linux Lab Machine (Realm RHEnterprise 5): EOS lab machine images

---

Compiling the Code:

1. We have created a Makefile that will generate all the required binaries to run the system.
2. The **sample_RFCs** folder hosts the actual RFCs that will be distributed amongst the peers.
3. Under folder **proj1** , we perform following commands.
    a. **make clean ; make**
    b. **./setup_chord_system.sh** - This shell script creates 10 folders p0, p1 .. p9 which act as sandboxes for each peer to run in. It copies the 50 RFCs we have downloaded in folder **p0** and the chord_peer and chord_client binaries that will be required to start each Peer.
4. We are now ready to start the system.

Running the Code:

- **System Setup / Initial transferring of keys**

1. The system always begins with p0. Start p0 as follows :
   cd p0
   ./chord_peer --start
2. The user will be prompted to select a network interface to connect to. Choose the appropriate network interface using **y/n**.
3. This will start p0 at a well known port and well known IP address.
4. All other peers in the system will start as follows :
   cd p<peer number> (peer number : 1,2,3...,9)
   ./chord_peer <IP address of P0> <Port Number of P0>

5. The user will be prompted to select a network interface to connect to. Choose the appropriate network interface using **y/n**.
6. Perform steps 4 and 5 for all the peers to start up. You can view the messages transferred between the appropriate peers on their respective consoles.

- **Starting a client**

To start a client, enter the following commands
cd p<peer number>
./chord_client

- **Getting the details of Peers in the system from P0**

1. Start a client as given in the section 'Starting a client' above
2. Enter the message type as : **PeerDetails**
3. Enter the well known IP address and port number of P0.
4. The Peer details will be printed at the client's console.

- **Getting the details of RFC List present at a particular Peer**

1. Start a client as given in the section 'Starting a client' above
2. Enter the message type as : **PrintRFCDb**
3. Enter the IP address and Port number of the required peer. (The **PeerDetails** command can be run before this to get the detailed list of IP address and Port number of the Peers present in the system)
4. The RFC List details will be displayed at the appropriate Peer's console.

- **Client Requests an RFC**

1. Start a client as given in the section 'Starting a client' above.
2. Enter the message type as : **FetchRFC**
3. The user will be prompted for, **IP address and Port Number** of the server to be contacted and **the RFC Title and Value** to be requested.
4. The IP Address and port number can be found out by client running the command **PeerDetails** as mentioned above.
5. The list of RFC Titles and their Values in the system is located in **RFCValuesList.txt** in the folder **p0** after the system is setup and all the peers are active in the system**.**
6. To know which RFCs are there at a node, the user can run the **PrintRFCDb** command as mentioned above.

- **Peer leaves in orderly manner**

1. Start a client as given in the section 'Starting a client' above.
2. Enter the message type as : **PeerExit**
3. Enter the IP address and Port number of the peer that will leave. (We have this functionality at the client's end as the Peer will always leave in an orderly fashion)
4. The peer will leave from the system and the appropriate message passing will be displayed on the appropriate peer consoles.

Note: The Appendix section at the end, **Work-flows in Chord System** that has actual examples of all the possible scenarios.

Message Formats:

All the communication (communication between two chord peers and the communication between a chord peer and the chord client) is done via messages that follow a fixed format - which resemble the HTTP packet formats with predefined set of headers and the expected values for all of the header fields.

We will explain each of the message formats used by us and the brief usage of the scenario in which they are used. As mentioned earlier, a more detailed usage of these messages is articulated and section **Work-flows in the Chord-LT System**

- **Message Types:**

Between Peer Servers
RegisterNode
NodeIdentity
FixFingers
GetDb
NodeList
GetRFC
ForwardGet
GetFinger
PutKey
RemoveNode
ExitNow

Between Peer Server and Peer Client
PrintRFCDb
PeerDetails
FetchRFC
PeerExit

- **Message Formats:**

1. RegisterNode:

GET RegisterNode Chord-LT/1.0
IP:<IP_Addr_P0>
Port:<Port_num_P0>

- This message is sent by a new peer joining the system to P0, to let P0 know the IP address and port number on which the new peer is listening.

2. NodeIdentity:

POST NodeIdentity Chord-LT/1.0
chord_id:<chord_id>
successor_id:<chord_id of successor>
pred_id:<chord_id of predecessor>
successor_IP:<IP_Addr of successor>
successor_Port:<Port_num of successor>
pred_IP:<IP_Addr of predecessor>
pred_Port:<Port_num of predecessor>

- In response to "RegisterNode", P0 sends the new peer information about the new peer's chord ID, as well as its successor's and predecessor's chord IDs, IP addresses and respective port numbers.

3. FixFingers:

POST FixFingers Chord-LT/1.0

- When the new peer sends a "RegisterNode" to P0, P0 sends the "FixFingers" message to the appropriate peers in the system to fix their finger tables.

4. GetDb:

GET GetDb Chord-LT/1.0

- This message is a notification to the newly joined peer to allow it to send "GetKey" messages to receive the keys it is responsible for. This message is for synchronization purposes only. The new peer cannot send "GetKey" message to its successor unless it receives the "GetDb" message from P0.

5. GetKey:

GET Getkey Chord-LT/1.0
IP:<IP_Addr>
Port:<Port_num>
Chord-Id:<chord_id>

- This message is sent by the new peer that has joined the system, to its successor for requesting the keys it is now responsible for. The IP address, port number and Chord ID of requesting peer is sent alongwith the message.

6. NodeList:

POST NodeList Chord-LT/1.0
IP: <IP_Addr>
Port:<Port_num>
count:<count of RFCs to be sent to this node>
<RFC1_key>:<RFC1_value>:<RFC1_title>
<RFC2_key>:<RFC2_value>:<RFC2_title>
.

.
<RFCn_key>:<RFCn_value>:<RFCn_title>

- In response to the "GetKey" message, the successor sends the list of RFCs in the form of <RFC Key, RFC Value and RFC title> tuples to the requesting peer, alongwith its IP address and port number. The count of total RFCs sent to the requesting peer is also included with the message.

7. GetRFC:

GET GetRFC Chord-LT/1.0
IP: <IP_Addr>
Port:<Port_num>
RFC-Value:<RFC_value>
Flag:0/1

- This message is passed between any 2 peers, regarding a request for RFC body. This includes initial transferring of keys and final transfer of RFC bodies.

8. ForwardGet:

GET ForwardGet Chord-LT/1.0
IP: <IP_Addr>
Port:<Port_num>
RFC-Value:<RFC_value>

- When a peer gets "FetchRFC", it checks to see if the requested RFC is with him. If it is not found, then this peer sends a "ForwardGet" message to the successor or closest preceeding finger, after checking the finger table.

9. GetFinger

GET GetFinger Chord-LT/1.0
IP: <IP_Addr>
Port: <Port_num>
Finger-val; <Finger-val>

- When new nodes join, it might happen that the successor of successor has changed. Thus, when a peer receives a "FixFingers" message from P0, it invokes "GetFinger" to ask the successor about its successor. If, for the ith finger, this newly received value is the correct successor of successor, then the peer enters this successor of successor in the ith entry of its finger table.

10. PutKey

POST PutKey Chord-LT/1.0
IP: <IP_Addr>
Port:<Port_num>
count:<count of RFCs to be sent to this node>
<RFC1_key>:<RFC1_value>:<RFC1_title>
<RFC2_key>:<RFC2_value>:<RFC2_title>
.
.
<RFCn_key>:<RFCn_value>:<RFCn_title>

- This message is invoked by a peer leaving the system. The RFC list present at the peer, represented in the form of <RFC Key, RFC Value, RFC Title> tuples is passed on to the successor of the peer, alongwith the IP address, port number and count of RFCs passed on.

11. RemoveNode

GET RemoveNode Chord-LT/1.0
chord_id:<chord_id>

IP: <IP_Addr>
Port:<Port_num>

- The successor of the peer which is to leave the system sends this message to P0 to tell it ti remove the peer from its list of peers.

12. ExitNow

POST ExitNow Chord-LT/1.0

- When all keys and RFCs have been transferred from the peer leaving the system to its successor, the successor invokes this message to tell the peer to exit from the system.

13. PrintRFCDb:

GET PrintRFCDb Chord-LT/1.0

- This message is invoked by a chord client for printing the list of RFCs present at a peer, on the console of the respective peer.

14. FetchRFC:

GET FetchRFC <RFC_TITLE> <RFC_Value> Chord-LT/1.0
IP: <IP_Addr>
Port:<Port_num>

- A chord client sends this message to a chord server, requesting a specific RFC identified by RFC_Title and RFC_Value..

15. PeerDetails

GET PeerDetails Chord-LT/1.0
IP:<IP_Addr>
Port:<Port_num>

- This message is invoked by the client to know details about all the peers in the system. It asks for the IP address and port number of P0, and prints out the list of all peers at P0, in the form of <Chord_ID, IP_Address, Port_Number> tuples.

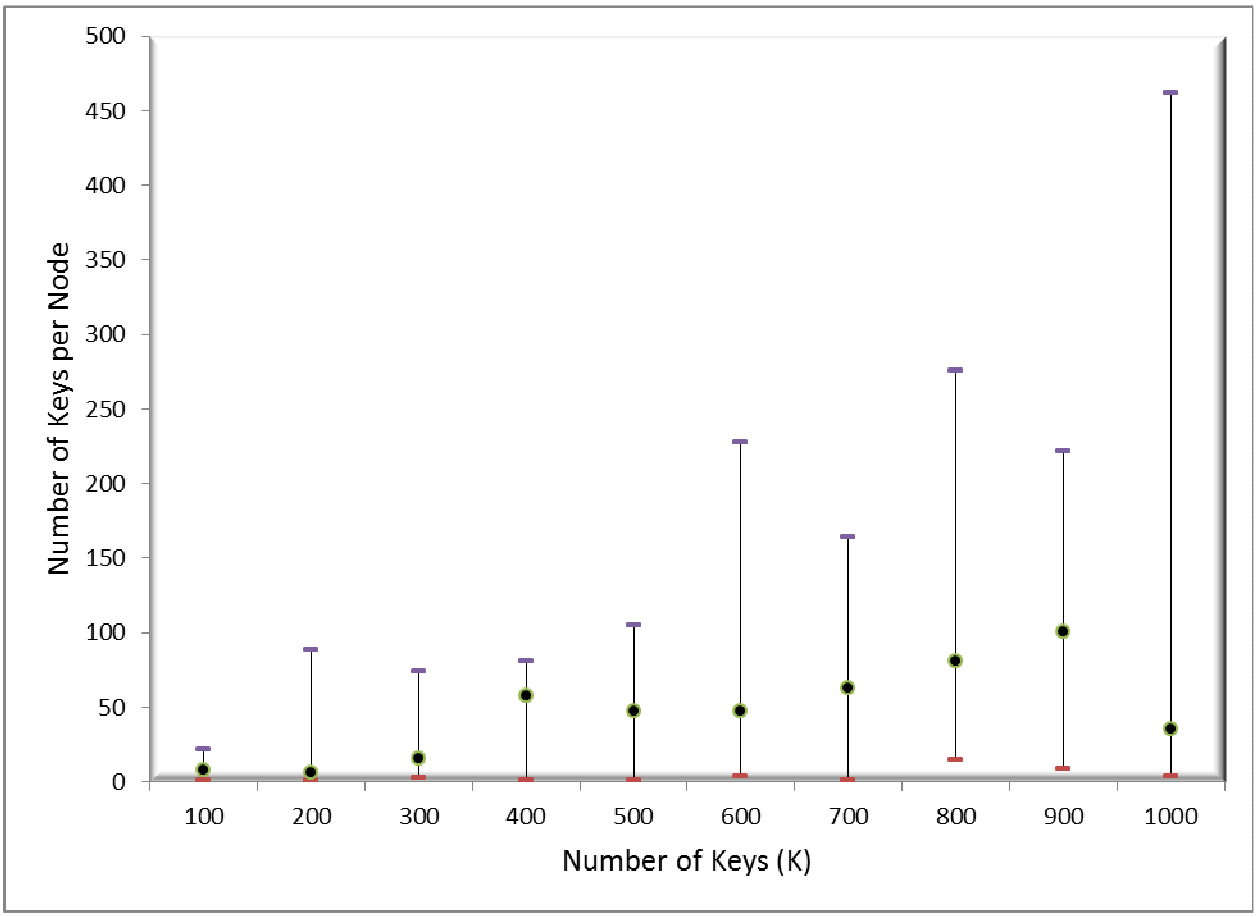
16. PeerExit

GET PeerExit Chord-LT/1.0
IP:<IP_Addr>
Port:<Port_num>

- A chord client sends this message to the peer which has to leave the system. The IP address and port number of the peer is included in the message.

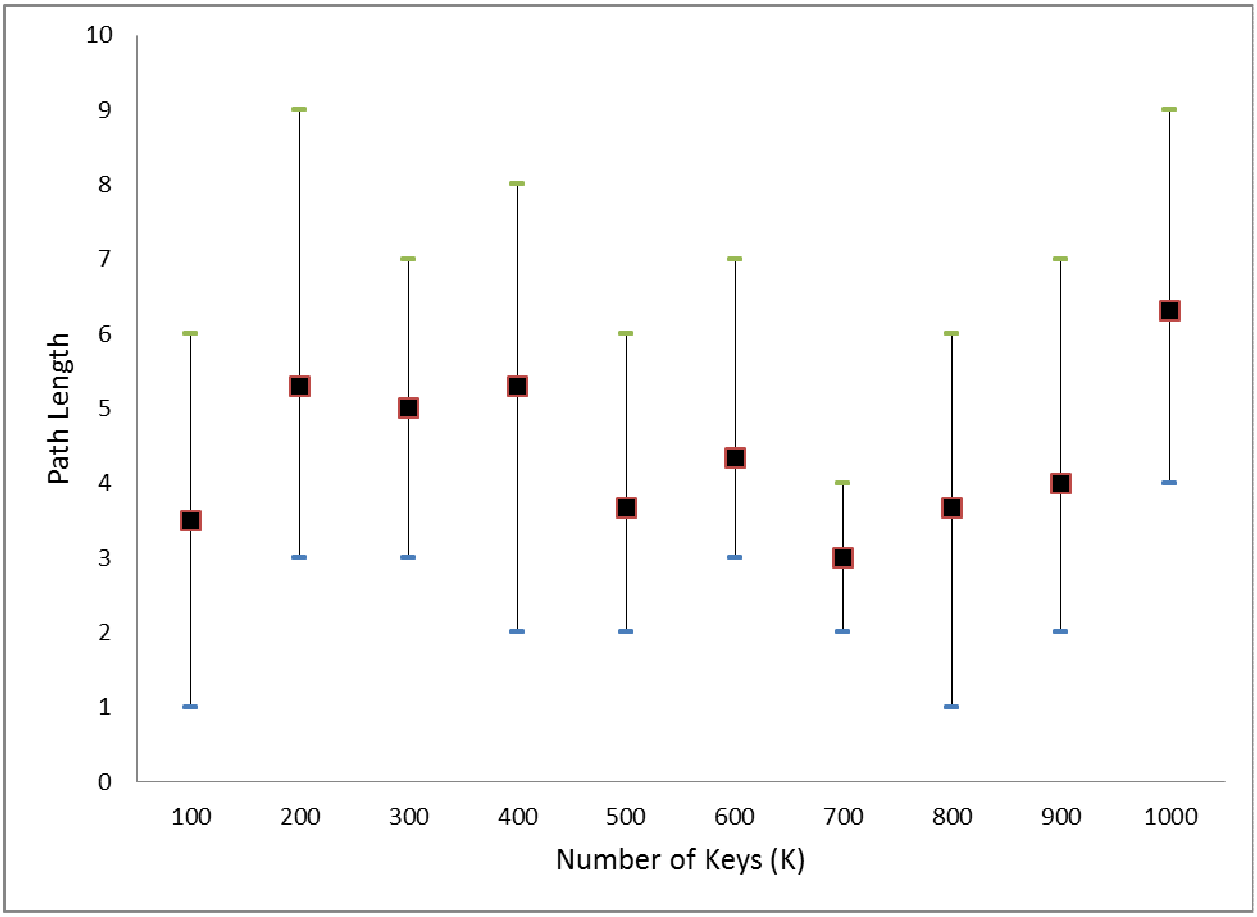
Tasks:

Task 1: Number of Keys per Node v/s Total Number of Keys (K)



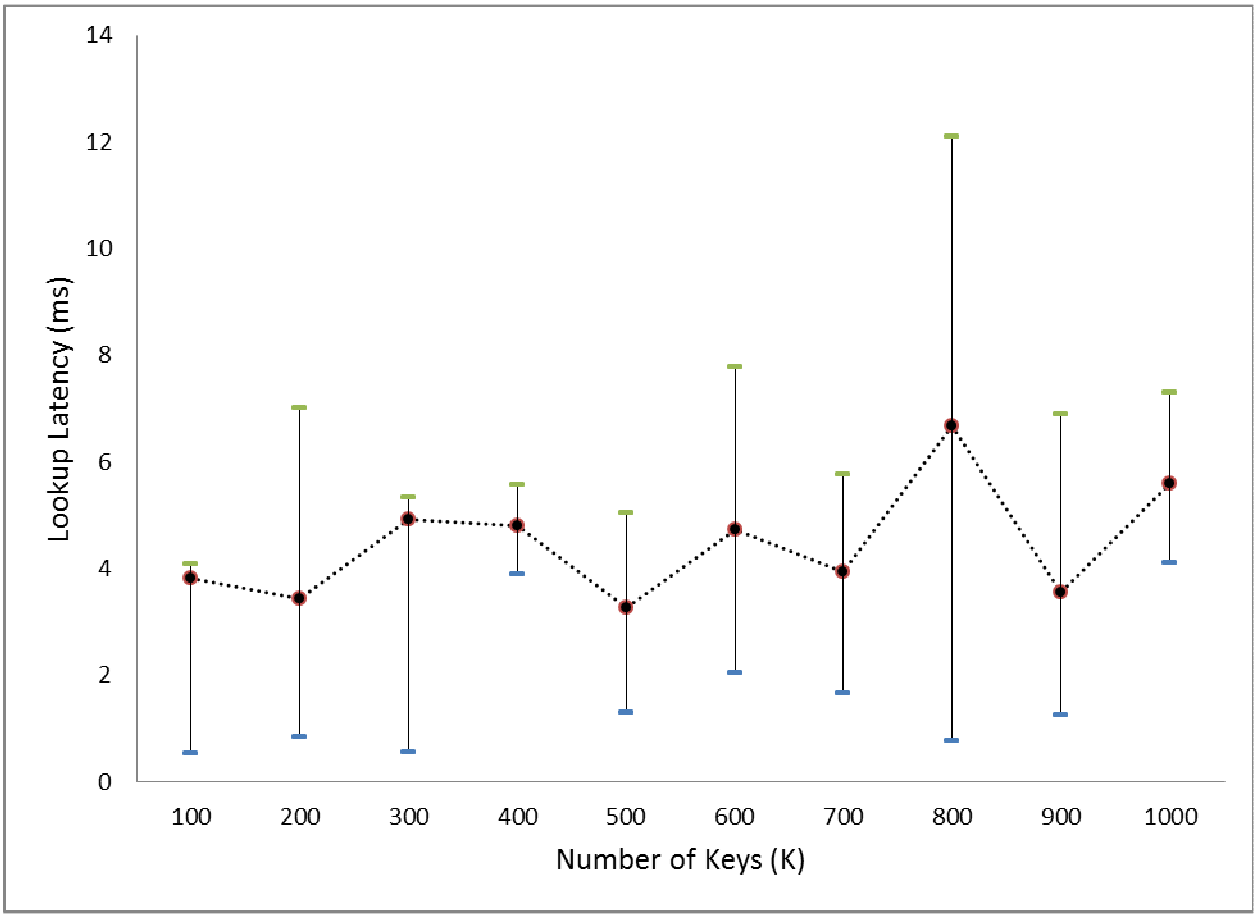
1. The above graph is a plot of the maximum, minimum and median number of keys of each peer against the total number of keys.
2. From the above graph, we can see that the median number of keys against the total number of keys is not exponentially increasing. This is because the number of keys are distributed amongst the peers randomly.
3. The number of keys to be transferred is decided from the random Chord ID assigned to a new node and the Chord ID of its predecessor. Thus, as a new seed is used every time by P0 to generate random Chord ID, we cannot have a uniform distribution of Chord IDs in the ring. Hence, we cannot have the number of keys per node to be uniformly distributed.

Task 2.1 : Path Length v/s Number of Keys (K)



1. The above figure shows a plot of the minimum, maximum and average path length observed for lookups for total number of keys with K = 100 to 1000.
2. We can see from the graph that path length varies from 1 to 7 in general. The average path length found is approximately between 3-5 which is between the result in case of short cuts in finger tables O(log n) and result in case of no shortcuts, O(n).

Task 2.2: Lookup Latency (ms) v/s Number of Keys (K)



1. The above graph shows a plot between the minimum, maximum and average lookup latency v/s the number of keys in the system, K = 100...1000.
2. The data collected is the time taken in from the initial request being sent for an RFC to the time at which the server responsible for that RFC was located. This does not include the time for transferring the actual RFC body.

<u>Appendix:</u> Work-flows in the Chord-LT System

- **Requesting an RFC download**

<output @client>
salil@Salil-Laptop:~/git-repo/IP_P1_ChordLT$ **./chord_client**

========== Chord Client ==========

Available Tpes of Messages
FetchRFC, PrintRFCDb, PeerDetails, PeerExit

Enter "End" to Exit from Chord Client

Enter the Type of the Message: **PeerDetails**
Enter the IP Address of Peer0: **192.168.0.12**
Enter the Port of Peer0: **65400**
peerDetails Msg:
GET PeerDetails Chord-LT/1.0
IP:192.168.0.12
Port:65400
The current list of Peers is:
Chord id: 0, IP Address:192.168.0.12, Port:65400
Chord id: 233, IP Address:192.168.0.12, Port:65435
Chord id: 258, IP Address:192.168.0.12, Port:65421
Chord id: 274, IP Address:192.168.0.12, Port:65427
Chord id: 337, IP Address:192.168.0.12, Port:65486
Chord id: 350, IP Address:192.168.0.12, Port:65415
Chord id: 762, IP Address:192.168.0.12, Port:65459
Chord id: 899, IP Address:192.168.0.12, Port:65492

Available Tpes of Messages
FetchRFC, PrintRFCDb, PeerDetails, PeerExit

Enter "End" to Exit from Chord Client

Enter the Type of the Message: **PrintRFCDb**
Enter the IP Address of the Peer to connect to: **192.168.0.12**
Enter the Port of the above Peer to connect to: **65400**

Available Tpes of Messages
FetchRFC, PrintRFCDb, PeerDetails, PeerExit

Enter "End" to Exit from Chord Client

Enter the Type of the Message: **FetchRFC**
Enter the IP Address of the Peer to connect to: **192.168.0.12**
Enter the Port of the above Peer to connect to: **65486**
Enter the RFC Title of the RFC to be fetched: **1RFC_502.txt**
Enter the RFC value of the RFC to be fetched: **3242**
FetchRFC Msg:
GET FetchRFC 1RFC_502.txt 3242 Chord-LT/1.0
IP:192.168.0.12
Port:65486
The latency for the Lookup operation in Microseconds is: 3368

Available Tpes of Messages
FetchRFC, PrintRFCDb, PeerDetails, PeerExit

Enter "End" to Exit from Chord Client

Enter the Type of the Message: **End**
Chord Cient Exiting ...
</output @client>

<output @peer to whom the request was aimed>
Received following Msg:
GET FetchRFC 1RFC_502.txt 3242 Chord-LT/1.0
IP:192.168.0.12
Port:65486

The RFC requested is: Key:170 Value 3242 Title: 1RFC_502.txt
Forwards are required to fetch this RFC!!!
The Requested RFC is at Node 233 192.168.0.12 65435
File to be SENT: 1RFC_502.txt
Sending end line of RFC
Node 337 is done sending an RFC to a Client.
</output @peer to whom the request was aimed>

<output @forwarding-node1>
Received following Msg:
GET ForwardGet Chord-LT/1.0
IP:192.168.0.12
Portnum:65426
RFC-value:3242
ForwardGet received! 350 762 3242 170
ForwardGet Sent by Node 350 to Node 762
</output @forwarding-node1>

<output @forwarding-node2>
Received following Msg:
GET ForwardGet Chord-LT/1.0
IP:192.168.0.12
Portnum:65426
RFC-value:3242
ForwardGet received! 762 899 3242 170
ForwardGet Sent by Node 762 to Node 899
</output @forwarding-node2>

<output @forwarding-node3>
Received following Msg:
GET ForwardGet Chord-LT/1.0
IP:192.168.0.12
Portnum:65426
RFC-value:3242
ForwardGet received! 899 0 3242 170
ForwardGet Sent by Node 899 to Node 0
</output @forwarding-node3>

<output @forwarding-node4>
Received following Msg:
GET ForwardGet Chord-LT/1.0
IP:192.168.0.12
Portnum:65426
RFC-value:3242
ForwardGet received! 0 233 3242 170
Sending 233:192.168.0.12:65435:

</output @forwarding-node4>

- **Print RFC Database at a Node**

  <output @ Peer Client>
  ========== Chord Client ==========

  Available Tpes of Messages
  FetchRFC, PrintRFCDb, PeerDetails, PeerExit

  Enter "End" to Exit from Chord Client

  Enter the Type of the Message: **PrintRFCDb**
  Enter the IP Address of the Peer to connect to: **192.168.2.3**
  Enter the Port of the above Peer to connect to: **65486**
  </output @ Peer Client>

  <output @ Peer Server with Port 65486 and IP 192.168.2.3>
  Received following Msg:
  GET PrintRFCDb Chord-LT/1.0
  The RFC db at this node

  The RFC Database:
  Key : 74 Value:3146 Title:43RFC_992.txt
  Key : 109 Value:3181 Title:15RFC_516.txt
  Key : 116 Value:3188 Title:17RFC_518.txt
  Key : 119 Value:3191 Title:6RFC_507.txt
  Key : 121 Value:4217 Title:8RFC_509.txt
  Key : 139 Value:3211 Title:14RFC_515.txt
  Key : 202 Value:1226 Title:3RFC_504.txt
  Key : 209 Value:2257 Title:24RFC_526.txt
  Key : 223 Value:5343 Title:39RFC_988.txt
  Key : 230 Value:2278 Title:27RFC_973.txt
  Number of RFCs ar this Node: 10
  </output @ Peer Server with Port 65486 and IP 192.168.2.3>

- **Get Details of Current Peers**

  <output @ Peer Client>
  ========== Chord Client ==========

  Available Types of Messages
  FetchRFC, PrintRFCDb, PeerDetails, PeerExit

  Enter "End" to Exit from Chord Client

  Enter the Type of the Message: **PeerDetails**
  Enter the IP Address of Peer0: **192.168.2.2**
  Enter the Port of Peer0: **65400**
  peerDetails Msg:
  GET PeerDetails Chord-LT/1.0
  IP:192.168.2.2
  Port:65400
  The current list of Peers is:
  Chord id: 0, IP Address:192.168.2.2, Port:65400
  Chord id: 151, IP Address:192.168.2.2, Port:65421
  Chord id: 374, IP Address:192.168.2.2, Port:65435
  Chord id: 419, IP Address:192.168.2.2, Port:65415
  Chord id: 604, IP Address:192.168.2.2, Port:65486

Chord id: 835, IP Address:192.168.2.2, Port:65492
</output @ Peer Client>

<output @ Peer 0>
Received following Msg:
GET PeerDetails Chord-LT/1.0
IP:192.168.2.2
Port:65400
In Get PeerDetails
Current Peer 0 in the list is: Chord_id:0
IP:192.168.2.2
Port:65400
Current Peer 1 in the list is: Chord_id:151
IP:192.168.2.2
Port:65421
Current Peer 2 in the list is: Chord_id:374
IP:192.168.2.2
Port:65435
Current Peer 3 in the list is: Chord_id:419
IP:192.168.2.2
Port:65415
Current Peer 4 in the list is: Chord_id:604
IP:192.168.2.2
Port:65486
Current Peer 5 in the list is: Chord_id:835
IP:192.168.2.2
Port:65492
Message to be sent is POST PeerDetais Chord-LT/1.0
Chord_id:0
IP:192.168.2.2
Port:65400
Chord_id:151
IP:192.168.2.2
Port:65421
Chord_id:374
IP:192.168.2.2
Port:65435
Chord_id:419
IP:192.168.2.2
Port:65415
Chord_id:604
IP:192.168.2.2
Port:65486
Chord_id:835
IP:192.168.2.2
Port:65492
</output @ Peer 0>

- **Adding a new Node**
  Let us assume the following ring is already in place. So as shown below, 5 nodes make up the ring.
  P0 prints the ring as shown below -

  <output @ Peer 0>
  The P2P System Details

  Chord_Id:0 IP:192.168.0.11 Port:65400 Successor:225 192.168.0.11 65486 Pred:765 192.168.0.11 65421 p0
  Chord_Id:225 IP:192.168.0.11 Port:65486 Successor:303 192.168.0.11 65435 Pred:0 192.168.0.11 65400 p1
  Chord_Id:303 IP:192.168.0.11 Port:65435 Successor:390 192.168.0.11 65492 Pred:225 192.168.0.11 65486 p3
  Chord_Id:390 IP:192.168.0.11 Port:65492 Successor:525 192.168.0.11 65415 Pred:303 192.168.0.11 65435 p4
  Chord_Id:525 IP:192.168.0.11 Port:65415 Successor:765 192.168.0.11 65421 Pred:390 192.168.0.11 65492 p2

Chord_Id:765 IP:192.168.0.11 Port:65421 Successor:0 192.168.0.11 65400 Pred:525 192.168.0.11 65415 p5
</output @ Peer 0>

Now, when a new node, say Node P6 wants to join - following sequence of actions happen in our system -

1. Start up the new peer by providing IP address and Portnumber of P0 as command line arguments.

<output @p6>
~/git-repo/IP_P1_ChordLT/p6$ **./chord_peer 192.168.0.11 65400**
A Peer Node is startig up.....
P0s Parameters: 192.168.0.11 65400

Do you want the chord_peer to bind to wlan0 interface?(y/n): **y**

My public interface and IP is: wlan0 192.168.0.11
</output @p6>

2. The new node first tests if the connection to P0 is active.
If P0 is not up yet, the new peer will not start and it will exit out gracefully by printing relevant message.
Please see function "test_if_P0_alive()" in chord_util.c

3. After confirming that P0 is alive, the new node P6 will send "RegisterNode" message to P0.
P0 will generate a random chord_id for this node. Since P0 is also aware of other nodes alive in the system right now, P0 can make sure that the chord_id assigned to every peers are distinct.

<output @p6>
RegisterNode Message sent to P0
</output @p6>

4. P0 will call "setup_successor" and it will adjust its global data structure - which maintains all the current active chord peers. Once the data structure is updated, P0 knows who the successor and predecessor of this newly joined node are going to be. Also, it will determine which nodes need to update their successor and predecessors as a result of this new node addition and it will send NodeIdentity to those nodes.
The nodes that receive NodeIdentity perform "fix_fingers"

Once all the affected nodes have received “NodeIdentity”, it mean that the ring vi successor and predecessor pointers is set up again.
Then every node will perform fix-fingers so that finger tables at every node are updated.

Once these actions are done, P will send "NodeIdentity" message back to the Node P6.
NodeIdentity thus contains the chord_id to be assigned for Node P6, its successor and predecessor.

<output @p0>
Received following Msg:
GET RegisterNode Chord-LT/1.0
IP:192.168.0.11
Port:65427
1) Send NodeIdentiy msg to Node 225
1) Send NodeIdentiy msg to Node 303
</output @p0>

<output @p1>
Received following Msg:
POST NodeIdentity Chord-LT/1.0
chord_id:225
successor_id:244
successor_IP:192.168.0.11
successor_Port:65427

pred.chord_id:0
pred_IP:192.168.0.11
pred_Port:65400

Received the NodeIdentity Message from P0
Chord_Id:225 IP:192.168.0.11 Port:65486 Successor:244 192.168.0.11 65427 Pred:0 192.168.0.11 65400

Received following Msg:
POST FixFingers Chord-LT/1.0
Calling fix_fingers at Node 225...

The Finger Table At Node 225
Finger 1) Start:226 Successor:244
Finger 2) Start:227 Successor:244
Finger 3) Start:229 Successor:244
</output @p1>

<output @p3>
Received following Msg:
POST NodeIdentity Chord-LT/1.0
chord_id:303
successor_id:390
successor_IP:192.168.0.11
successor_Port:65492
pred.chord_id:244
pred_IP:192.168.0.11
pred_Port:65427

Received the NodeIdentity Message from P0
Chord_Id:303 IP:192.168.0.11 Port:65435 Successor:390 192.168.0.11 65492 Pred:244 192.168.0.11 65427

Received following Msg:
POST FixFingers Chord-LT/1.0
Calling fix_fingers at Node 303...

The Finger Table At Node 303
Finger 1) Start:304 Successor:390
Finger 2) Start:305 Successor:390
Finger 3) Start:307 Successor:390
</output @p3>

5. Upon receiving NodeIdentity message, P6 now knows who its successor and predecessor are.
From the ring status displayed above, the successor of node P6, whose chord is 244, is node with chord_id 303.
This successor has the RFCs for which node P6 is going to be responsible for.
However P6 does not yet know the exact values, keys and titles of these RFCs.
So node P6 sends a "GetKey" message to node with chord_id "303".

After the "fix_fingers" at Node P6, it will send GetDb to its successor(p3).
Successor will find out the RFCs corresponding to this node. This message is called "NodeList"
The node p6, upon receiving NodeList, will send GetRFC for each of the RFC names received in NodeList.
And then finally, the node join is complete.

<output @p6>
Received following Msg:
POST NodeIdentity Chord-LT/1.0

chord_id:244
successor_id:303
successor_IP:192.168.0.11
successor_Port:65435
pred_id:225
pred_IP:192.168.0.11
pred_Port:65486

Received the NodeIdentity Message from P0
Chord_Id:244 IP:192.168.0.11 Port:65427 Successor:303 192.168.0.11 65435 Pred:225 192.168.0.11 65486

Received following Msg:
POST FixFingers Chord-LT/1.0
Calling fix_fingers at Node 244...

The Finger Table At Node 244
Finger 1) Start:245 Successor:303
Finger 2) Start:246 Successor:303
Finger 3) Start:248 Successor:303

Received following Msg:
GET GetDb Chord-LT/1.0
GetKey Message sent from Peer with ChordID 244 to its successor with ChordID 303

Received following Msg:
POST NodeList Chord-LT/1.0
IP:192.168.0.11
Portnum:65435
count:4
228:3300:20RFC_521.txt
231:1255:36RFC_983.txt
233:4329:46RFC_996.txt
238:3310:40RFC_989.txt
Request for the actual RFC Files.....
Fetching RFC with key: 228
Fetching RFC with key: 231
Fetching RFC with key: 233
Fetching RFC with key: 238
RFC Db LL created!
Chord_Id:244 IP:192.168.0.11 Port:65427 Successor:303 192.168.0.11 65435 Pred:225 192.168.0.11 65486
</output @p6>

<output @p3>
Received following Msg:
GET GetDb Chord-LT/1.0

Received following Msg:
GET GetKey Chord-LT/1.0
IP:192.168.0.11
Port:65427
Chord-Id:244
NodeList Message sent: 244 to 303

Received following Msg:
GET GetRFC Chord-LT/1.0
IP:192.168.0.11

Port:65435
RFC-Value:3300
Flag:0
Sending end line of RFC
Node 303 is done sending an RFC to requesting server.

Received following Msg:
GET GetRFC Chord-LT/1.0
IP:192.168.0.11
Port:65435
RFC-Value:1255
Flag:0
Sending end line of RFC
Node 303 is done sending an RFC to requesting server.

Received following Msg:
GET GetRFC Chord-LT/1.0
IP:192.168.0.11
Port:65435
RFC-Value:4329
Flag:0
Sending end line of RFC
Node 303 is done sending an RFC to requesting server.

Received following Msg:
GET GetRFC Chord-LT/1.0
IP:192.168.0.11
Port:65435
RFC-Value:3310
Flag:0
Sending end line of RFC
Node 303 is done sending an RFC to requesting server.
</output @p3>

- **Removing a Node**

<output @ Peer Client>
========== Chord Client ==========

Available Tpes of Messages
FetchRFC, PrintRFCDb, PeerDetails, PeerExit

Enter "End" to Exit from Chord Client

Enter the Type of the Message: **PeerExit**
Enter the IP Address of the Peer leaving the System: **192.168.2.2**
Enter the Port of the above Peer leaving the system: **65421**

PeerExit Msg:
GET PeerExit Chord-LT/1.0
IP:192.168.2.2
Port:65421
<output @ Peer Client>

When the node to be removed (P_R) gets the PeerExit message, it has to transfer his RFCs to its successor before exiting. For this, P_R sends a PutKey message to its successor. PutKey contains list of all the RFC titles for which P_R is responsible.

<output @ Peer to be removed (IP: 192.168.2.2 Port: 65421)>
Received following Msg:
GET PeerExit Chord-LT/1.0
IP:192.168.2.2
Port:65421
PutKey Message sent: to the successor 419 by leaving node 151
</output @ Peer to be removed (IP: 192.168.2.2 Port: 65421)>

When the successor of P_R gets the PutKey message, it requests the corresponding RFC files from P_R. After receiving all the RFCs, we no more need P_R and hence ExitNow message is sent to P_R. after which P_R exits. successor of P_R then sends a RemoveNode message to P0 so that P0 knows P_R is exiting and updates its own data structures.

<output @ Successor of Peer P_R>
Received following Msg:
POST PutKey Chord-LT/1.0
IP:192.168.2.2
Portnum:65421
count:6
30:5150:134RFC_793_TCP_220.txt
51:2099:12RFC_793_TCP_110.txt
75:1099:101RFC_793_TCP_191.txt
102:3174:133RFC_793_TCP_21.txt
130:4226:137RFC_793_TCP_223.txt
149:3221:113RFC_793_TCP_201.txt
Request for the actual RFC Files.....
Fetching RFC with key: 30
Fetching RFC with key: 51
Fetching RFC with key: 75
Fetching RFC with key: 102
Fetching RFC with key: 130
Fetching RFC with key: 149
RFC Db LL created!
Chord_Id:419 IP:192.168.2.2 Port:65415 Successor:604 192.168.2.2 65486 Pred:151 192.168.2.2 65421
RemoveNode Message sent to P0 is
GET RemoveNode Chord-LT/1.0
Chord id:151
IP:192.168.2.2
Portnum:65421
</output @ Successor of Peer P_R>

<output @ Peer P_R>
Received following Msg:
GET GetRFC Chord-LT/1.0
IP:192.168.2.2
Port:65421
RFC-Value:5150

Flag:1
Sending end line of RFC
Node 151 is done sending an RFC to requesting server.

Received following Msg:
GET GetRFC Chord-LT/1.0
IP:192.168.2.2
Port:65421
RFC-Value:2099
Flag:1
Sending end line of RFC
Node 151 is done sending an RFC to requesting server.

Received following Msg:
GET GetRFC Chord-LT/1.0
IP:192.168.2.2
Port:65421
RFC-Value:1099
Flag:1
Sending end line of RFC
Node 151 is done sending an RFC to requesting server.

Received following Msg:
GET GetRFC Chord-LT/1.0
IP:192.168.2.2
Port:65421
RFC-Value:3174
Flag:1
Sending end line of RFC
Node 151 is done sending an RFC to requesting server.

Received following Msg:
GET GetRFC Chord-LT/1.0
IP:192.168.2.2
Port:65421
RFC-Value:4226
Flag:1
Sending end line of RFC
Node 151 is done sending an RFC to requesting server.

Received following Msg:
GET GetRFC Chord-LT/1.0
IP:192.168.2.2
Port:65421

RFC-Value:3221
Flag:1
Sending end line of RFC
Node 151 is done sending an RFC to requesting server.

Received following Msg:
POST ExitNow Chord-LT/1.0The removed node is exiting
</output @ Peer P_R>

When P0 receives the RemoveNode message, it updates the list of peers to remove P_R from it. It also reforms the ring by pointing successor and predecessor of P_R to each other. It then send NodeIdentity and FixFingers message to successor and predecessor of P_R so that they changes in P0 are propogated in the system.

<output @ P0>
Received following Msg:
GET RemoveNode Chord-LT/1.0
Chord id:151
IP:192.168.2.2
Portnum:65421
Send NodeIdentity msg to Successor of deleted node
Peer with chord id 151 removed
Updated peer List is:
Chord_Id:0 IP:192.168.2.2 Port:65400 Successor:419 192.168.2.2 65415 Pred:835 192.168.2.2 65492
Chord_Id:419 IP:192.168.2.2 Port:65415 Successor:604 192.168.2.2 65486 Pred:0 192.168.2.2 65400
Chord_Id:604 IP:192.168.2.2 Port:65486 Successor:835 192.168.2.2 65492 Pred:419 192.168.2.2 65415
Chord_Id:835 IP:192.168.2.2 Port:65492 Successor:0 192.168.2.2 65400 Pred:604 192.168.2.2 65486
</output @ P0>

Note: We don't need to send NodeIdentity to predecessor of node 151, since predecessor is P0 itself.

<output @ Successor of P_R >

Received the NodeIdentity Message from P0
Chord_Id:419 IP:192.168.2.2 Port:65415 Successor:604 192.168.2.2 65486 Pred:0 192.168.2.2 65400

Received following Msg:
POST FixFingers Chord-LT/1.0
Calling fix_fingers at Node 419...

The Finger Table At Node 419
Finger 1) Start:420 Successor:604
Finger 2) Start:421 Successor:604
Finger 3) Start:423 Successor:604

<output @ Successor of P_R >